

WEB TECHNOLOGIES (A60512)

COURSE DESCRIPTION

| | |
|-----------------------|-----------------------------------|
| Course Title | Web Technologies |
| Course Code | A60522 |
| Regulation | R13-JNTUH |
| Course Faculty | Mr P. Chiranjeevi Asst. Professor |

COURSE OVERVIEW:

This course introduces intermediate to advanced web design techniques. Topics include customer expectations, advanced markup language, multimedia technologies, usability and accessibility practices, and techniques for the evaluation of web design. Upon completion, students should be able to employ advanced design techniques to create high impact and highly functional websites.

COURSE OBJECTIVES:

- To introduce PHP language for server side scripting.
- To introduce XML and processing of XML data with Java.
- To introduce Server side programming with Java servlets and JSP.
- To introduce Client side scripting with Java Script and AJAX

COURSE OUTCOMES:

- Gain knowledge of client side scripting, validation of forms and AJAX programming
- Have Understanding of server side scripting with PHP language
- Have Understanding of what is XML and How to parse and use XML data with Java
- To introduce Server side programming with Java Servlets and JSP

SYLLABUS:

UNIT-1

Introduction PHP: Declaring Variables, data types, arrays, strings, operators, expressions, control structures, functions, reading data from web form controls like text boxes, radio buttons, lists etc., Handling file uploads, connecting to database(MYSQL reference), executing simple queries, handling results, Handling sessions and cookies.

File Handling in PHP: File operations opening, closing, reading, writing, appending, deleting etc. on text and binary files, listing directories.

UNIT-2

XML: Introduction to XML, defining XML tags, their attributes and values, Document type definition, XML schemas, document object model, XHTML. **Parsing XML Data:** DOM and SAX parsers in java.

UNIT-3:

Introduction to Servlets: Common Gateway Interface(CGI), lifecycle of a Servlets, deploying Servlets, the servlet API, reading servlet parameters, reading initialization parameters, handling http requests and responses, using cookies and sessions, connection to a database using JDBC.

UNIT -4:

Introduction to JSP: the autonomy of a JSP page, JSP processing, Declarations, Directives, expressions, code snippets, implicit objects, using beans in JSP pages, using cookies and session for session tracking, connecting to a database in JSP.

UNIT-5

Client side scripting: Introduction to java script: java script language, declaring variables, scope of variables, functions, event handlers(on click and on submit etc.), document object model, form validation, simple AJAX application

TEXT BOOKS:

1. Web Technologies , Uttam K Roy, Oxford University Press
2. The Complete Reference PHP – Steven Holzer, Tata McGraw-Hill.

REFERENCE BOOKS:

1. Web Programming , building internet applications, Chris Bates 2 nd edition, Wiley Dreamtech
2. Java Server Pages – Hans Bergsten, SPD O'Reilly.
3. Java Script, D.Flanagan, O'Reilly, SPD.
4. Beginning Web Programming – Jon Duckett WROX.
5. Programming world wide web, R.W. Sebesta, Fourth Edition, Pearson.
6. Internet and World Wide Web – How to program, Dietel and Nieto, Pearson.

OBJECTIVE QUESTIONS

UNIT-I

1. What does PHP stand for?

- i) Personal Home Page ii) Hypertext Preprocessor
- iii) Pretext Hypertext Processor iv) Preprocessor Home Page
- A. Both i) and iii)
- B. Both ii) and iv)
- C. Only ii)
- D. Both i) and ii)

Answer: D

2. A PHP script should start with ____ and end with ____:

- A. < php >
- B. < ? php ?>
- C. <? ?>
- D. <?php ?>

Answer: C or D

3. Which of the following php statement/statements will store 111 in variable num?

- i) int \$num = 111; ii) int mum = 111; iii) \$num = 111; iv) 111 = \$num;
- A. Both i) and ii)
- B. All of the mentioned.
- C. Only iii)
- D. Only i)

Answer: C

4. What will be the output of the following php code?

```
<? php
$num=1;
$num1=2;
print$num."+".$num1;
?>
```

- A. 3
- B. 1+2
- C. 1.+2
- D. Error

Answer: B

5. What will be the output of the following PHP code?

```
<?php
$color="maroon";
$var=$color[2];
echo"$var";
?>
```

- a)ab)Errorc)\$vard) r

Answer: D

6. What will be the output of the following code?

```
<?php
```

```

function track(){
static $count=0;
$count++;
echo$count;
}
track();
track();
track();
?>

```

- A. 123
- B. 111
- C. 000
- D. 011

Answer: A

7. What is the value of \$a and \$b after the function call?

```

<?php
function doSomething(&$arg){
$return=$arg;
$arg+=1;
return$return;
}
$a=3;
$b= doSomething($a);
?>

```

- A. a is 3 and b is 4
- B. a is 4 and b is 3
- C. Both are 3.
- D. Both are 4.

Answer: B

8. Which one of the following functions can be used to compress a string?

- A. zip compress()
- B. zip()
- C. compress()
- D. gzcompress()

Answer: D

9. Which one of the following PHP function is used to determine a file's last access time?

- A. fileltime()
- B. filectime()
- C. fileatime()
- D. filetime()

Answer: C

10. Which directive determines whether PHP scripts on the server can accept file uploads?

- A. file_uploads
- B. file_upload

- C. file_input
- D. file_intake

Answer: A

11. If you want to temporarily store uploaded files in the /tmp/phpuploads/ directory, which one of the following statement will you use?

- A. upload_tmp_dir "/tmp/phpuploads/ directory"
- B. upload_dir "/tmp/phpuploads/ directory"
- C. upload_temp_dir "/tmp/phpuploads/ directory"
- D. Dupload_temp_director "/tmp/phpuploads/ directory"

Answer: A

12. Which one of the following databases has PHP supported almost since the beginning?

- A. Oracle Database
- B. SQL
- C. SQL+
- D. MySQL

Answer: D

13. Which one of the following statements can be used to select the database

- A. \$mysqli=select_db('databasename');
- B. mysqli=select_db('databasename');
- C. mysqli->select_db('databasename');
- D. \$mysqli->select_db('databasename');

Answer: D

14. Which one of the following methods can be used diagnose and display information about a MySQL connection error?

- A. connect_errno()
- B. connect_error()
- C. mysqli_connect_errno()
- D. mysqli_connect_error()

Answer: C

14. Which one of the following is the very first task executed by a session enabled page?

- A. Delete the previous session
- B. Start a new session
- C. Check whether a valid session exists
- D. Handle the session

Answer: C

15. Which one of the following function is used to start a session?

- A. start_session()
- B. session_start()
- C. session_begin()
- D. begin_session()

Answer: B

16. The session_start() function must appear..

- A. after the html tag
- B. after the body tag

- C. before the body tag
- D. before the html tag

Answer: D

17. Which one of the following function is capable of reading a file into a string variable?

- A. file_contents()
- B. file_get_contents()
- C. file_content()
- D. file_get_content()

Answer: B

18. Which one of the following methods is responsible for sending the query to the database?

- A. query()
- B. send_query()
- C. sendquery()
- D. query_send()

Answer: A

19. Which one of the following method is used to retrieve the number of rows affected by an INSERT, UPDATE, or DELETE query?

- A. num_rows()
- B. affected_rows()
- C. changed_rows()
- D. new_rows()

Answer: B

20. Which of the following methods is used to execute the statement after the parameters have been bound?

- A. bind_param()
- B. bind_result()
- C. bound_param()
- D. bound_result()

Answer: A

UNIT-II

1. What does XML stand for?

- A. eXtra Modern Link
- B. eXtensible Markup Language
- C. Example Markup Language
- D. X-Markup Language

Answer: B

2. What is the correct syntax of the declaration which defines the XML version?

- A. <xml version="A.0" />
- B. <?xml version="A.0"?>
- C. <?xml version="A.0" />
- D. None of the above

Answer: B

3. Which statement is true?

- A. All the statements are true
- B. All XML elements must have a closing tag
- C. All XML elements must be lower case
- D. All XML documents must have a DTD

Answer: B

4. Is it easier to process XML than HTML?

- A. Yes
- B. No
- C. Sometimes
- D. Cant say

Answer: A

5. Which of the following programs support XML or XML applications?

- A. Internet Explorer 5.5
- B. Netscape D.7
- C. RealPlayer.
- D. both A and B

Answer: D

6. Kind of Parsers are

- A. Ans: C well-formed
- B. well-documented
- C. non-validating and validating
- D. none of the above

Answer: C

7. Well formed XML document means

- A. it contains a root element
- B. it contain an element
- C. it contains one or more elements
- D. must contain one or more elements and root element must contain all other elements

Answer: D

8. Comment in XML document is given by

- A. <?-- -->
- B. <!-- --!>
- C. <!-- -->
- D. </-- -- >

Answer: C

9. When processing an output XML, "new line" symbols

- A. Are copied into output "as is", i.e. "CR+LF" for Windows, CR for Macintosh, LF for Unix.
- B. are converted to single LF symbol
- C. are converted to single CR symbol
- D. are discarded

Answer: D

10. Which of the following strings are correct XML names?

- A. _myElement

- B. my Element
- C. #myElement
- D. None of the above

Answer: D

11. Valid XML document means (most appropriate)

- A. the document has root element
- B. the document contains atleast one or more root element
- C. the XML document has DTD associated with it & it complies with that DTD
- D. Each element must nest inside any enclosing element property

Answer: C

12. XML uses the features of

- A. HTML
- B. XHTML
- C. VML
- D. SGML

Answer: D

13. there is a way of describing XML data, how?

- A. XML uses a DTD to describe the data
- B. XML uses XSL to describe data
- C. XML uses a description node to describe data
- D. Both A and C

Answer: D

13. Which of the following XML documents are well-formed?

- A. <firstElement>some text goes here <secondElement>another text goes here</secondElement>
</firstElement>
- B. <firstElement>some text goes here</firstElement>
<secondElement> another text goes here</secondElement>
- C. <firstElement>some text goes here
<secondElement> another text goes here</firstElement>
</secondElement>
- D. </firstElement>some text goes here
</secondElement>another text goes here
<firstElement>

Answer: A

14. Which of the following XML fragments are well-formed?

- A. <myElement myAttribute="someValue"/>
- B. <myElement myAttribute=someValue/>
- C. <myElement myAttribute='someValue'>
- D. <myElement myAttribute="someValue' />

Answer: A

15. How can we make attributes have multiple values:

- A. <myElement myAttribute="value1 value2"/>
- B. <myElement myAttribute="value1" myAttribute="value2"/>
- C. <myElement myAttribute="value1, value2"/>
- D. attributes cannot have multiple values

Answer: D

16. Which of the following XML fragments are well-formed?

- A. <myElement myAttribute="value1 <= value2"/>
- B. <myElement myAttribute="value1 & value2"/>
- C. <myElement myAttribute="value1 > value2"/>
- D. None of the above

Answer: C

17. The use of a DTD in XML development is:

- A. required when validating XML documents
- B. no longer necessary after the XML editor has been customized
- C. used to direct conversion using an XSLT processor
- D. a good guide to populating a templates to be filled in when generating an XML document automatically

Answer: A

18. Parameter entities can appear in

- A. xml file
- B. dtd file
- C. xsl file
- D. Both 1 and 2

Answer: B

19. Attribute standalone="no" should be included in XML declaration if a document

- A. is linked to an external XSL stylesheet
- B. has external general references
- C. has processing instructions
- D. has an external DTD

Answer: B

UNIT-III

1. The method getWriter returns an object of type PrintWriter. This class has println methods to generate output. Which of these classes define the getWriter method?

Select the one correct answer.

- A. HttpServletRequest
- B. HttpServletResponse
- C. ServletConfig
- D. ServletContext

Answer: B

2. Name the method defined in the HttpServletResponse class that may be used to set the content type. Select the one correct answer.

- A. setType
- B. setContent
- C. setContentType
- D. setResponseContentType

Answer: B

3. Which of the following statement is correct? Select the one correct answer.

- A. The response from the dedicated server to a HEAD request consists of status line,

content type and the document.

B. The response from the server to a GET request does not contain a document.

C. The setStatus method defined in the HttpServletRequest class takes an int as an argument and sets the status of Http response

D. The HttpServletResponse defines constants like SC_NOT_FOUND that may be used as a parameter to setStatus method.

Answer: D.

4. The send Error method defined in the HttpServlet class is equivalent to invoking the set Status method with the following parameter. Select the one correct answer.

A. SC_MOVED_TEMPORARILY

B. SC_NOT_FOUND

C. SC_INTERNAL_SERVER_ERROR

D. ESC_BAD_REQUEST

Answer: C.

5. The send Redirect method defined in the HttpServlet class is equivalent to invoking the set Status method with the following parameter and a Location header in the URL. Select the one correct answer.

A. SC_MOVED_TEMPORARILY

B. SC_NOT_FOUND

C. SC_INTERNAL_SERVER_ERROR

D. ESC_BAD_REQUEST

Answer: B

6. Which of the following statements are correct about the status of the Http response. Select the one correct answer.

A. A status of 200 to 299 signifies that the request was successful.

B. A status of 300 to 399 are informational messages.

C. A status of 400 to 499 indicates an error in the server.

D. A status of 500 to 599 indicates an error in the client.

Answer: A.

7. To send binary output in a response, the following method of HttpServletResponse may be used to get the appropriate Writer/Stream object. Select the one correct answer.

A. getStream

B. getOutputStream

C. getBinaryStream

D. getWriter

Answer: B.

8. To send text output in a response, the following method of HttpServletResponse may be used to get the appropriate Writer/Stream object. Select the one correct answer.

A. getStream

B. getOutputStream

C. getBinaryStream

D. getWriter

Answer: D

9. Is the following statement true or false. URL rewriting may be used when a

browser is disabled. In URL encoding the session id is included as part of the URL.

Answer: true.

10. Name the class that includes the get Session method that is used to get the Http Session object.

- A. HttpServletRequest
- B. HttpServletResponse
- C. SessionContext
- D. SessionConfig

Answer: A

11. Which of the following are correct statements? Select the two correct answers.

- A. The get Request Dispatcher method of Servlet Context class takes the full path of the servlet, whereas the get Request Dispatcher method of Http Servlet Request class takes the path of the servlet relative to the Servlet Context.
- B. They include method defined in the Request Dispatcher class can be used to access one servlet from another. But it can be invoked only if no output has been sent to the server.
- C. The get Request Dispatcher(String URL) is defined in both Servlet Context and Http Servlet Request method
- D. The get Named Dispatcher (String) defined in Http Servlet Request class takes the name of the servlet and returns an object of Request Dispatcher class.

Answer: A, C.

12. A user types the URL <http://www.javaprep.com/scwd/index.html> . Which HTTP request gets generated? Select the one correct answer.

- A. GET method
- B. POST method
- C. HEAD method
- D. PUT method

Answer: A.

13. Which HTTP method gets invoked when a user clicks on a link? Select the one correct answer.

- A. GET method
- B. POST method
- C. HEAD method
- D. PUT method

Answer: A.

13. When using HTML forms which of the following is true for POST method? Select the one correct answer.

- A. POST allows users to bookmark URLs with parameters.
- B. The POST method should not be used when large amount of data needs to be transferred.
- C. POST allows secure data transmission over the http method.
- D. POST method sends data in the body of the request.

Answer: D.

14. Which of the following is not a valid HTTP/1.1 method? Select the one correct answer.

- A. CONNECT method

- B. COMPARE method
- C. OPTIONS method
- D. TRACE method

Answer: B.

15. Name the http method used to send resources to the server. Select the one correct answer.

- A. FTP method
- B. PUT method
- C. WRITE method
- D. COPY method

Answer: B.

16. Name the http method that sends the same response as the request. Select the one correct answer.

- A. DEBUG method
- B. TRACE method
- C. OPTIONS method
- D. HEAD method

Answer: B.

17. Which three digit error codes represent an error in request from client? Select the one correct answer.

- A. Codes starting from 200
- B. Codes starting from 300
- C. Codes starting from 400
- D. Codes starting from 500

Answer: C.

18. Name the location of compiled class files within a war file? Select the one correct answer.

- A. /META-INF/classes
- B. /classes
- C. /WEB-INF/classes
- D. /root/classes

Answer: C.

19. To send binary output in a response, the following method of HttpServletResponse may be used to get the appropriate Writer/Stream object. Select the one correct answer.

- A. getStream
- B. getOutputStream
- C. getBinaryStream
- D. getWriter

Answer: B.

UNIT-IV

1. When a JSP page is compiled, what is it turned into?

- A. Applet
- B. Servlet

- C. Application
- D. Maillet

Answer: B

2. Which of the following is not a standard method called as part of the JSP life cycle?

- A. jspInit()
- B. jspService()
- C. jspService()
- D. jspDestroy()

Answer: A

3. If you want to override a JSP file's initialization method, within what type of tags must you declare the method?

- A. <@ @>
- B. <%@ %>
- C. <% %>
- D. <%! %>

Answer: B

4. What is the default location for applications served by tomcat?

- A. /conf
- B. /webapps
- C. /bin
- D. /include

Answer: B

5. _____ keep the information on the server and pass only an identifier between a browsers and servers.

- A. URL Rewriting
- B. Session tracking
- C. Session ID
- D. Session Control

Answer: B

6. The doGet() method in the example extracts values of the parameter's type and number by using _____

- A. request.getParameter()
- B. request.setParameter()
- C. response.getParameter()
- D. response.getAttribute()

Answer: A

7. A JSP is transformed into a(n):

- A. Java applet
- B. Java servlet
- C. Either 1 or 2 above
- D. Neither 1 nor 2 above

Answer: B

8. What programming language(s) or scripting language(s) does Java Server Pages (JSP) support?

- A. VBScript only

- B. Jscript only
- C. Java only
- D. All of the above are supported

Answer: C

9. How many copies of a JSP page can be in memory at a time? [a].

- A. One
- B. Two
- C. Three
- D. Unlimited

Answer: A

10. How does Tomcat execute a JSP?

- A. As CGI Script
- B. As an Independent process
- C. By one of Tomcat Thread
- D. None of the above is correct

Answer: C

11. A bean with a property color is loaded using the following statement `<jsp:usebean id="fruit" class="Fruit"/>` What happens when the following statement is executed. Select the one correct answer. `<jsp:setProperty name="fruit" property="*" />`

- A. This is incorrect syntax of `<jsp:setProperty/>` and will generate a compilation error. Either value or param must be defined.
- B. All the properties of the fruit bean are initialized to a value of null.
- C. All the properties of the fruit bean are assigned the values of input parameters of the JSP page that have the same name.
- D. All the properties of the fruit bean are initialized to a value of *.

Answer: C

12. Is the following statement true or false. If the `isThreadSafe` attribute of the `page` directive is false, then the generated servlet implements the `SingleThreadModel` interface.

Answer: true. (The `page` directive is defined as `<%@page isThreadSafe="false"%>`)

13. Which of the following represents a correct syntax for usebean? Select the two correct answers.

- A. `<jsp:usebean id="fruit" scope ="page"/>`
- B. `<jsp:usebean id="fruit" type ="String"/>`
- C. `<jsp:usebean id="fruit" type ="String" beanName="Fruit"/>`
- D. `<jsp:usebean id="fruit" class="Fruit" beanName="Fruit"/>`

Answer: B,C.

14. Name the default value of the scope attribute of `<jsp:usebean>`.

- A. page
- B. application
- C. session
- D. request

Answer: A

15. Which of the following statements are true for `<jsp:usebean>`. Select the two correct answers.

- A. The id attribute must be defined for <jsp:usebean>.
- B. The scope attribute must be defined for <jsp:usebean>.
- C. The class attribute must be defined for <jsp:usebean>.
- D. The <jsp:usebean> must include either type or class attribute or both.

Answer: A,D.

16. Which of these are legal attributes of page directive. Select the two correct answers.

- A. include
- B. scope
- C. errorPage
- D. session

Answer: C,D.

17. Which of the following represents the XML equivalent of this statement <% @include file="a.jsp"%> . Select the one correct statement

- A. <jsp:include file="a.jsp"/>
- B. <jsp:include page="a.jsp"/>
- C. <jsp:directive.include file="a.jsp"/>
- D. There is no XML equivalent of include directive.

Answer: C

18. Assume that you need to write a JSP page that adds numbers from one to ten, and then print the output. <% int sum = 0; for(j = 0; j < 10; j++) { %> // XXX --- Add j to sum <% } %> // YYY --- Display this sum Which statement when placed at the location XXX can be used to compute the sum. Select the one correct statement

- A. <% sum = sum + j %>
- B. <% sum = sum + j; %>
- C. <%= sum = sum + j %>
- D. <%= sum = sum + j; %>

Answer: B

19. Now consider the same JSP example as last question. What must be added at the location YYY to print the sum of ten numbers. Select the one correct statement

- A. <% sum %>
- B. <% sum; %>
- C. <%= sum %>
- D. <%= sum; %>

Answer: C

20. JSP pages have access to implicit objects that are exposed automatically. One such object that is available is request. The request object is an instance of which class?

- A. HttpRequest
- B. ServletRequest
- C. Request
- D. HttpServletRequest

Answer: D

UNIT-V

1. Why so JavaScript and Java have similar name?

- A. JavaScript is a stripped-down version of Java
- B. JavaScript's syntax is loosely based on Java's
- C. They both originated on the island of Java
- D. None of the above

Answer: B

2. When a user views a page containing a JavaScript program, which machine actually executes the script?

- A. The User's machine running a Web browser
- B. The Web server
- C. A central machine deep within Netscape's corporate offices
- D. None of the above

Answer: A

3. _____ JavaScript is also called client-side JavaScript.

- A. Microsoft
- B. Navigator
- C. LiveWire
- D. Native

Answer: B

4. _____ JavaScript is also called server-side JavaScript.

- A. Microsoft
- B. Navigator
- C. LiveWire
- D. Native

Answer: C

5. What are variables used for in JavaScript Programs?

- A. Storing numbers, dates, or other values
- B. Varying randomly
- C. Causing high-school algebra flashbacks
- D. None of the above

Answer: A

6. _____ JavaScript statements embedded in an HTML page can respond to user events such as mouse-clicks, form input, and page navigation.

- A. Client-side
- B. Server-side
- C. Local
- D. Native

Answer: A

7. What should appear at the very end of your JavaScript?

The <script LANGUAGE="JavaScript">tag

- A. The </script>
- B. The <script>
- C. The END statement
- D. None of the above

Answer: A

8. Which of the following can't be done with client-side JavaScript?

- A. Validating a form
- B. Sending a form's contents by email
- C. Storing the form's contents to a database file on the server
- D. None of the above

Answer: C

9. Which of the following are capabilities of functions in JavaScript?

- A. Return a value
- B. Accept parameters and Return a value
- C. Accept parameters
- D. None of the above

Answer: C

10. Which of the following is not a valid JavaScript variable name?

- A. 2names
- B. _first_and_last_names
- C. FirstAndLast
- D. None of the above

Answer: A

11. _____ tag is an extension to HTML that can enclose any number of JavaScript statements.

- A. <SCRIPT>
- B. <BODY>
- C. <HEAD>
- D. <TITLE>

Answer: A

12. How does JavaScript store dates in a date object?

- A. The number of milliseconds since January 1st, 1970
- B. The number of days since January 1st, 1900
- C. The number of seconds since Netscape's public stock offering.
- D. None of the above

Answer: A

13. Which of the following attribute can hold the JavaScript version?

- A. LANGUAGE
- B. SCRIPT
- C. VERSION
- D. None of the above

Answer: A

14. What is the correct JavaScript syntax to write "Hello World"?

- A. System.out.println("Hello World")
- B. println ("Hello World")
- C. document.write("Hello World")
- D. response.write("Hello World")

Answer: C

15. Which of the following way can be used to indicate the LANGUAGE attribute?

- A. <LANGUAGE="JavaScriptVersion">

- B. <SCRIPT LANGUAGE="JavaScriptVersion">
- C. <SCRIPT LANGUAGE="JavaScriptVersion"> JavaScript statements...</SCRIPT>
- D. <SCRIPT LANGUAGE="JavaScriptVersion"!> JavaScript statements...</SCRIPT>

Answer: C

16. Inside which HTML element do we put the JavaScript?

- A. <js>
- B. <scripting>
- C. <script>
- D. <javascript>

Answer: C

17. What is the correct syntax for referring to an external script called " abc.js"?

- A. <script href=" abc.js">
- B. <script name=" abc.js">
- C. <script src=" abc.js">
- D. None of the above

Answer: C

18. Which types of image maps can be used with JavaScript?

- A. Server-side image maps
- B. Client-side image maps
- C. Server-side image maps and Client-side image maps
- D. None of the above

Answer: B

19. Which of the following navigator object properties is the same in both Netscape and IE?

- A. navigator.appCodeName
- B. navigator.appName
- C. navigator.appVersion
- D. None of the above

Answer: A

20. Which is the correct way to write a JavaScript array?

- A. var txt = new Array(1:"tim",2:"kim",3:"jim")
- B. var txt = new Array:1=("tim")2=("kim")3=("jim")
- C. var txt = new Array("tim","kim","jim")
- D. var txt = new Array="tim","kim","jim"

Answer: C

ASSIGNMENT QUESTION BANK

PART – A (SHORT ANSWER QUESTIONS)

UNIT-I

- 1 What is PHP?
- 2 What is Data Type?
- 3 Explain different types of operators?
- 4 Explain String Functions?
- 5 What is a Function?
- 6 Define Multi-dimensional Array?
- 7 Explain Scope, Local, Global, Static?
- 8 What is the output of the following:

```
<?php
$actor[0]='laxmi';
Echo "\$actor[0]";
?>
```
- 9 Discuss associative arrays with example What is the output of following:
10.

```
<?php
$x=10;
function f()
{
echo GLOBALS['$x'];
}
f();
echo $x;
?>
```

UNIT-II

- 1 Define an xml scheme show how an XML Scheme can be created
- 2 How do you define the elements of an XML document in an XML Schema?
- 3 Explain is XML? What are the advantages of xml?
- 4 Explain are the different revolution in which XML is playing a major role?
- 5 Explain and show how XML is useful in defining data for web applications.
- 6 Explain the various terms related to Document Type Definition.
- 7 Design an XML schema for hospital information management. Include every feature available with schema.
- 8 Explain how styling XML with cascading style sheets is done for the library information domain.
- 9 Discuss the important features of XML which make it more suitable than HTML for creating web related services Creating A
- 10 What is DOM and SAX?

UNIT-III

- 1 List out various phases of Servlet life cycle?
- 2 Write a Servlet program to illustrate parameter reading and parameter initializing. ?
- 3 When init() method of servlet gets called?
- 4 When service() method of servlet gets called?
- 5 When doGet() method of servlet to be called?
- 6 What are Servlets?
- 7 What are the major tasks of servlets?
- 8 For what purpose doGet() method of a servlet is used?
- 9 How to read form data in servlet?
- 10 How to read name of all parameters in servlet?

UNIT-IV

- 1 What is JSP?
- 2 What are advantages of using JSP?
- 3 What are the advantages of JSP over Active Server Pages (ASP)?
- 4 What are the advantages of JSP over Pure Servlets?
- 5 What are the advantages of JSP over JavaScript?
- 6 Explain lifecycle of a JSP.
- 7 What are JSP declarations?
- 8 What is a scriptlet in JSP and what is its syntax?
- 9 What are JSP expressions?
- 10 What are JSP comments?

UNIT-V

- 1 What is JavaScript?
- 2 Name some of the JavaScript features.
- 3 What are the advantages of using JavaScript?
- 4 What are disadvantages of using JavaScript?
- 5 Is JavaScript a case-sensitive language?
- 6 How can you create an Object in JavaScript?
- 7 How can you read properties of an Object in JavaScript?
- 8 How can you create an Array in JavaScript?
- 9 How to read elements of an array in JavaScript?
- 10 What is arguments object in JavaScript?

PART – B (LONG ANSWER QUESTIONS)

UNIT-I

- 1 What is string Interpolation? Explain briefly about PHP's Internal Data Types
- 2 Explain the conversion between different data types in PHP with an Example program.

- 3 How text manipulation is done in PHP. Write a program to compare two strings and print respectively.
- 4 Write a program in PHP to create a registration page using FORMS.
- 5 Write a program in PHP for passing arguments using Functions.
- 6 Write a PHP program to print addition of two matrices.
- 7 Write a PHP program to count number of words in the given string.
- 8 Write a PHP program to check whether given number is Armstrong or not.
- 9 Write a PHP program to create user validation Form.
- 10 Explain about for each loop.

UNIT-II

- 1 Explain and show how XML is useful in defining data for web applications.
- 2 Explain the various terms related to Document Type Definition.
- 3 Design an XML schema for hospital information management. Include every feature available with schema.
- 4 Explain how styling XML with cascading style sheets is done for the library information domain.
- 5 Discuss the important features of XML which make it more Suitable than HTML for creating web related services.
- 6 Define an xml scheme show how an XML Scheme can be created
- 7 Write a JavaScript program to validate XML document against a schema?
- 8 When an element is called simple? How does it differ from a complex element?
- 9 How do you define the elements of an XML document in an XML Schema?
- 10 How do you set default and fixed values for simple Elements?

UNIT-III

- 1 Write a Servlet that generates HTML page and explain the process of generation of HTML page
- 2 List and explain the classes and interfaces of javax. Servlet .http package.
- 3 Develop a Servlet that handles HTTP get Request
- 4 Describe about session tracking with relevant code snippet.
- 5 Servlet offer several advantages over CGI".
- 6 Write about Security Issues in Servlet
- 7 Write about Servlet? Explain lifecycle of a Servlet. Illustrate with an example program.
- 8 Write a Servlet program to illustrate parameter reading and parameter initializing.
- 9 Explain Cookies session tracking with relevant code snippet.
- 10 **List** the methods defined in Http Servlet Request.

UNIT-IV

- 1 Write about the JSP processing.
- 2 Explain the mechanism to include resources dynamically and to forward request to other JSPs?
- 3 Explain about JSP Elements?
- 4 List the different Action Tags used in JSP with their functionality

- 5 Explain the types of Scripting tags and Directive tags in JSP.
- 6 Write a Short note on JSP Implicit Objects.
- 7 Explain different JSP Directive Elements? Explain each one of them in detail?
- 8 Explain JSP application design with suitable example?
- 9 Write a JSP with a Bean in the session scope.
- 10 Describe the MVC architecture and write a JSP program which prints the current date?

UNIT-V

- 1 Write a JavaScript that displays the as per the following: Understand (calculates the squares and cubes of the numbers from 0 to 210)
- 2 Write a JavaScript to analyze a subject code for subject in a semester. The subject code may be visualized like 12 CS 43 engineering discipline as Computer Science and Engineering and „4“ gives the semester details as 4 them, and „3“ gives the subject informatin.
- 4 Show how JavaScript can handle the events? “JavaScript is event driven”. What is meant by an event
- 5 Define an Object. Explain the various objects that are used in theJavaScript
- 6 Write a java script which accepts text in lower case and displays text in uppercase.
- 7 Write a java script to validate a form consisting of user name .Also navigate to another web pages after navigation
- 8 Write a java script that read four integers and display the largest and displays the largest and smallest integers from the given integers.
- 9 Write a java Script program to determine whether a given number is an Armstrong number or not
- 10 Write a JavaScript that reads list of ten numbers and displays the count of negative numbers, the count of positive numbers and the count of zeros from the list.

PART – C (PROBLEM SOLVING AND CRITICAL THINKING QUESTIONS)

UNIT –I

- 1 Write a php program that replaces some characters with some other characters in a string.
- 2 Write a php program uses a constant inside a function, even if it is defined outside the function.
- 3 Write a php program that uses array operators

UNIT –II

- 1 Try an XML program to print the food menu.
- 2 Implement xml in doing CD catalog.

UNIT –III

- 1 Write a servlet program to print hello world.
- 2 Write a sample program which will pass two values to Hello Form program using GET method.
- 3 Write a servlet program using get Header Names () method of Http Servlet Request to read the HTTP header information.

UNIT –IV

- 1 Write login HTML page, we will put it in the welcome files list in the web.xml so that when we launch the application it will open the login page. If the login will be successful, the user will be presented with new JSP page with login successful message
- 2 Write a jsp program to print the current date.
- 3 Use **setInt Header()** method to set **Refresh** header to simulate a digital clock.

UNIT –V

- 1 Write a java script program to print fibnocci series.
- 2 Write a JavaScript form validation program with example
- 3 Write a JavaScript palindrome program with example.

TUTORIALS QUESTION BANK

PART – A (SHORT ANSWER QUESTIONS)

UNIT-I

1. What is PHP?
2. What is the use of "echo" in php?
3. What's the difference between include and require?
4. require_once(), require(), include(). What is difference between them?
5. Differences between GET and POST methods ?
6. How to declare an array in php?
7. What is the use of 'print' in php? .
8. What is use of in_array() function in php ? .
9. What is use of count() function in php ? .
10. what is sql injection ? .

UNIT-II

1. What is XML? .
2. What is the difference between XML and HTML? .
3. What is a well-formed XML document? .
4. What is a valid XML document? .
5. What is a Processing Instruction in XML? .
6. How does the XML structure is defined?
7. What is DTD? .
8. What is a Complex Element? .
9. What is a Simple Element? .
10. What are namespaces? Why are they important? .

UNIT-III

1. What are the advantages of servlets over CGI? .
2. How to redirect a request from a servlet to another servlet? .
3. How sendRedirect method works? .
4. What are servlets filters? .
5. For what purpose destroy() method of a filter is used? .
6. How to create a cookie using servlet? .

7. How to read a cookie using servlet? .
8. How to delete a cookie using servlet? .
9. How to create a session in servlet? .
10. How to update an attribute in session in servlet? .

UNIT-IV

1. What is a page directive? .
2. What is a buffer attribute? .
3. What is contentType attribute? .
4. What is session attribute? .
5. What is a taglib directive? .
6. What is a include directive? .
7. What is <jsp:setProperty> action? .
8. What is <jsp:getProperty> action? .
9. What are JSP implicit objects? .
10. What is a response object? .

UNIT-V

1. Which built-in method sorts the elements of an array? .
2. Which built-in method returns the calling string value converted to lower case? .
3. How typeof operator works? .
4. What typeof returns for a null value? .
5. How to read a Cookie using JavaScript? .
6. How to delete a Cookie using JavaScript? .
7. How to handle exceptions in JavaScript? .
8. Which built-in method reverses the order of the elements of an array? .
9. Which built-in method combines the text of two strings and returns a new string? .
10. What is the purpose of 'this' operator in JavaScript? .

PART – B (LONG ANSWER QUESTIONS)

UNIT-I

1. How to set a page as a home page in a php based site? How to find the length of a string?
2. What is mean by an associative array? Write a program to print the name and the age of a person's using associative arrays.
3. Extend the importance of "method" attribute in a html form? Explain it with the help of an example.
4. What is the importance of "action" attribute in a html form? .
5. How to create an array of a group of items inside an HTML form? .

UNIT-II

1. How XSL-FO Works (or) How would you produce PDF output using XSL's?
2. Explain XSLT? Understanding e
3. What is a CDATA section in XML? .
4. What is XML parser? .
5. What are the interfaces of SAX? .

UNIT-III

1. What are common tasks performed by Servlet Container?
2. What is Servlet Config object? What is Servlet Context object? .
3. What is difference between Servlet Config and Servlet Context?
4. What is Request Dispatcher? What is difference between Print Writer and Servlet Output Stream? .
5. What is Single Thread Model interface? .

UNIT-IV

1. Explain JSP directives.
2. Show attributes of page directives.
3. What are standard actions in JSP? .
4. Explain the jsp:set Property action.
5. Explain client and server side validation. .

UNIT-V

1. What is called Variable typing in Java script? How can you convert the string of any base to integer in JavaScript? .
2. What is an undefined value in JavaScript? What are all the types of Popup boxes available in JavaScript?
3. What is the difference between an alert box and a confirmation box? What are escape characters? .
4. Explain what is pop() method in JavaScript? Mention what is the disadvantage of using inner HTML in JavaScript? .

PART – C (PROBLEM SOLVING AND CRITICAL THINKING QUESTIONS)

UNIT-I

1. How to set a page as a home page in a php based site? How to find the length of a string?
2. What is mean by an associative array? Write a program to print the name and the age of a person's using associative arrays.
3. Extend the importance of "method" attribute in a html form? Explain it with the help of an example.
4. What is the importance of "action" attribute in a html form? .
5. How to create an array of a group of items inside an HTML form? .

UNIT-II

1. Explain XSLT? Understanding e
2. What is a CDATA section in XML? .
3. What is XML parser? .
4. What are the interfaces of SAX? .

UNIT-III

5. What is a web application and what is it's directory structure? .
1. What are common tasks performed by Servlet Container? .

2. What is Servlet Config object? What is Servlet Context object? .
3. What is difference between Servlet Config and Servlet Context? .
4. What is Request Dispatcher? What is difference between Print Writer and Servlet Output Stream? .
5. What is Single Thread Model interface? .

UNIT-IV

1. Explain JSP directives. Understand a
2. Show attributes of page directives. Apply a
3. What are standard actions in JSP? .
4. Explain the jsp:set Property action. Understand i
- 5 Explain client and server side validation. .

UNIT-V

1. What is an undefined value in JavaScript? What are all the types of Popup boxes available in JavaScript?
2. What is the difference between an alert box and a confirmation box? What are escape characters? .
3. Explain what is pop() method in JavaScript? Mention what is the disadvantage of using inner HTML in JavaScript? .

UNIT-1

Topics:

Introduction to PHP: Declaring variables, data types, arrays, strings, operators, expressions, control structures, functions, Reading data from web form controls like text boxes, radio buttons, lists etc., Handling File Uploads. Connecting to database (MySQL as reference), executing simple queries, handling results, Handling sessions and cookies

File Handling in PHP: File operations like opening, closing, reading, writing, appending, deleting etc. on text and binary files, listing directories.

Introduction

PHP is a server-side scripting language. PHP processor is open source and can be downloaded without any cost. PHP was developed by Rasmus Lerdorf, a member of the Apache Group, in 1994. Most of the binaries were written in C. Lerdorf called the initial PHP as Personal Home Page. But later it was changed to Hypertext PreProcessor.

What is PHP? What are the common uses of PHP? (2 marks May 2017)

PHP stands for "PHP: Hypertext Preprocessor". Earlier it was called Personal Home Page. PHP is a server side scripting language that is embedded in HTML. The default file extension for PHP files is ".php". PHP supports all the databases that are present in the market. PHP applications are platform independent. PHP application developed in one OS can be easily executed in other OS also.

Common uses of PHP

- PHP performs system functions, i.e. from files on a system it can create, open, read, write, and close them.
- PHP can handle forms, i.e. gather data from files, save data to a file, thru email we can send data, return data to the user.
- We add, delete, modify elements within web database using PHP.
- Access cookies variables and set cookies.
- Using PHP, we can restrict users to access some pages of web website.
- It can encrypt data.

Creating a first PHP Page

PHP scripts can be embedded in a (X)HTML document or written separately without any (X)HTML markup. In both cases, the file must be saved with the extension .php.

Creating a First PHP Page:

Any PHP script (code) must be enclosed within the PHP tags which are represented using <?php (opening tag) and ?> (closing tag). Let's consider a simple PHP script which prints "Hello World" on the web document:

<?php

```
echo "Hello World";  
?>
```

Running Wer first PHP Page:

To run a PHP script, a web server must be installed and started. Well known web servers are Apache Http Server and Microsoft's Internet Information Service (IIS).

After a web server is installed, place the PHP file hello.php in the web server's root directory and start the web server. Now, open a web browser like chrome, firefox or internet explorer and type the following URL in the address bar:

http://localhost/hello.php or

http://localhost:80/hello.php

80 is the port at which the web server listens for incoming HTTP requests. The output of the PHP script is:

Mixing HTML and PHP

```
<html>                                // Page starts with standard <html> <head> <title> section.
```

```
<head>
```

```
<title>
```

Using PHP and HTML together

```
</title>
```

```
</head>
```

```
<body>                                // <body> section
```

```
<h1> Using PHP and HTML together</h1> //Contain <h1> header and some text
```

```
Here is PHP info: </br></br>
```

```
<?php                                // <?php starts a PHP section
```

```
phpinfo();                            //PHP function phpinfo(); displays information about the PHP  
                                       //installation, Every PHP statement ends with semicolon
```

```
?>
```

```
</body>
```

```
</html>
```

When this page is run by PHP engine on the server HTML will be passed through browser and PHP part will be executed

Printing Some Text

Output Functions in PHP

In PHP there are two basic ways to get output: echo and print.

The PHP echo Statement

The echo() function outputs one or more strings.

The echo statement can be used with or without parentheses: echo or echo().

Syntax: echo(strings)

Example

```
<?php
```

```
echo "Hello world!";
```

```
echo ("Hello world!");
```

?>

PHP print() Function :

Syntax: print(strings)

Example:

```
<?php
print "Hiee!";
?>
```

echo and print are used to output data to the screen. **The differences are small:** echo has no return value while print has a return value of 1 so it can be used in expressions. echo can take multiple parameters (although such usage is rare) while print can take one argument. echo is marginally faster than print.

PHP printf() Function

The printf() function outputs a formatted string.

Syntax : printf(format,arg1,arg2,arg++)

Possible format values: %d - Signed decimal number, %f - Floating-point number, %s - String

Example:

Using the format value %f:

```
<?php
$number = 123;
printf("%f",$number);
?>
```

Example 2

```
<?php
$number = 9; $str = "Beijing";
printf("There are %u million bicycles in %s.", $number, $str);
?>
```

Adding comments to PHP code

// This is single-line comment

This is also a single-line comment

/*

This is a multiple-lines comment block
that spans over multiple
lines

*/

Working with Variables

Variables :

A variable is a named location in memory to store data temporarily. PHP is dynamically typed language. So, there is no need for mentioning the data type of a variable. The type will be detected automatically based on the value assigned to the variable. A variable can be created as shown below:

\$var1 = 10;

A variable which is not assigned a value contains NULL. In expressions containing numbers, NULL will be coerced to 0 and in case of strings, NULL will be coerced to an empty string. A variable can be printed as shown below:

```
print("Value of var1 is: $var1");
```

A variable can be checked if it contains a value other than NULL by using the function IsSet. This function returns TRUE if the variable contains a non-NULL value and FALSE if it contains a NULL value.

Storing Data in Variables

PHP variables can hold numbers or strings of characters. We can store information in variables with an assignment operator/ single equal sign (=).

```
<?php
$name="ACE";
$age = 21;
$price = 2.55;
$number = -2;
?>
```

After the execution of the statements above, the variable \$name will hold the value ACE, the variable \$age will hold the value 21, the variable \$price will hold the value 2.55, and the variable \$number will hold the value -2.

Creating Variables

Variables are "containers" that are used for storing information.

In PHP, a variable starts with the \$ sign, followed by the name of the variable.

What are the rules for naming a variable in PHP?

Every variable starts with the \$ sign, followed by the name of the variable.

A variable name must start with a letter or the underscore character(can't start with a number)

A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)

Variable names are case-sensitive (\$age and \$AGE are two different variables)

1) \$name 2) \$_name 3) \$1name 4) \$_1name

Creating constants

Constants are identifiers which are used to store values that cannot be changed once initialized.

A constant doesn't begin with a dollar (\$) sign. The convention for constants is, the name of a constant should always be written in uppercase. We use the function define to create constants.

Below example demonstrates creating and using constants in PHP:

```
<?php
define("PI", 3.142); print(PI);
$area = PI*10*10; //Let radius be 10 print("<br />Area of circle is: $area");
?>
```

Operators and Flow Control

Data Types

PHP provides four scalar types namely Boolean, integer, double and string and two compound types namely array and object and two special types namely resource and NULL.

PHP has a single integer type, named integer. This type is same as long type in C. The size of an integer type is generally the size of word in the machine. In most of the machines that size will be 32 bits.

PHP's double type corresponds to the double type in C and its successors. Double literals can contain a decimal point, an exponent or both. An exponent is represented using E or e followed by a signed integer literal. Digits before and after the decimal point are optional. So, both .12 and 12. are valid double literals.

String is a collection of characters. There is no special type for characters in PHP. A character is considered as a string with length 1. String literals are represented with single quotes or double quotes. In a string literal enclosed in single quotes, escape sequences and variables are not recognized and no substitutions occurs. Such substitution is known as interpolation. In string literals enclosed in double quotes, escape sequence and variables are recognized and corresponding action is taken.

The only two possible values for a **Boolean type** are TRUE and FALSE both of which are case-insensitive. Integer value 0 is equal to Boolean FALSE and anything other than 0 is equal to TRUE. An empty string and string "0" are equal to Boolean FALSE and remaining other strings are equal to TRUE. Only double value equal to Boolean FALSE is 0.0.

PHP Maths Operators

| Operator | Name | Example | Description |
|----------|----------------|------------|--------------------------------------|
| + | Addition | \$x + \$y | Sum of x and y |
| - | Subtraction | \$x - \$y | Difference of x and y |
| * | Multiplication | \$x * \$y | Product of x and y |
| / | Division | \$x / \$y | Quotient of x divided by y |
| % | Modulus | \$x % \$y | Remainder of x divided by y |
| ** | Exponentiation | \$x ** \$y | Result of x raised to the power of y |

Working with the Assignment Operators

PHP assignment operators are used in assignment expressions to store the value of expression in to a variable. Below is a list of assignment operators:

Assignment Description

| | |
|--------|--|
| x = y | Assigning value of y to x |
| x += y | Adding x and y and store the result in x |
| x -= y | Subtracting y from x and store the result in x |
| x *= y | Multiplying x and y and store the result in x |
| x /= y | Dividing x by y and store the quotient in x |
| x %= y | Dividing x by y and store the remainder in x |

Increment and Decrement Operator

The increment/decrement operators are used to increment the value of variable by 1 or decrement the value of variable by 1. The increment operator is ++ and decrement operator is --.

The PHP String Operators

PHP provides two operators which are used with strings only. They are listed below:

| Operator | Name | Example | Description |
|-----------------|---------------|-----------------|--------------------------------|
| . | Concatenation | \$str1.\$str2 | str1 and str2 are concatenated |
| .= | Concatenation | \$str1.= \$str2 | str2 is appended to str1 |

The Bitwise Operators

PHP logical operators are used to find the Boolean value of multiple conditional expressions. Below is a list of logical operators:

| Operator | Name | Example | Description |
|-----------------|-------------|----------------|--|
| and | And | \$x and \$y | Returns true when both x and y are true |
| or | Or | \$x or \$y | Returns true when either x or y or both of them are true |
| xor | Xor | \$x xor \$y | Returns true when either x or y is true |
| && | And | \$x && \$y | Returns true when both x and y are true |
| | Or | \$x \$y | Returns true when either x or y or both of them are true |
| ! | Not | !\$x | Returns true when x is false and vice versa |

Using the if Statement

if statement is a conditional statement that allows us to make decisions on what code to execute.

The structure looks like:

if (expression)
statement

Here, expression evaluates to a TRUE or FALSE.

If expression is TRUE, The statement that follows is executed; if it is FALSE, statement is not executed

We use conditional and Logical Operators to create expression of if statement.

Example: We want to display some text if the outside temperature is above 28 degree.

```
<?php
$temperature=35;
if($temperature>28)
{
    echo "Its hotter outside";
}
?>
```

we put the statements in curly braces.

Example: To check how many minutes someone has been in the pool- if its more than 30minute, it's time to get out.

```
<html>
<head>
<title> Using the if statement</title>
```



```

</head>
<body>
<h1>Using the if Statement</h1>

<?php
$minutes=31;
if($minutes > 30)
{
    echo "Wer time is UP!!<br>";
    echo "Please get out of the pool.";
}
?>
</body>
</html>

```

The else statement

Often we want to execute a statement if a condition is TRUE, and we execute different statement if the condition is FALSE.

```

if (expression) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}

```

Example: We want to print a message if the outside temperature is outside the range 24 degree and 32 degree, and another message if the temperature is inside the range.

```

<html>
<head><title> Using the else statement</title></head>
<body>
<h1>Using the else Statement</h1>
<?php
$temperature=44;
if($temperature<24 || $temperature>32)
{
    echo "Better Stay inside today!!";
}
else
{
    echo "Nice weather outside";
}
?>
</body></html>

```

The elseif statement

elseif, as its name suggests, is a combination of if and else.

If an if statements condition is false, we can make additional tests with elseif.

```
if (condition1) {  
    code to be executed when condition1 is true;  
} elseif (condition2) {  
    code to be executed when condition2 is true;  
} else {  
    code to be executed when conditions are false;  
}
```

```
<html>  
<head>  
<title> Using the elseif statement</title>  
</head>  
<body>  
<h1>Using the elseif Statement</h1>  
<?php
```

```
$a=40;  
$b=30;  
if ($a > $b)  
{  
    echo "$a is bigger than $b";  
} elseif ($a == $b)  
{  
    echo "$a is equal to $b";  
} else {  
    echo "$a is smaller than $b";  
}  
>  
</body>  
</html>
```

If the if statement's conditional expression is false, It will check the first elseif statements expression, if it's true elseif code is executed and if its false it moves to the next elseif statement and so on. At the end else statement is executed if the no other code is executed in the if statement up to this point.

switch statement:

switch statement lets us replace long if-elseif-else ladder of condition checking with switch statement. switch statement compares a value against case statement and execute a code block whose value matches the case value. if no case value matches the value, default statement is executed. break statement ends execution of the switch statement, if we don't write break statement, execution will continue with the code.

```
<?php  
switch(value){
```

```

    case value1:
        // code block 1
        break;
    case value2:
        // code block 2
        break;

    default:
        // default code block
        break;
}

```

Example:

```

<html>
<head>
<title> Using the switch statement</title>
</head>
<body>
<h1>Using the switch Statement</h1>
<?php

```

```

$favcolor = "red";

```

```

switch ($favcolor)
{
    case "red":
        echo "Wer favorite color is red!";
        break;
    case "blue":
        echo "Wer favorite color is blue!";
        break;
    case "green":
        echo "Wer favorite color is green!";
        break;
    default:
        echo "Wer favorite color is neither red, blue, nor green!";
}

```

```

?>
</body>
</html>

```

for loop

for loop is a repetition control structure that allows us to execute a block of code a specified number of times.

When we want to execute same block of code over and over again specified number of times.

Syntax:

**for(expression1 ; expression1 ; expression1)
statement**

First, for loop executes expression1; then it checks the value of expression2 - if true , loop executes statement once.

Then it executes expression3 and after that checks the expression2 again-- if true loop execute statement once again.

Then loop executes expression3 again, and keeps going untill expression2 becomes false.

<html>

<head>

<title>

Using the for loop

</title>

</head>

<body>

<?php

```
for ($x = 0; $x <= 10; $x++) {  
    echo "The number is: $x <br>";
```

```
}
```

```
?>
```

</body>

</html>

for loop can handle multiple loop counters by seperating them with comma operator.

<?php

```
for ($i=2, $k=2; $i<6 && $k<6 ; $i++, $k++) {  
    echo "$i * $k = ", $i * $k, "<br>";
```

```
}
```

```
?>
```

The two variables \$i and \$k are initialized with 2.

At the end of each loop \$i and \$k will be incremented by one.

We can put one loop inside a another loop, call as nested loop.

<?php

```
for ($row = 1; $row <= 5; $row++)
```

```
{
```

```
    for ($col = 1; $col <= 5; $col++)
```

```
    {
```

```
        echo '*';
```

```
    }
```

```
    echo "\n";
```

```
}
```

```
?>
```

Outputs:

While Loop

When we want to execute same block of code over and over again as long as the condition is true, we go for while loop.

Syntax:

```
while (expression)
{
statement
}
```

While expression is true, the loop executes statement. When expression is false, the loop ends.

```
<html>
<head>
<title>
Using the while loop
</title>
</head>
<body>
<h1> Using the while loop </h1>
<?php
$x=1;

while($x<10)
{
    echo "Now $x holds : ",$x,"<br>";
    $x++;
}

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

Using do... while loop

"do while" loop is a slightly modified version of the while loop

In while loop, the condition is checked at the beginning of each iteration, in do-while condition is checked at the end of each iteration.

It means do-while loop always executes its block of code at least once even the condition is evaluated to false.

while vs. do-while

do-while loop body executes at least once even when condition is false whereas while loop body may not execute at all if condition evaluates to false.

The condition in the do-while loop is evaluated at the end whereas the condition in the while loop is evaluated at the beginning of each iteration.

Syntax:

```
do
{
    statement
} while (condition);
```

do... while loop keeps executing statement while condition is true- note expression is tested at the end.

Example:

```
<html>
<head>
<title>
Using the do...while loop
</title>
</head>
<body>
<h1> Using the do...while loop </h1>
<?php
$x=1;

do
{
    echo "Now \$x holds : ",$x,"<br>";
    $x++;
} while($x<10);

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

Using the foreach loop

The foreach loop is especially designed to work with array.

foreach loop will loop over all elements of an array.

for loop and while Loop will continue until some condition fails, the foreach loop will continue until it has gone through every item in the array.

There are two syntax:

- o foreach (array_expression as \$value) statement
- o foreach (array_expression as \$key => \$value) statement

break: terminating loop early

- We can stop a loop or switch statement at any time with the break statement.
- break ends the execution of for, foreach, while, do-while or switch statement.
- If we use break inside inner loop, it breaks the execution of inner loop only.

```
<html>
<head>
<title> Using the break statement</title>
</head>

<body>
<h1> Using the break statement</h1>
<?php
for ( $x=0 ; $x < 100 ; $x++)
{
    echo "I 'm going to do this hundred times!<br>";
    if ($x == 10)
    {
        echo "Alright, i'm quitting,<br>";
    }
}
?>
</body>
</html>
```

continue :skipping iteration

Sometimes, we want to skip an iteration of a loop to avoid some kind of problem like division by zero-- we can do that with continue statement.

continue is used within looping structures to skip the rest of the current loop iteration and continue execution at the condition evaluation and proceed to the beginning of the next iteration. break ends a loop completely, continue just shortcuts the current iteration and moves on to the next iteration.

```
while ($foo) { <-----]
    continue; --- goes back here --]
    break; ---- jumps here ----]
}
|
<-----]
```

break exits the loop in which we are in, continue starts with the next cycle of the loop immediatly.

```
<html>
<head>
<title>Using the continue statement</title>
</head>
```

```

<body>
<?php
$x=1;
for( $x=-2; $x < 3 ; $x++ )
{
    if ($x == 0)
    {
        continue;
    }

    echo "1/$x = ",1/$x,"<br>";
    $x++
}

?>
</body>
</html>

```

Arrays

Array is a collection of heterogeneous elements. There are two types of arrays in PHP. First type of array is a normal one that contains integer keys (indexes) which can be found in any typical programming languages. The second type of arrays is an associative array, where the keys are strings.

Array Creation

Arrays in PHP are dynamic. There is no need to specify the size of an array. A normal array can be created by using the integer index and the assignment operator as shown below:

\$array1[0] = 10;

If no integer index is specified, the index will be set to 1 larger than the previous largest index value used. Consider the following example:

\$array2[3] = 5;

\$array2[] = 90;

In the above example, 90 will be stored at index location 4.

There is another way for creating an array, using the array construct, which is not a function. The data elements are passed to the array construct as shown in the below example:

\$array3 = array(10, 15, 34, 56);

\$array4 = array();

In the above example array4 is an empty array which can be used later to store elements. A traditional array with irregular indexes can be created as shown below:

\$array5 = array(3 => 15, 4 => 37, 5 => 23);

The above code creates an array with indexes 3, 4 and 5.

An associative array which contains named keys (indexes) can be created as shown below:

\$ages = array("Ken" => 29, "John" => 30, "Steve" => 26, "Bob" => 28); An array in PHP can be a mixture of both traditional and associative arrays.

Accessing Array Elements

Array elements can be accessed using the subscript (index or key) value which is enclosed in square brackets.

Consider a traditional array as shown below:

\$array1 = array(10, 20, 30, 40);

Third element (30) in the above array can be accessed by writing \$array1[2]. **The index of any traditional array starts with 0.**

Consider an associative array or hash as shown below:

\$ages = array("Ken" => 29, "John" => 30, "Steve" => 26, "Bob" => 28); We can access the age (30) of John by writing \$ages['John'].

Handling Arrays with Loops:

The for Loop

The for loop is used to iterate over the elements of an array when we know in advance the number of elements of an Array. The count() function is used to return the length (the number of elements) of an array & this number is useful when we want to set the number of times the for loop to repeat.

Syntax

```
for (init counter; test counter; increment/decrement counter) {  
    code to be executed;  
}
```

```
<html>
```

```
<head>
```

```
<title>Using a for loop to loop over an Array</title>
```

```
</head>
```

```
<body>
```

```
<h1>Using a for loop to loop over an Array</h1>
```

```
</br></br><?php
```

```
$cities=array("Hyderabad","Chennai","Delhi","Mumbai","Pune","Kolkata","Lucknow  
","Chandigarh");
```

```
$arrayLength=count($cities);
```

```
for($i=0; $i<$arrayLength;$i++)
```

```
{
```

```
    echo $cities[$i],"<br>";
```

```
}
```

```
?>
```

```
</body>
```

```
</html>
```

The print_r Function

There is a simple way to print the contents of an array using print_r function. For example:

```
<?php
$cities=array("Hyderabad","Chennai","Delhi","Mumbai","Pune","Kolkata","Lucknow",
"Chandigarh");
print_r($cities);
?>
```

The foreach Loop

The for each loop is designed to iterate over an elements of an Array (Associative array)/Object.

We have two syntax to use foreach loop:

foreach (array as \$value) statement

foreach (array as \$key => \$value) statement

The following example, puts the foreach loop to work, looping over the \$cities array like:

```
<html>
<head>
<title>
Using a foreach loop to loop over an Array
</title>
</head>

<body>
<h1>Using a foreach loop to loop over an Array</h1>
</br></br>
<?php

$cities=array("Hyderabad","Chennai","Delhi","Mumbai","Pune","Kolkata","Lucknow",
"Chandigarh");

foreach ($cities as $value)
{
    echo "$value<br>";
}

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

when foreach starts executing, the array pointer is automatically set to the first element of the array. On each iteration, the value of the current element is assigned to \$value and the array pointer is incremented by one.

The second form of foreach loop lets us to work with keys as well as values.

Element is a combination of element key and element value. PHP also support slightly different type of array in which index numbers are replaced with user defined string keys.

This type of array is known as Associative Array. The keys of the array must be unique, each key references a single value. The key value relationship is expressed through => symbol.

For example:

```
<html>
<head>
<title>Using a foreach loop with keys and values in an Array
</title>
</head>
<body>
<h1>Using a foreach loop with keys and values in n Array</h1></br></br>
<?php
$details=array("name"=>"ACE","Type"=>"College","Place"=>"Hyderabad");
foreach ($cities as $key=>$value)
{
    echo "$key : $value<br>";
}
?></body></html>
```

Note: If we don't provide an explicit key to an array element, The new key value depends upon the previous key.

```
<?php
$a=array(10,100=>"ACE",30);           //[0]=>10    [100]=>ACE [101]=30
print_r($a);
?>
```

```
<?php
$a=array(10,20,0=>30);
print_r($a);           //[0]=30    [1]=20
?>
```

The while loop

The meaning of a while statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the while expression evaluates to TRUE.

The value of the expression is checked each time at the beginning of the loop.

if the while expression evaluates to FALSE the execution of nested statement will stop.

We can use while loop to iterate over an array.

To handle multiple-item return value from the each function, we can use list() function .

Example:

```
<?php
$Language=array("PHP","XML","Servlet","JSP","JavaScript");
list($x, $y, $z)=$Language;
```

```
echo "We have covered $x, $y and $z.";
```

```
?>
```

list() function assign the two return value from each separate variable. Like:

```
<html>
```

```
<head>
```

```
<title>Using a while loop with keys and values in an Array</title></head>
```

```
<body>
```

```
<h1>Using a while loop with keys and values in n Array</h1></br></br>
```

```
<?php
```

```
$details=array("name"=>"ACE","Type"=>"College","Place"=>"Hyderabad");
```

```
while (list($key, $value)=each($details))
```

```
{
```

```
    echo "$key : $value<br>;
```

```
}
```

```
?>
```

```
</body>
```

```
</html>
```

PHP Array Function

array() - function creates and returns an array.

array_change_key_case() - function changes the case of all key of an array.

array_chunk() - function splits array into chunks. we can divide array into many parts.

count() - function counts all elements in an array.

sort() - function sorts all the elements in an array.

array_reverse() - function returns an array containing elements in reversed order.

array_search() - function searches the specified value in an array.

array_intersect() - function returns the intersection of two array. It returns the matching elements of two array.

array_push() — function pushes one or more elements onto the end of array.

array_pop() — function pops the element off the end of array.

array_merge() — function merge one or more arrays.

array_sum() — function calculate the sum of values in an array.

Converting Between String and Arrays Using implode and explode

The implode function is used to "join elements of an array with a string".

Implode() accepts two argument, first argument the separator which specifies what character to use between array elements, and second one is array.

```
<?php
```

```
$ice_cream[0]="Chocolate";
```

```
$ice_cream[1]="Mango";
```

```
$ice_cream[2]="Strawberry";
```

```
$text=implode(" , ", $ice_cream); //Outputs: Chocolate, Mango, Strawberry
```

```
?>
```

The explode function breaks a string into an array.

Explode() accepts three arguments, first one is the delimiter, second one is the string that needs splitting, and third one is not mandatory.

```
<?php
$ice_cream="Chocolate, Mango, Orange";
$ice_cream=explode(" ", $text);
print_r($ice_cream); //Outputs: Array ( [0]=>Chocolate [1]=>Mango [2]=>Strawberry)
?>
```

Functions

Now that we should have learned about variables, loops, and conditional statements it is time to learn about functions.

When we have a block of code that needs to be run multiple times in our application, we put that block of code into a function.

A function is a block of code (that performs a task) that can be executed whenever we need it.

Advantages of using functions is, it reduces the repetition of code within a program.

Creating functions in PHP

while creating a function its name starts with keyword 'function', followed by the name of the function we want to create followed by parentheses i.e. ()

We put our function's code inside { and }

Syntax:

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

functionName starts with a letter or underscore, followed by any number of letters, numbers, or underscores.

```
<?php
function display() {
    echo "Hello world!";
}
display(); // call the function
?>
```

Passing Functions Some Data

Data can be passed to functions via the argument list/ Parameter list.

We can pass data to functions so functions can operate on that data.

We can define a function to accept values through parameters.

A parameter appears with the parentheses "()" separated by comma.

Syntax:

```
function functionName([parameter_list...])
{
    [statements];
}
```

Example: If we want to pass the person's name, and our function will concatenate this name onto a greeting string.

```
<html>
<head><title> Passing data to functions</title></head>
<body>
<?php
function display($name) {
    echo "Hello $name !!";
}
display("Aditya"); // call the function
?>
</body>
</html>
```

Passing Arrays to Functions

We can pass arrays to functions is very simple.

Example:

Define a function to find total of student's test scores and displays the total.

```
<html>
<head>
<title> Passing arrays to function </title>
</head>
<body>
<h1> Passing arrays to function </h1>
<?php
$scores=array(90,40,65,98,75);    //First create an array of test score
//Now, Lets define an total function:
function total($array)
{
    $totalmarks=0;
    foreach($array as $value)    //we use foreach statement to loop over the array
    {
        $totalmarks += $value ;
    }
}
echo " The total marks: ",total($scores); //Call the function: total($scores)
?>
</body>
</html>
```

Introducing Variable Scope in PHP

Scope refers to the visibility of a variable in the PHP script. A variable defined inside a function will have local scope i.e., within the function. A variable outside the function can have the same name as a local variable in a function, where the local variable has higher precedence over the outer variable.

There are only two scopes available in PHP namely:

- 1) Local scope -- Local variables, Static variables, Function parameters
- 2) Global scope -- Global variables

Accessing Global Data:

- Variables in global scope can be accessed from anywhere from outside a function independent of its boundary.
- Global variables can be defined by using global keyword.
- If we want to use global variables inside a function, we have to prefix the global keyword with the variable.

Example:

```
<html>
<head>
<title>Handling Local and Global scope in functions</title>
</head>
<body>
<h1>Handling Local and Global scope in functions</h1>
```

```
<?php
```

```
$value = 4;
```

```
function func1()
{
$value = 80000;
echo "Inside function one \ $value : ", $value, "<br";
}
```

```
function func2()
{
global $value;
echo "Inside function two \ $value : ", $value, "<br";
}
```

```
echo "outside function \ $value : ", $value;
func1();
func2();
```

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

Working with Static Variables(local scope)

A static variable is again a variable with local scope.

The issue with local variable inside a function is that, as the function exits- the local variable vanishes. When the function is called again, a new local variable is created.

```
<html>
<head>
<title>Keeping a Count of function call</title>
</head>
<body>
<h1>Keeping a Count of function call</h1>

<?php
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:1
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:1
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:1
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:1

function count_function()
{
$counter = 0;
$counter++;
return $counter;
}
?>
</body>
</html>
```

Above code will print counter value always 1, because \$counter is reset to 0 every time we call the count_function.

Making \$counter as global variable

```
<?php
$counter = 0;
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:1
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:2
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:3
echo "Now the count is: ". count_function(),"<br>";           //Now the count is:4

function count_function()
{
global $counter;

```



```
$counter++;  
return $counter;  
}  
?>
```

Connecting to Database:

PHP have Built-in Database Access.

PHP provides built-in database connectivity for a wide range of databases – MySQL, PostgreSQL, Oracle, Berkeley DB, Informix, mSQL, Lotus Notes, and more.

What is MySQL?

MySQL is a database. The data in MySQL is stored in database objects called tables.

A table is a collection of related data entries and it consists of columns and rows.

Databases are useful when storing information categorically. A company may have a database with the following tables: "Employees", "Products", "Customers" and "Orders".

Database Tables

A database most often contains one or more tables. Each table is identified by a name (e.g. "Customers" or "Orders"). Tables contain records (rows) with data.

Create a Connection to a MySQL Database

Before we can access data in a database, we must create a connection to the database. In PHP, this is done with the `mysql_connect()` function.

Syntax

```
mysql_connect(servername,username,password);
```

Example: In the following example we store the connection in a variable (\$con) for later use in the script. The `die()` function prints a message and exits the current script, and will be executed if the connection fails:

```
<?php  
$con = mysql_connect("localhost","peter","abc123"); ]  
if (!$con)  
{  
die('Could not connect: ' . mysql_error());  
}  
// some code  
?>
```

Closing a Connection

The connection will be closed automatically when the script ends. To close the connection before, use the `mysql_close()` function:

```
<?php  
$con = mysql_connect("localhost","peter","abc123"); if (!$con)  
{  
die('Could not connect: ' . mysql_error());
```

```
}  
// some code mysql_close($con);  
?>
```

Create a Database

The CREATE DATABASE statement is used to create a database in MySQL.

Syntax

CREATE DATABASE database_name

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example creates a database called "my_db":

```
<?php  
$con = mysql_connect("localhost","peter","abc123"); // connect to database if (!$con)  
{  
    die('Could not connect: ' . mysql_error());  
} // exit script  
if (mysql_query("CREATE DATABASE my_db",$con)) // create database my_db  
{  
    echo "Database created";  
}  
else  
{  
    echo "Error creating database: " . mysql_error();  
}  
mysql_close($con); // close db connection  
?>
```

Create a Table

The CREATE TABLE statement is used to create a table in MySQL.

Syntax

**CREATE TABLE table_name (
column_name1 data_type, column_name2 data_type, column_name3 data_type,
....
)**

We must add the CREATE TABLE statement to the mysql_query() function to execute the command.

Example

The following example creates a table named "Persons", with three columns. The column names will be "FirstName", "LastName" and "Age":

```

<?php
$con = mysql_connect("localhost","peter","abc123"); if (!$con)
{
die('Could not connect: ' . mysql_error()); }
// Create database
if (mysql_query("CREATE DATABASE my_db",$con))
{
    echo "Database created"; } else
    {
        echo "Error creating database: " . mysql_error(); }
    // Create table mysql_select_db('my_db', $con);
    $sql = "CREATE TABLE Persons (
    FirstName varchar(15), LastName varchar(15), Age int)";
    // Execute query mysql_query($sql,$con);
    mysql_close($con);
?>

```

Insert Data Into a Database Table

The INSERT INTO statement is used to add new records to a database table.

Syntax

It is possible to write the INSERT INTO statement in two forms.

The first form doesn't specify the column names where the data will be inserted, only their values:

```

INSERT INTO table_name
VALUES (value1, value2, value3,...)

```

The second form specifies both the column names and the values to be inserted:

```

INSERT INTO table_name (column1, column2, column3,...) VALUES (value1, value2,
value3,...)

```

To get PHP to execute the statements above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example:In the previous chapter we created a table named "Persons", with three columns; "Firstname", "Lastname" and "Age". We will use the same table in this example. The following example adds two new records to the "Persons" table:

```

<?php
$con = mysql_connect("localhost","peter","abc123");
if (!$con)
{
die('Could not connect: ' . mysql_error());
}

```

```
mysql_select_db('my_db', $con);
```

```
mysql_query('INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Peter',  
'Griffin',35)');
```

```
mysql_query('INSERT INTO Persons (FirstName, LastName, Age) VALUES ('Glenn',  
'Quagmire',33)');
```

```
mysql_close($con);  
?>
```

Select Data From a Database Table

The SELECT statement is used to select data from a database.

Syntax

SELECT column_name(s) FROM table_name

To get PHP to execute the statement above we must use the mysql_query() function. This function is used to send a query or command to a MySQL connection.

Example

The following example selects all the data stored in the "Persons" table (The * character selects all the data in the table):

```
<?php  
$con = mysql_connect("localhost","peter","abc123");  
if (!$con)  
{  
die('Could not connect: ' . mysql_error());  
}  
mysql_select_db('my_db', $con);  
$result = mysql_query('SELECT * FROM Persons');  
while($row = mysql_fetch_array($result))  
{  
echo $row['FirstName'] . " " . $row['LastName']; echo "<br />";  
}  
mysql_close($con);?>
```

Handling Session and Cookies:

PHP Sessions

A PHP session variable is used to store information about, or change settings for a user session. Session variables hold information about one single user, and are available to all pages in one application.

PHP Session Variables

When we are working with an application, we open it, do some changes and then we close it.

This is much like a Session. The computer knows who we are. It knows when we start the application and when we end. But on the internet there is one problem: the web server does not know who we are and what we do because the HTTP address doesn't maintain state.

A PHP session solves this problem by allowing we to store user information on the server for later use (i.e. username, shopping items, etc). However, session information is temporary and will be deleted after the user has left the website. If we need a permanent storage we may want to store the data in a database.

Sessions work by creating a unique id (UID) for each visitor and store variables based on this UID. The UID is either stored in a cookie or is propagated in the URL.

Starting a PHP Session

Before we can store user information in our PHP session, we must first start up the session.

The session_start() function must appear BEFORE the <html> tag:

The code above will register the user's session with the server, allow we to start saving user information, and assign a UID for that user's session.

Destroying a Session

If we wish to delete some session data, we can use the unset() or the session_destroy() function.

The unset() function is used to free the specified session variable:

```
<?php  
unset($_SESSION['views']);  
?>
```

We can also completely destroy the session by calling the session_destroy() function.

PHP Cookie

- A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer.
- Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, we can both create and retrieve cookie values.

How to Create a Cookie?

The setcookie() function is used to set a cookie.

Note: The setcookie() function must appear BEFORE the <html> tag.

setcookie(name, value, expire, path, domain);

In the example below, we will create a cookie named "user" and assign the value "Alex Porter" to it. We also specify that the cookie should expire after one hour:

```
<?php  
setcookie("user", "Alex Porter", time()+3600);  
?>
```

To Retrieve a Cookie Value?

The PHP \$_COOKIE variable is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "user" and display it on a page:

```
<?php
// Print a cookie
echo $_COOKIE["user"];
// A way to view all cookies print_r($_COOKIE);
?>
```

In the following example we use the isset() function to find out if a cookie has been set:

```
<html>
<body>
<?php
if (isset($_COOKIE["user"]))
echo "Welcome " . $_COOKIE["user"] . "!"<br />; else
echo "Welcome guest!"<br />;
?>
</body>
</html>
```

To Delete a Cookie?

When deleting a cookie we should assure that the expiration date is in the past.

```
<?php
// set the expiration date to one hour ago setcookie("user", "", time()-3600);
?>
```

File Handling in PHP

File operation Opening and Closing:

Files are opened in PHP using the fopen command. The command takes two parameters, the file to be opened, and the mode in which to open the file. The function returns a file pointer if successful, otherwise zero (false). Files are opened with fopen for reading or writing.

```
$fp = fopen("myfile.txt", "r");
```

If fopen is unable to open the file, it returns 0. This can be used to exit the function with an appropriate message.

```
if ( !($fp = fopen("myfile.txt", "r")) )
exit("Unable to open the input file.");
```

File Modes

The following table shows the different modes the file may be opened in.

| <u>Mode</u> | <u>Description</u> |
|--------------------|---|
| r | Read Only mode, with the file pointer at the start of the file. |
| r+ | Read/Write mode, with the file pointer at the start of the file. |
| w | Write Only mode. Truncates the file (effectively overwriting it). If the file doesn't exist, fopen will attempt to create the file. |
| w+ | Read/Write mode. Truncates the file (effectively overwriting it). If the file doesn't exist, fopen will attempt to create the file. |
| a | Append mode, with the file pointer at the end of the file. If the file doesn't exist, fopen will attempt to create the file. |
| a+ | Read/Append, with the file pointer at the end of the file. If the file doesn't exist, fopen will attempt to create the file. |

Note: The mode may also contain the letter 'b'. This is useful only on systems which differentiate between binary and text files (i.e. Windows. It's useless on Unix/Linux). If not needed, it will be ignored.

Closing a File

The fclose function is used to close a file when we are finished with it.

fclose(\$fp);

Reading:

Reading from Files

We can read from files opened in r, r+, w+, and a+ mode. The feof function is used to determine if the end of file is true.

```
if ( feof($fp) )  
echo "End of file<br>"
```

The feof function can be used in a while loop, to read from the file until the end of file is encountered. A line at a time can be read with the fgets function:

```
while( !feof($fp) ) {  
//      Reads one line at a time, up to 254 characters. Each line ends with a newline.  
$myline = fgets($fp, 255);  
echo $myline;  
}
```

We can read in a single character at a time from a file using the fgetc function:

```
while ( !feof($fp) ) {  
//      Reads one character at a time from the file. $ch = fgetc($fp);  
echo $ch;  
}
```

We can also read a single word at a time from a file using the fscanf function. The function takes a variable number of parameters, but the first two parameters are mandatory. The first parameter

is the file handle, the second parameter is a C-style format string. Any parameters passed after this are optional, but if used will contain the values of the format string.

```
$listFile = "list.txt"; // See next page for file contents if (!$fp = fopen($listFile, "r"))
exit("Unable to open $listFile.");
while (!feof($fp)) {
// Assign variables to optional arguments
$buffer = fscanf($fp, "%s %s %d", $name, $title, $age);
if ($buffer == 3) // $buffer contains the number of items it was able to assign print "$name
$title $age<br>\n";
}
```

We can also store the variables in an array by omitting the optional parameters in fscanf:

```
while (!feof($fp)) {
$buffer = fscanf($fp, "%s %s %d"); // Assign variables to an array
// Use the list function to move the variables from the array into variables list($name,
$title, $age) = $buffer;
print "$name $title $age<br>\n";
}
```

We can read in an entire file with the fread function. It reads a number of bytes from a file, up to the end of the file (whichever comes first). The filesize function returns the size of the file in bytes, and can be used with the fread function, as in the following example.

```
$listFile = "myfile.txt";
if (!$fp = fopen($listFile, "r"))
exit("Unable to open the input file, $listFile.");
$buffer = fread($fp, filesize($listFile));
echo "$buffer<br>\n";
fclose($fp);
```

We can also use the file function to read the entire contents of a file into an array instead of opening the file with fopen:

```
$array = file('filename.txt');
```

Each array element contains one line from the file, where each line is terminated by a newline.

Writing to Files

The fwrite function is used to write a string, or part of a string to a file. The function takes three parameters, the file handle, the string to write, and the number of bytes to write. If the number of bytes is omitted, the whole string is written to the file. If we want the lines to appear on separate lines in the file, use the \n character. Note: Windows requires a carriage return character as well as a new line character, so if we're using Windows, terminate the string with \r\n. The following example logs the visitor to a file, then displays all the entries in the file.

```
<?php
$logFile = "stats.txt";
$fp = fopen($logFile, "a+"); // Open the file in append/read mode
```



```
$userDetails = $_SERVER['HTTP_USER_AGENT']; // Create a string containing user
details if (isset($_SERVER['HTTP_REFERER']))
$userDetails .= " {$_SERVER['HTTP_REFERER']}<br>\r\n"; else
$userDetails .= "<br>\r\n";
fwrite($fp, "$userDetails"); // Write the user details to the file rewind($fp); // Move the file
pointer to the start of the file $entries = 0;
while(!feof($fp)) { // Display each line in the file
$buffer = fgets($fp, 1024);
if ($buffer != "") {
echo $buffer;
$entries++;
}
} // Show a summary
echo "There are $entries entries in the log file<br>"; fclose ($fp);
?>
```

UNIT 2

Topics:

XML: Introduction to XML, Defining XML tags, their attributes and values, Document Type Definition, XML Schemes, Document Object Model, XHTML Parsing XML Data – DOM and SAX Parsers in java.

Introduction to XML

XML means **eXtensible Markup Language**. XML was developed by an organization called the World Wide Web Consortium (W3C) and is available as an open standard.

Markup is a set of instructions, often called tags. XML allows we to create our own self-descriptive tags and use them in XML document to satisfy application requirement. XML tags are not predefined like HTML tags. HTML supports a predefined set of tags and HTML document can use only those tags.

Since more such new tags may be defined, XML is said to be **extensible**.

We can create XML document using text editor and save it with “.xml” extension.

XML is derived from Standard Generalized Markup Language (SGML), which is a system for defining the markup Language.

XML is a case-sensitive meta-markup language because it allows we to create custom defined markup languages.

XML is used to describe structured data or information. HTML is designed to display data on the browser’s screen; they carry no information about the data. On the other hand, XML is designed to carry data. XML is used as a primary means to manipulate and transfer structured data over the web.

Role Of XML

XML plays a significant role in web development. XML document separates the content from their presentation. It does not specify how the content should be displayed. It only specifies the contents with their logical structure. The formatting task is done on an external style sheet; the look and feel may be changed very easily by choosing different style sheets.

XML also promotes easy data sharing among applications. Since application hold data in different structures, XML help us in mapping data from one to another. Each data structure is mapped to an agreed-upon XML structure. This XML structure may then be used by other applications. So, each application has to deal with only two structures: its own and the XML structure.

An XML document consists of the following parts:

- Prolog
- Body

Prolog

This part of the XML document contain the following parts:

- XML Declaration
- Optional Processing Instruction
- Comments
- Document Type Declaration

XML Declaration

Every XML document start with a one-line XML declaration. This declaration must be always the first statement in every XML document.

<?xml version="1.0" ?>

This declaration describes the document itself. This declaration is a instruction that tells the processing agent that the document is an XML document. The mandatory version attributes indicates the version used in the document. XML Declaration may use two optional attributes: encoding & standalone.

Processing Instruction

Processing instructions start with <? and end with ?>. The XML Declaration discussed above is a processing instruction.

Comments

Comments are used to describe what our program/ Block of code is going to do. Comments are ignored by XML parsers and applications using the XML document. An XML comments start with <!-- and ends with -->

<!-- comment text -->

The following point should be remembered while using the comments:

- Do not use double hyphen ‘--’
- Never place comments within a tag.
- Never place a comment before the XML Declaration.

Document Type Declaration[DTD]

The Document Type Declaration is used to specify the logical structure of the XML document. Each valid XML document has associated with DTD. This DTD usually held in a separate file, so it can be used with many documents. The DTD file holds the rules of grammar for a particular XML data structure. Those rules are used by the validating parser to check that is not only the valid XML file, but it also obeys its own internal rules.

Body

Body portion of the XML document contains textual data marked up by tags. It must have one element, called the document element or root element. Consider the following XML Document:

<?xml version="1.0" encoding="UTF-8"?>
<greeting>Hello World!!</greeting>

The name of the root element is <greeting>, which contain the text “Hello World!!”.

There can be one and only one root element. The following XML document is not valid as it have two top level elements, <greeting> and <message>

<?xml version="1.0" encoding="UTF-8"?>
<greeting>Hello World!!</greeting>
<message>World is beautiful</message>

The root element contains other elements which, in turn contain other elements:

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

```
<contact>
```

```
<person>
```

```
<name>My College</name>
```

```
<number>9876543210</number>
```

```
</person>
```

```
<person>
```

```
<name>Wers College</name>
```

```
<number>901234567</number>
```

```
</person>
```

```
</contact>
```

This XML document has the root element `<contact>` which has two `<person>` elements. Each `<person>` element has two elements, `<name>` and `<number>`

Elements

An XML element consists of starting tag, an ending tag, its contents and attributes. The content may be simple text, or other elements, or both. XML elements are used to store data.

The XML tag begins with the less than character ('<') and ends with the greater than ('>') character. Every tag must have the corresponding ending tag. The ending tag looks exactly like the starting tag except that it includes a slash('/') character before the tag name. All information in this element must be contained in the starting and ending tags.

Example:

```
<greeting> Hello world!! </greeting>
```

Here, `<greeting>` is the starting tag and `</greeting>` is the ending tag. Element have only text content "Hello World!!"

Attributes

Attributes are used to describe elements or to provide more information about element. They appear in the starting tag of the element.

Syntax:

```
<element-name attribute-name="attribute-value">....</element-name>
```

Example:

```
<employee gender="male">....</employee>
```

An element can also have multiple attributes:

```
<employee gender="male" id="12345">....</employee>
```

Well-Formed XML

An XML document is said to be well-formed if it contains tags and text that conforms with the basic XML well-formedness constraints.

The following rules must be followed by the XML document to be well-formed:

An XML document must have one and exactly one root element.

All tags must be closed

Every element must have a closing tag corresponding to its opening tag.

The following code is well formed because the opening tag `<name>` has its corresponding closing tag `</name>`

<name> CSE A-Section</name>

The following document fragment is not well-formed as there is no closing tag for the opening tag <name>

<name> CSE A-Section

A tag must be closed even if it does not have any content(empty element). In HTML , some tags such as ,
, <hr> do not require any closing tag. But in XML closing tag is always needed.

The following syntax is incorrect in XML:

Its correct representation is: **
 </br>**

Alternatively, XML has an abbreviated version as <br / >

All tags must be properly nested:

When we insert one tag within another, an ending must have the same name as the most recent unmatched tag.

If we open element A and then element B, we must first close B before closing A.

Right: <i>This text is bold and italic</i>

Wrong: <i>This text is bold and italic</i>

XML Tags are Case Sensitive

XML element names are case sensitive. The tag <Message> is different from the tag <message>.

Opening and closing tags must be written with the same case.

The following code is not well formed:

<Message>This is incorrect</message>

The following two elements are well formed:

<Message>This is correct</Message>

<message>This is also correct</message>

Attribute must always be Quoted

The value of an attribute must be quoted by double inverted comma.

The following syntax is correct:

<speed unit="rpm">7200</speed>

The following syntax is incorrect because the value rpm is written without any quotation mark:

<speed unit=rpm>7200</speed>

Certain characters are reserved for processing

Certain characters can't be used in the XML document directly. For example: '<' , '>' and '"' can't be used.

The following syntax is incorrect as it confuses XML parsers:

<condition> if salary < 1000 then </condition>

XML provides entities for these special character and XML parser will substitute each entity by its value.

An entity starts with an '&' character and ends with a ';'.

The following code is correct:

<condition>if salary < 1000 then </condition>

Predefined Entities:

Valid XML:

Valid XML documents are those that:

- are well formed.
- Comply with rules specified in the DTD or schema

The basic difference between well formed and valid XML document is that the former is defined without DTD or schema and the latter requires a DTD or Schema.

Validation

It is a method of checking whether an XML document is well-formed and confirms to the rules specified by a DTD or schema. Many tools are available to validate XML document against DTD or Schema. Linux provide one such application called xmllint for the validation.

Example: To validate sample.xml using xmllint with this command:

Xmllint - -valid sample.xml

Elements or Attributes

Data can be stored in elements or in attributes.

Example-1:

```
<person sex="male">
  <name>Aditya</name>
</person>
```

Example-2:

```
<person>
  <sex>male</sex>
  <name>Anna</name>
</person>
```

In the first example sex is an attribute. In the second, sex is an element. Both examples provide the same information.

So, When do we use elements and when do we use attributes for presenting our information?

There are no rules about when to use attributes, and when to use an elements.

However, it is recommended not to use attribute as far as possible due to the following reason:

- Too many attributes reduce the readability of an XML document.
- Attributes cannot contain multiple values (child elements can).
- Attributes are not easily expandable (for future changes).
- Attribute values are not easy to check against DTD

XML Schema Languages

•An XML Schema represents the interrelationship between the elements and attributes in an XML document. An XML schema is a language for expressing constraints on the structure and content of XML documents.

- There are several different schema languages W3C XML Schema, RELAX NG and Document Type Definition (DTD) is one such XML Schema Language.
- Two primary XML schema language, namely, DTD – which is very popular and is widely used and W3C XML Schema- which is known to be an extremely powerful schema language. Document Type Definition (DTD) and XML Schema is in our syllabus.

Introduction to DTD

Document Type Definition (DTD) is one of several XML Schema Language.

Purpose Of DTD

As we know there are two types of XML document (i) Well-formed (ii) Valid

An XML document is said to be Valid with respect to DTD, if it confirms the rules specified by a given XML DTD.

- The purpose of DTDs is to provide a framework for validating XML documents.
- The purpose of a DTD is to define the legal building blocks of an XML document.
- A DTD defines the document structure with a list of legal elements and attributes.
- DTD used to specify a content model for each elements and attribute used in an XML document.
- DTD can't specify all the constraints that may arise in XML document, But Large number of constraints can be specified using XML Schema.

Using a DTD in an XML document

To validate XML document against a DTD, we should know how DTD and XML documents are linked and where validator find the DTD.

The keyword DOCTYPE is used to make such link.

There are three ways to make this link:

- 1) The DTD can be embedded directly in the XML document as part of it- called **internal DTD**.
- 2) From an XML document, a reference to an external file containing DTD can be made- called **external DTD**.
- 3) Both internal and external DTD can be used.

Declaration of Simple element greeting:

<!ELEMENT greeting (#PCDATA)>

Above line in our DTD allows the greeting element to contain non-markup data in our XML document. So, following is a valid usage:

<greeting>Hello World!!</greeting>

PCDATA is the only available text type for simple element declaration.

Sometimes it is wrongly specified as CDATA. Text specified as CDATA will not be parsed and tags/entities will not be treated as markups.

If we want to use some text that probably contains illegal character such as '<', '>' and '&', then it can be embedded within "<![CDATA[" and "]]>".

JavaScript code contains illegal character, So it is good to use the CDATA section to use JavaScript code in our XML document.

Example:

```
<code>
<![CDATA[
<script language="JavaScript">
function less(x, y) {
    return x<y;
}
a=less(2,3);
</script>
]]>
</code>
```

in the above example, Everything inside CDATA section is ignored by the parser.

Rules to be followed using CDATA section:

- A CDATA section can't contain the string "]]"
- The "<![CDATA[" and "]]>" which marks the beginning and end of CDATA section can't contain spaces or line breaks.
- Nested CDATA section are neither allowed nor needed.

Example:

```
<![ELEMENT message (#PCDATA)]>
<![ELEMENT title (#PCDATA)]>
<![ELEMENT firstName (#PCDATA)]>
<![ELEMENT surname (#PCDATA)]>
<![ELEMENT dob (#PCDATA)]>
```

Compound Elements

Compound element can contain other elements, an element contain one child element. Here's how we can do that:

```
<![ELEMENT element_name (child_element_name) ]>
```

Example:

```
<![ELEMENT employee (name) ]>
```

Above statement indicates the element employee contain another child element name exactly once. The child element name must be declared separately.

Here is a complete declaration:

```
<?xml version="1.0"?>
<![DCOTYPE employee [
<!-- 'employee have one child element 'name' -->
<![ELEMENT employee (name)]>
<!-- namecontain only text -->
<![ELEMENT name (#PCDATA)]>
]]>
<employee>
<name> UK Roy</name>
```


</employee>

Occurrence indicators

Occurrence indicator specifies the number of times an element occurs in a document.

When no occurrence indicator is specified, the child element must occur exactly once in an XML document.

List of occurrence indicator:

- NONE – Exactly one occurrence
- *(Asterisk) - Zero or more occurrence
- o Example: <!ELEMENT employee (contact*)>

Indicates that element employee can have child element contact any number of times.

- + (Plus Sign) – One or More Occurrence. Occurs at least once within its enclosing element.

- o Example: <!ELEMENT employee (contact+)>

Indicates that company contain employee one or more times. A company can't exist without any employee.

- o Consider element book, must have at least one author, can be declared as:

<! ELEMENT book (author+)>

- ?(Question Mark) – Zero or one Occurrence
- o Example: <!ELEMENT employee (PAN?)>

Element employee can have optional PAN

Declaring Multiple children

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

Operators which can be specified when declaring elements with multiple children:

, [sequence] – a followed by b

| [choice] –

() [singleton]

Declaring Multiple children

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

Operators which can be specified when declaring elements with multiple children:

Sequence operator [,]

The sequence operator (,) is used to enforce the ordering of child elements.

The general syntax for using sequence operator is:

<!ELEMENT parent (expr1, expr2, expr3, ...) >

expr1, expr2, expr3... must occur in the document in the order as specified.

Example:

<! ELEMENT employee (firstName, middleName, lastName) >

<!ELEMENT firstName (#PCDATA) >

<!ELEMENT middleName (#PCDATA) >

<!ELEMENT lastName (#PCDATA) >

Above declaration requires firstName followed by middleName and lastName.

So the following XML document fragment is valid:

```
<employee>
<firstName>Uttam</firstName>
<middleName>Kumar</middleName>
<lastName>Roy</lastName>
</employee>
```

But, the following is not valid:

```
<employee>
<firstName>Uttam</firstName>
<lastName>Roy</lastName>
<middleName>Kumar</middleName>
</employee>
```

Choice Operator [|]

The choice operator (|) is used to select one from among choices separated by ‘|’ symbol.

General Syntax of using choice operator:

<!ELEMENT parent (expr1 | expr2 | expr3 | ...) >
expr1 | expr2 | expr3 | ... can occur as the child of the element parent.

Example:

<! ELEMENT product (price | discountprice) >

In above declaration, element product can have either price or discountprice as child.

<! ELEMENT tutorial (name | title | subject) >

Element tutorial can have either a name, a title or a subject but not all

Composite Operator [()]

An expression surrounded by parenthesis is treated as single unit (singleton) and could have any one of the following occurrence indicator : ? , * , or + .

Example:

<! ELEMENT biodata (dob, (company , title) * , contact +) >

<!ELEMENT dob (#PCDATA) >

<!ELEMENT company (#PCDATA) >

<!ELEMENT title (#PCDATA) >

<!ELEMENT contact (#PCDATA) >

In the above example, company and title form a single unit. The biodata element can contain Zero or more such units.

ATTRIBUTE DECLARATION

- If an element have attributes, we have to declare the name of the attributes in DTD.
- An XML attribute is always in the form of a (name, value) pair with element.
- Attributes are useful as it gives more information about an element.
- Attributes declaration is done with ATTLIST declarations.

- A single ATTLIST can declare multiple attributes for a single element type.
- The declaration starts with ATTLIST followed by the name of the element the attributes are associated with, and the declaration of the individual attributes.

Syntax

Syntax of Attributes declaration is as follows:

```
<!ATTLIST element-name attribute-name attribute-type default-value>
```

where, element-name specifies the name of the element to which the attribute attribute-name applies.

Example:

```
<!ELEMENT line EMPTY>
```

```
<!ATTLIST line width CDATA '100'>
```

This specifies the element line has an optional attribute width with value '100', if attribute is not mentioned.

Rules to be followed while declaring an Attribute:

- The keyword ATTLIST must be in upper case
- All attributes used in an XML document must be declared in the Document Type Definition (DTD)
- Attributes may only appear in start or empty tags.
- An element can have any number of Attributes.

Default value of Attribute

Default – This is optional means XML document author may or may not provide this attribute.

Default value is enclosed in a single or double quote.

Example:

```
<! ATTLIST line width CDATA '100'>
```

This declaration says that the attribute width is optional. If the value of this attribute is specified then the value of this attribute will be the value specified. If it does not occur, width attribute will get the value 100 from XML processor.

```
<line width='200' />
```

width attribute is specified with value 200

```
<line />
```

no width attribute is specified, XML processor will put value 100

#REQUIRED

This means that the attribute must be specified with a value every time the element is listed.

#REQUIRED keyword is used to specify that the attribute must be provided.

```
<! ATTLIST price currency CDATA #REQUIRED> //currency must appear for price
```

Example:

```
<price currency='INR'>100</price>
```

VALID

```
<price>100</price>
```

NOT VALID

XML Schemas

The purpose of an XML Schema is to define the legal building blocks of an XML document. It is used to represent data types of an XML document. It is an alternative to DTD (Document Type Definition). XML Schema defines elements, attributes and values of elements and attributes.

An XML Schema:

- defines elements that can appear in a document
- defines attributes that can appear in a document
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

XML Schemas Data Types

One of the greatest strength of XML Schemas is the support for data types. Using XML schema it is easier to describe allowable document content; it is easier to validate the correctness of data; it is easier to convert data between different data types.

Built in data types supported by XML schema

XML Schema has a lot of built-in data types.

The most common types are:

- xs:string
- xs:decimal
- xs:integer
- xs:boolean
- xs:date
- xs:time

There are two classes of XML schema data types:

- Complex type is a data type is represented using markup.
- Simple type is a data type whose values are represented in XML doc by character data.

XML markup such as <types> is known as XML schema that conforms to W3C defined XML schema vocabulary which defines all or part of the vocabulary for another XML document. The <schema> is the root element for any XML schema document. The child elements of schema define the data types.

Example for writing a simple XML Schema

Step 1: Write a xsd file in which the desired structure of the XML document is defined and named it as StudentSchema.xsd.

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="URL">
    <xs:element name="student" >
        <xs:complexType>
            <xs:sequence>
                <xs:element name="name" value="xs:string" />
                <xs:element name="regno" value="xs:string" />
                <xs:element name="dept" value="xs:string" />
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

The xs qualifier used to identify the schema elements and its types. The xs:schema is the root element. It takes the attributes xmlns:xs which has the value, "http://www.w3.org/2001/XMLSchema. This declaration indicates that the document follows the rule of XMLSchema defined by W3 in year 2001.

The xs:element is used to define the xml element. The Student element is complexType which has three child elements: name, regno and dept. All these elements are of simple type string.

Step 2: Write a XML document and reference the xsd file. Named it as myXML.xml

```
<?xml version="1.0"?>
<student xmlns:xsi="URL"
        xsi:schemaLocation="StudentSchema.xsd">
  <name> Raj </name>
  <regno>3212654556</regno>
  <dept>CSE</dept></student>
```

The following code, xsi:schemaLocation="StudentSchema.xsd" is used to reference the XML schema document (xsd) file which contains the definition of this xml document.

Various xml validation tools are available which can be used to validate xml and its schema document.

Advantages

- Supports data types
- Supports namespaces
- XML schemas support a set of data types, similar to the ones used in most common programming languages
- Provides the ability to define custom data types
- Easy to describe allowable content in the document
- Easy to validate the correctness of data
- Object oriented approach like inheritance and encapsulation can be used in creating the document
- Easier to convert data between different data types
- Easy to define data formats
- Easy to define restrictions on data
- It is written in xml so any xml editor can be used to edit xml schema

Xml Parser can be used to parse the schema file

- XML schemas are more powerful than DTDs. Everything that can be defined by the DTD can also be defined by schemas, but not vice versa.

Disadvantages

- Complex to design and learn
- Maintaining XML schema for large XML document slows down the processing of XML document

DTD for the above xml file instead of XML schema

The purpose of a DTD (Document Type Definition) is to define the legal building blocks of an XML document.

A DTD defines the document structure with a list of legal elements and attributes.

Step 1: Create a DTD. Name it as student.dtd

```
<!ELEMENT student(name, regno, dept)>
```

```
<!ELEMENT name(#PCDATA)>
```

```
<!ELEMENT regno(#PCDATA)>
```

```
<!ELEMENT dept(#PCDATA)>
```

!ELEMENT student defines that the note element contains three elements: "name, regno, dept"

PCDATA means parsed character data which is the text found between the start tag and the end tag of an XML element.

Step 2: Write the XML document and reference the DTD file in it. Name it as myXML.xml

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

```
<!DOCTYPE student SYSTEM student.dtd>
```

```
<student>
```

```
<name> Raj </name>
```

```
<regno>3212654556</regno>
```

```
<dept>CSE</dept>
```

```
</student>
```

!DOCTYPE student defines that the root element of this document is student and links the myXML.xml file with the student.dtd file.

Document Object Model:

Document Object Model is for defining the standard for accessing and manipulating XML documents. XML DOM is used for

- Loading the xml document
- Accessing the xml document
- Deleting the elements of xml document
- Changing the elements of xml document

According to the DOM, everything in an XML document is a node. It considers

- The entire document is a document node
- Every XML element is an element node
- The text in the XML elements are text nodes
- Every attribute is an attribute node
- Comments are comment nodes

DOM Levels

- Level 1 Core: W3C Recommendation, October 1998
 - o It has feature for primitive navigation and manipulation of XML trees
 - o other Level 1 features are: All HTML features
- Level 2 Core: W3C Recommendation, November 2000
 - o It adds Namespace support and minor new features
 - o other Level 2 features are: Events, Views, Style, Traversal and Range

- Level 3 Core: W3C Working Draft, April 2002
- o It supports: Schemas, XPath, XSL, XSLT

We can access and parse the XML document in two ways:

- o Parsing using DOM (tree based)
- o Parsing using SAX (Event based)

Parsing the XML doc. using DOM methods and properties are called as tree based approach whereas using SAX (Simple Api for Xml) methods and properties are called as event based approach.

DOM based XML Parsing:(tree based)

JAXP is a tool, stands for Java Api for Xml Processing, used for accessing and manipulating xml document in a tree based manner.

In this approach, to access XML document, the document object model implementation is defined in the following packages:

- javax.xml.parsers
- org.w3c.dom

The following DOM java Classes are necessary to process the XML document:

- DocumentBuilderFactory class creates the instance of DocumentBuilder.
- DocumentBuilder produces a Document (a DOM) that conforms to the DOM specification

Parsing XML Data: DOM and SAX Parser in JAVA

DOM Parser

An alternative to SAX approach is DOM. In a XML processor, the parser builds the DOM tree for an XML document. The nodes of the tree are represented as objects that can be accessed and manipulated by the application.

The advantages of DOM over SAX are:

- DOM is better for accessing a part of an XML document more than once.
- DOM is better for rearranging (sorting) the elements in a XML document.
- DOM is best for random access over SAX's sequential access.
- DOM can detect invalid nodes later in the document without any further processing.

The disadvantages of DOM over SAX are:

- DOM structure (tree) is stored entirely in the memory, so large XML documents require more memory.
- Large documents cannot be parsed using DOM.
- DOM is slower when compared to SAX.

Below is an example Java program which reads an XML document using DOM API:

//file.xml - XML document

```

<?xml version="1.0"?>
<company>
<staff id="1001">
<firstname>yong</firstname>
<lastname>mook kim</lastname>
<nickname>mkyong</nickname>
<salary>100000</salary>
</staff>
<staff id="2001">
<firstname>low</firstname>
<lastname>yin fong</lastname>
<nickname>fong fong</nickname>
<salary>200000</salary>
</staff>
</company>

```

//ReadXMLFile.java - Java File

```

import javax.xml.parsers.DocumentBuilderFactory; import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document; import org.w3c.dom.NodeList; import org.w3c.dom.Node;
import org.w3c.dom.Element; import java.io.File;

```

```

public class ReadXMLFile {
public static void main(String argv[]) { try {
File fXmlFile = new File("/Users/mkyong/staff.xml"); DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance(); DocumentBuilder dBuilder =
dbFactory.newDocumentBuilder();
Document doc = dBuilder.parse(fXmlFile); doc.getDocumentElement().normalize();
System.out.println("Root element :" + doc.getDocumentElement().getNodeName()); NodeList
nList = doc.getElementsByTagName("staff");
System.out.println("-----");
for (int temp = 0; temp < nList.getLength(); temp++) { Node nNode = nList.item(temp);
System.out.println("\nCurrent Element :" + nNode.getNodeName()); if (nNode.getNodeType()
== Node.ELEMENT_NODE) {
Element eElement = (Element) nNode; System.out.println("Staff id : " +
eElement.getAttribute("id"));
System.out.println("First Name : " +
eElement.getElementsByTagName("firstname").item(0).getTextContent());
System.out.println("Last Name : " +
eElement.getElementsByTagName("lastname").item(0).getTextContent());
System.out.println("Nick Name : " +
eElement.getElementsByTagName("nickname").item(0).getTextContent());
System.out.println("Salary : " +
eElement.getElementsByTagName("salary").item(0).getTextContent());
}
}
} catch (Exception e) { e.printStackTrace();

```


}}}

Output:

Root element :company

Current Element :staff Staff id : 1001

First Name : yong

Last Name : mook kim Nick Name : mkyong Salary : 100000

Current Element :staff Staff id : 2001

First Name : low Last Name : yin fong

Nick Name : fong fong Salary : 200000

SAX Parser

SAX stands for Simple API for XML

SAX provides a mechanism for reading data from an XML document, SAX parsers operate on each piece of the XML document sequentially.

It is a kind of event oriented approach for parsing the xml document.

An XML tree is not viewed as a data structure, but as a stream of events generated by the parser.

The kinds of events are:

- the start of the document is encountered
- the end of the document is encountered
- the start tag of an element is encountered
- the end tag of an element is encountered
- character data is encountered
- a processing instruction is encountered

Scanning the XML file from start to end, each event invokes a corresponding callback method that the programmer writes.

SAX packages

- javax.xml.parsers: Describing the main classes needed for parsing
- org.xml.sax: Describing few interfaces for parsing

SAX classes

- SAXParser Defines the API that wraps an XMLReader implementation class
- SAXParserFactory Defines a factory API that enables applications to configure and obtain a SAX based parser to parse XML documents
- ContentHandler Receive notification of the logical content of a document.
- DTDHandler Receive notification of basic DTD-related events.
- EntityResolver Basic interface for resolving entities.
- ErrorHandler Basic interface for SAX error handlers.
- DefaultHandler Default base class for SAX event handlers.

SAX parser methods

- startDocument() and endDocument() – methods called at the start and end of an XML document.

- startElement() and endElement() – methods called at the start and end of a document element.
- characters() – method called with the text contents in between the start and end tags of an XML document element.

The SAX approach to processing is known as event processing. The processor scans the document from beginning to end sequentially. Every time a syntactic structure like opening tag, attributes, text or a closing tag is recognized, the processor signals an event to the application by calling an event handler for the particular structure that was found. The interfaces that describe the event handlers form the SAX API.

Below is an example Java program which reads an XML document using SAX API:

```
//file.xml - XML document
```

```
<?xml version="1.0"?>
<company>
<staff>
<firstname>yong</firstname>
<lastname>mook kim</lastname>
<nickname>mkyong</nickname>
<salary>100000</salary>
</staff>
<staff>
<firstname>low</firstname>
<lastname>yin fong</lastname>
<nickname>fong fong</nickname>
<salary>200000</salary>
</staff>
</company>
```

```
//ReadXMLFile.java - Java File import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory; import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;
```

```
public class ReadXMLFile {
public static void main(String argv[]) { try {
SAXParserFactory factory = SAXParserFactory.newInstance(); SAXParser saxParser =
factory.newSAXParser(); DefaultHandler handler = new DefaultHandler() {
boolean bfname = false; boolean blname = false; boolean bnname = false; boolean bsalary =
false;
public void startElement(String uri, String localName,String qName, Attributes attributes)
throws SAXException {
System.out.println("Start Element :"+ qName); if (qName.equalsIgnoreCase("FIRSTNAME"))
{
bfname = true;
```

```

    } if (qName.equalsIgnoreCase("LASTNAME")) { blname = true;
    } if (qName.equalsIgnoreCase("NICKNAME")) { bnname = true;
    } if (qName.equalsIgnoreCase("SALARY")) { bsalary = true;
    } }
    public void endElement(String uri, String localName, String qName) throws SAXException {
        System.out.println("End Element : " + qName);
    } public void characters(char ch[], int start, int length) throws SAXException { if (bfname) {
        System.out.println("First Name : " + new String(ch, start, length)); bfname = false;
    } if (blname) {
        System.out.println("Last Name : " + new String(ch, start, length)); blname = false;
    } if (bnname) {
        System.out.println("Nick Name : " + new String(ch, start, length));
        bnname = false;
    } if (bsalary) {
        System.out.println("Salary : " + new String(ch, start, length)); bsalary = false;
    } } }
    saxParser.parse("c:\\file.xml", handler);
} catch (Exception e)
{ e.printStackTrace();
} } }

```

Output:

```

Start Element :company Start Element :staff Start Element :firstname First Name : yong
End Element :firstname Start Element :lastname Last Name : mook kim End Element :lastname
Start Element :nickname Nick Name : mkyong End Element :nickname Start Element :salary
Salary : 100000
End Element :salary End Element :staff Start Element :staff
Start Element :firstname First Name : low
End Element :firstname Start Element :lastname Last Name : yin fong End Element :lastname
Start Element :nickname Nick Name : fong fong End Element :nickname Start Element :salary
Salary : 200000
End Element :salary End Element :staff
End Element :company

```

Unit-3 Servlets

Introduction to web servers:

What is web?

- ✓ A collection of cross-linked “websites” which uses URI.
- ✓ The consistent use of URIs to represent resources.
- ✓ HTTP, HTML, and everything built around them(web)
- ✓ Which provides to invoke the data across universally over the net

What is server?

- ✓ A server is a **computer or device** on a network that manages network resources.
- ✓ Most servers are dedicated. This means that they perform only one task rather than multiple tasks on multiprocessing operating systems, however, a single computer can execute several programs at once

What is web server?

- ✓ A Web server is a **program** that generates and transmits responses to **client requests** for Web resources.
- ✓ Handling a client request consists of **several key steps**:
 - Parsing the request message
 - Checking that the request is authorized
 - Associating the URL in the request with a file name
 - Constructing the response message
- Transmitting the response message to the requesting client
- ✓ The server can generate the response message in a variety of ways:
 - The server simply retrieves the file associated with the URL and returns the contents to the client.
 - The server may invoke a script that communicates with other servers or a back-end database to construct the response message.

Web Site versus Web Server?

- ✓ **Web site** and **Web server** are different:
 - A **Web site** consists of a collection of Web pages associated with a particular hostname.
 - A **Web server** is a program to satisfy client requests for Web resources.

Types Of Web Servers:

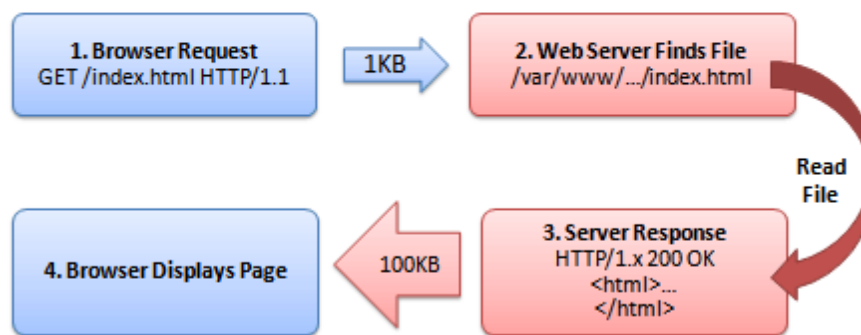
1. Apache Web Server
2. IIS Server
3. Xampp Server
4. WAMP Server

Apache Web Server:

Introduction:

- ✓ Apache Web server is the most commonly used http server today. About 80% of all websites and Intranets use Apache web server to deliver their content to requesting Browsers.
- ✓ Server side programming languages such as PHP, Perl, Python, Java and many others
- ✓ The name "**Apache**" derives from the word "**patchy**" that the Apache developers used to describe early versions of their software.
- ✓ The Apache Web server provides a full range of Web server features, including CGI, SSL, and virtual domains. Apache also supports plug-in modules for extensibility. Apache is reliable, free, and relatively easy to configure.
- ✓ Apache is free software distributed by the **Apache Software Foundation**. The Apache Software Foundation promotes various free and open source advanced Web technologies.
- ✓ It can be downloaded and used completely free of cost. The first version of **Apache web server**, based on the NCSA httpd Web server, was developed in 1995.
- ✓ Apache is developed and maintained by an open community of developers under the auspices of the **Apache Software Foundation**.

HTTP Request and Response



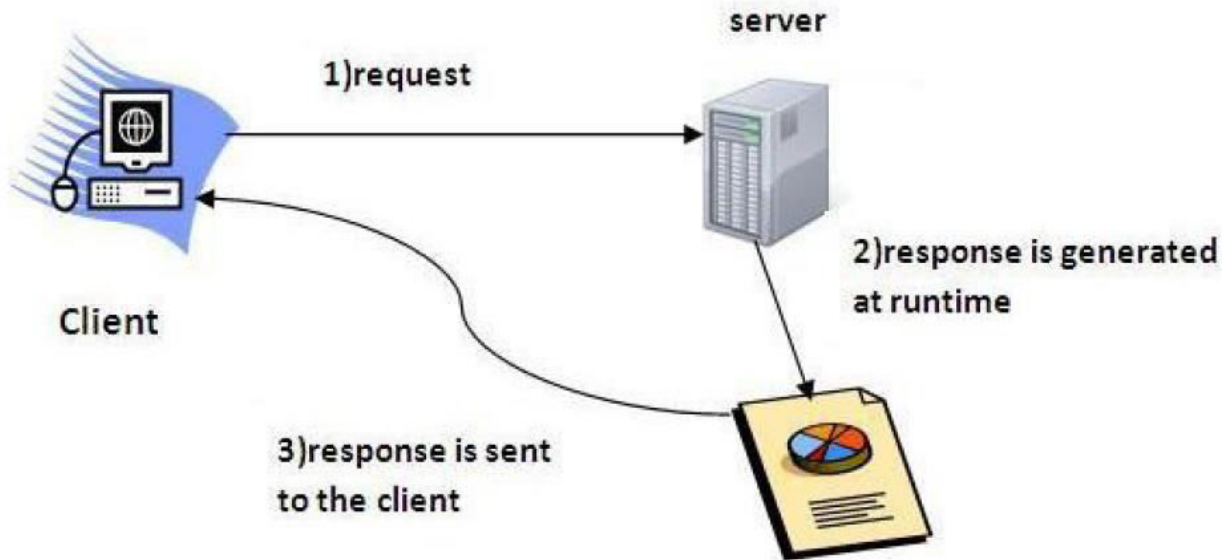
Servlet technology is used to create web application (resides at server side and generates dynamic web page).

A **Java servlet** is a Java program that extends the capabilities of a server . Web servlets are the Java counterpart to other dynamic Web content technologies such as PHP and ASP.NET.

What is a Servlet?

Servlet can be described in many ways, depending on the context.

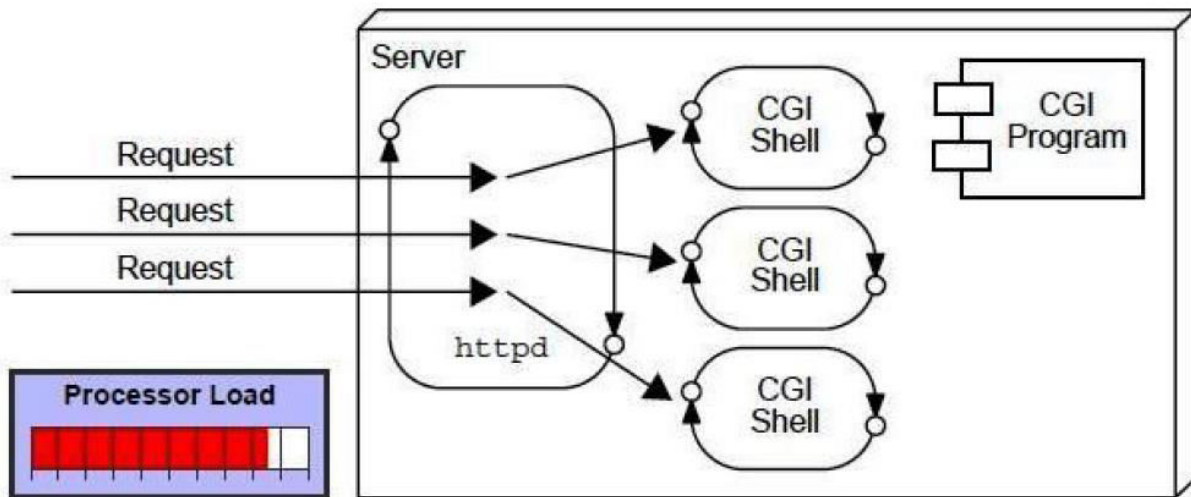
- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page



Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

CGI(Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



Disadvantages of CGI

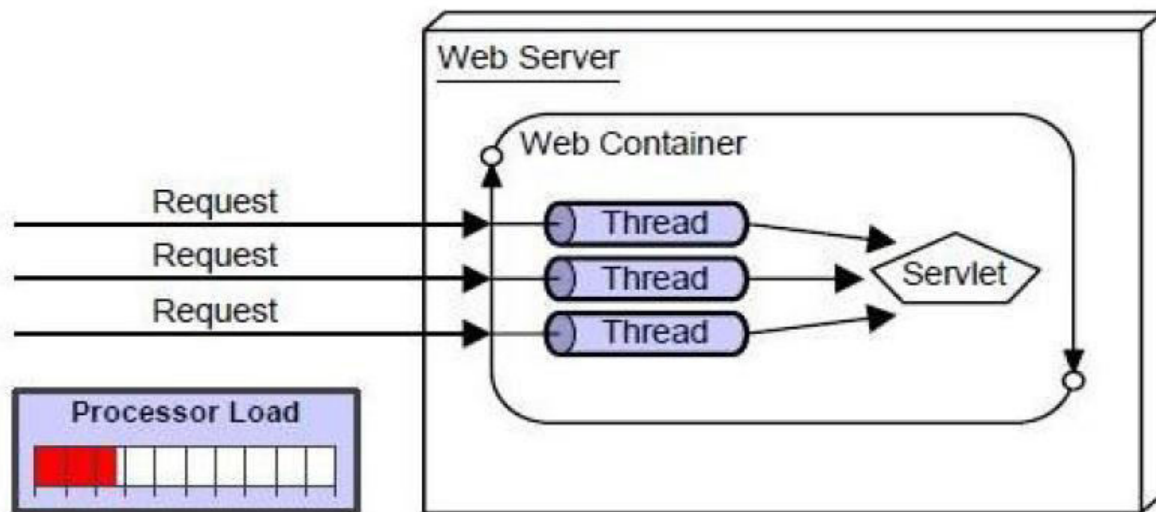
There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low. The basic benefits of servlet are as follows:

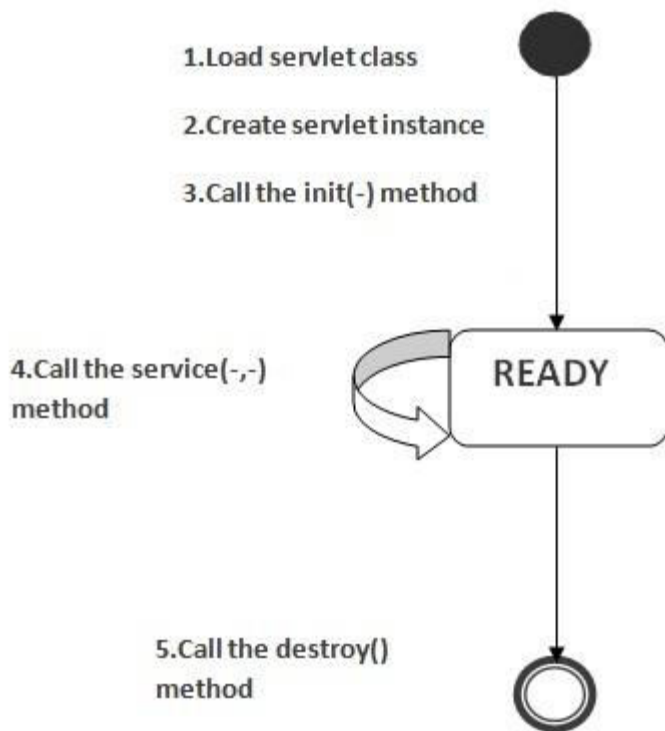
1. **better performance:** because it creates a thread for each request not process.
2. **Portability:** because it uses java language.
3. **Robust:** Servlets are managed by JVM so we don't need to worry about memory leak, garbage collection etc.
4. **Secure:** because it uses java language.



Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked.
5. destroy method is invoked.



As displayed in the above diagram, there are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the `init()` method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the `destroy()` method, it shifts to the end state

1) Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

2) Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

3) `init` method is invoked

The web container calls the `init` method only once after creating the servlet instance. The `init` method is used to initialize the servlet. It is the life cycle method of the `javax.servlet.Servlet` interface. Syntax of the `init` method is given below:

`public void init(ServletConfig config) throws ServletException`

4) `service` method is invoked

The web container calls the `service` method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the `service`

method. If servlet is initialized, it calls the service method. Notice that servlet is initialized only once. The syntax of the service method of the Servlet interface is given below:

5) destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc. The syntax of the destroy method of the Servlet interface is given below:

public void destroy()

The servlet is initialized by calling the **init ()** method.

The servlet calls **service()** method to process a client's request.

The servlet is terminated by calling the **destroy()** method.

Finally, servlet is garbage collected by the garbage collector of the JVM. Now let us discuss the life cycle methods in details.

The init() method :

The init method is designed to be called only once. It is called when the servlet is first created, and not called again for each user request. So, it is used for one-time initializations, just as with the init method of applets

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started. When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The service() method :

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client. Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

**public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException{ }**

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client. The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //      Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    //      Servlet code
}
```

The destroy() method :

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this:

```
public void destroy() {
// Finalization code...
}
```

Servlet Deployment:

By default, a servlet application is located at the path <Tomcat-installation-directory>/webapps/ROOT and the class file would reside in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/classes. If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installation-

directory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file

located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

<servlet>

```
<servlet-name>HelloWorld</servlet-name>
<servlet-class>HelloWorld</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>HelloWorld</servlet-name>
<url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind. You are almost

done, now let us start tomcat server using

<Tomcat-installation-directory>\bin\startup.bat (on windows) or

<Tomcat-installation-directory>/bin/startup.sh

(onLinux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in browser's address box. If everything goes fine, you would get result:

Servlet API

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api. The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only. Let's see what are the interfaces of javax.servlet package.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

K.Yellaswamy ,AssistantProfessor|ACE College of Engineering & Technology E-mail:toyellaswamy@gmail.com

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api. The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
- 4
9. FilterConfig
10. FilterChain

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletException
3. ServletInputStream
4. ServletOutputStream
5. ServletRequestWrapper
6. ServletResponseWrapper
7. ServletRequestEvent
8. ServletContextEvent

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
- 5
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

Servlet interface provides common behaviour to all the servlets.

Servlet interface needs to be implemented for creating any servlet (either directly or indirectly).

It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and

to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

| Method | Description |
|--|---|
| public void init(ServletConfig config) | initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once. |
| public void service(ServletRequest request, ServletResponse response) | provides response for the incoming request. It is invoked at each request by the web container. |
| public void destroy() | is invoked only once and indicates that servlet is being destroyed. |
| public ServletConfig getServletConfig() | returns the object of ServletConfig. |
| public String getServletInfo() | returns information about servlet such as writer, copyright, version etc. |

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

HttpServlet class

Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req, ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles

the HEAD request. It is invoked by the web container.

6. protected void doOptions(HttpServletRequest req, HttpServletResponse res) handles the OPTIONS request. It is invoked by the web container.

7. protected void doPut(HttpServletRequest req, HttpServletResponse res) handles the PUT request. It is invoked by the web container.

8. protected void doTrace(HttpServletRequest req, HttpServletResponse res) handles the TRACE request. It is invoked by the web container.

9. protected void doDelete(HttpServletRequest req, HttpServletResponse res) handles the DELETE request. It is invoked by the web container.

10. protected long getLastModified(HttpServletRequest req) returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

ServletRequest Interface

An object of ServletRequest is used to provide the client request information to a servlet such as content type, content length, parameter names and values, header informations, attributes etc.

Methods of ServletRequest interface

There are many methods defined in the ServletRequest interface. Some of them are as follows:

| Method | Description |
|--|--|
| public String getParameter(String name) | is used to obtain the value of a parameter by name. |
| public String[] getParameterValues(String name) | returns an array of String containing all values of given parameter name. It is mainly used to obtain values of a Multi select list box. |

| | |
|---|---|
| java.util.Enumeration getParameterNames() | returns an enumeration of all of the request parameter names. |
| public int getContentLength() | Returns the size of the request entity data, or -1 if not known. |
| Public String getCharacterEncoding() | Returns the character set encoding for the input of this request. |
| public String getContentType() | Returns the Internet Media Type of the request entity data, or null if not known. |
| public ServletInputStream getInputStream()throws IOException | Returns an input stream for reading binary data in the request body. |
| Public abstract String getServerName() | Returns the host name of the server that received the request. |
| public int getServerPort() | Returns the port number on which this request was received. |

Example of ServletRequest to display the name of the user In this example, we are displaying the name of the user in the servlet. For this purpose, we have used the `getParameter` method that returns the value for the given request parameter name.

Servlet API provides two important interfaces **ServletResponse** and **HttpServletResponse** to assist in sending response to client.

Some Important Methods of ServletResponse

| Methods | Description |
|----------------------------------|---|
| PrintWriter getWriter() | returns a PrintWriter object that can send character text to the client. |
| void setBufferSize(int size) | Sets the preferred buffer size for the body of the response |
| void setContentLength(int len) | Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header |
| void setContentType(String type) | sets the content type of the response being sent to the client before sending the response. |
| void setBufferSize(int size) | sets the preferred buffer size for the body of the response. |
| boolean isCommitted() | returns a boolean indicating if the response has been committed |

| | |
|----------------------------|--|
| void setLocale(Locale loc) | sets the locale of the response, if the response has not been committed yet. |
|----------------------------|--|

HttpServletResponse Interface Methods

Some Important Methods of HttpServletResponse

| Methods | Description |
|---|---|
| void addCookie(Cookie cookie) | adds the specified cookie to the response. |
| void sendRedirect(String location) | Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer |
| int getStatus() | gets the current status code of this response |
| String getHeader(String name) | gets the value of the response header with the given name. |
| void setHeader(String name, String value) | sets a response header with the given name and value |
| void setStatus(int sc) | sets the status code for this response |
| void sendError(int sc, String msg) | sends an error response to the client using the specified status and clears the buffer |

Reading Servlet Parameters :

The ServletRequest class includes methods that allow you to read the names and values of parameters that are included in a client request. We will develop a servlet that illustrates their use.

The example contains two files.

A Web page is defined in **sum.html** and a servlet is defined in **Add.java**

sum.html:

```
<html>
<body>
<center>
<form name="Form1" method="post"
action="Add">
<table>
<tr>
<td><B>Enter First Number</td>
<td><input type="text" name="Enter First Number" size="25" value=""></td>
</tr>
<tr>
<td><B>Enter Second Number</td>
<td><input type="text" name="Enter Second Number" size="25" value=""></td>
</tr>
</table>
<input type="submit" value="Submit">
</body>
</html>
```

The HTML source code for sum.html defines a table that contains two labels and two text fields. One of the labels is Enter First Number, and the other is Enter Second Number. There is also a submit button. Notice that the action parameter of the form tag specifies a URL. The URL identifies the servlet to process the HTTP POST request.

Add.java

```
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class Add
extends HttpServlet
{
public void doPost(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException
```

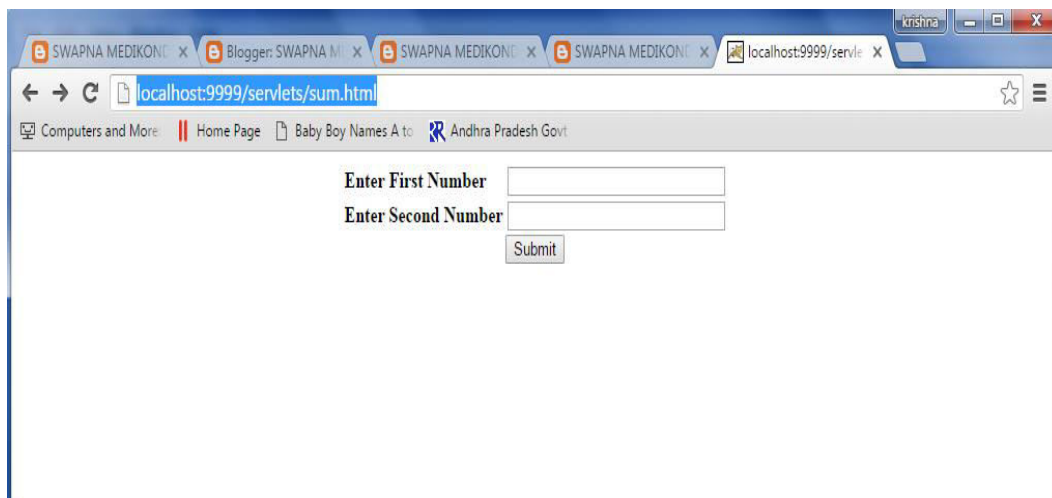
```

{
// Get print writer.
response.setContentType("text/html");
PrintWriter pw = response.getWriter();
// Get enumeration of parameter names.
Enumeration e = request.getParameterNames();
// Display parameter names and values.
int sum=0;
while(e.hasMoreElements())
{
String pname = (String)e.nextElement();
pw.print(pname + " = ");
String pvalue = request.getParameter(pname);
sum+=Integer.parseInt(pvalue);
pw.println(pvalue);
}
pw.println("Sum = "+sum);
pw.close();
}
}

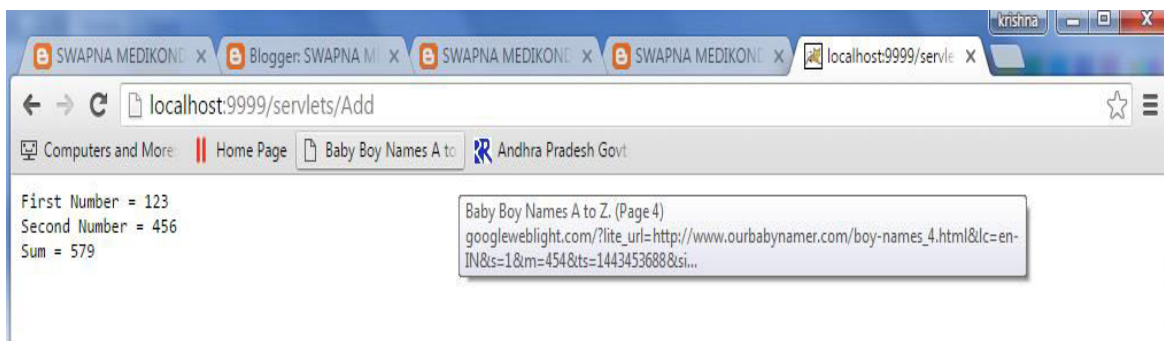
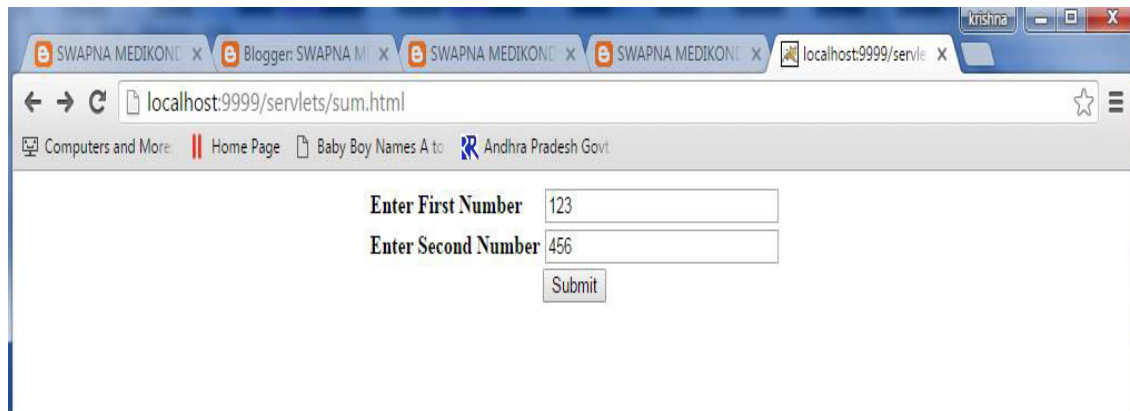
```

The source code for Add.java contains doPost() method is overridden to process client requests. The getParameterNames() method returns an enumeration of the parameter names. These are processed in a loop. we can see that the parameter name and value are output to the client. The parameter value is obtained via the getParameter() method.

after typing URL : <http://localhost:9999/servlets/sum.html>



The screenshot shows a web browser window with the address bar displaying `localhost:9999/servlets/sum.html`. The browser has several tabs open, including "SWAPNA MEDIKONI" and "Blogger: SWAPNA M". The page content consists of a simple form with two text input fields. The first field is labeled "Enter First Number" and the second field is labeled "Enter Second Number". Below these fields is a "Submit" button. The browser's address bar also shows a search bar with the text "Computers and More: || Home Page | Baby Boy Names A to | Andhra Pradesh Govt".



Reading Initialization Parameters

The servlet container provides the initialization parameters for a servlet or filter within the configuration object the container passes to the init method. The configuration object provides `getInitParameter()` function that takes a *std::string* name and returns the contents of the initialization parameter by that name. A filter uses the configuration object to access initialization parameters. A servlet uses the convenience functions provided by *rwfs::GenericServlet* to access initialization parameters directly.

Reading Initialization parameters:

Syntax to provide the initialization parameter for a servlet

The `init-param` sub-element of `servlet` is used to specify the initialization parameter for a servlet.

```
<web-app>
  <servlet>
    .....

    <init-param>
      <param-name>parametername</param-name>
      <param-value>parametervalue</param-value>
    </init-param>
    .....
  </servlet>
</web-app>
```

ServletConfig to get initialization parameter

In this example, we are getting the one initialization parameter from the `web.xml` file and printing this information in the servlet.

```
DemoServletInit.java
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

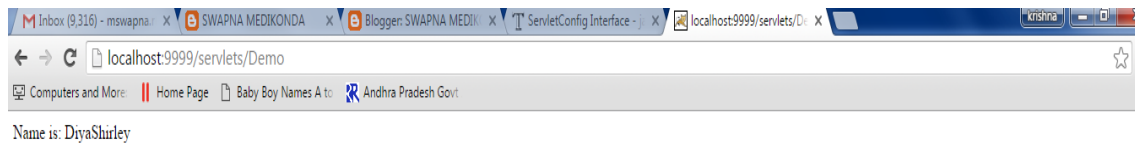
public class DemoServletInit extends HttpServlet {
public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter PW = response.getWriter();
    ServletConfig config=getServletConfig();
    String name=config.getInitParameter("name");
    pw.print("Name is: "+driver);
    pw.close();
}
```

```
}
```

Here web.xml is updated as following

```
<servlet>
  <servlet-name>D</servlet-name>
  <servlet-class>DemoServletInit</servlet-class>
  <init-param>
    <param-name>name</param-name>
    <param-value>DiyaShirley</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>D</servlet-name>
  <url-pattern>/Demo</url-pattern>
</servlet-mapping>
```

1. after executing output displays like following:



HTTP Requests

The request sends by the computer to a web server that contains all sorts of potentially interesting information is known as HTTP requests.

The HTTP client sends the request to the server in the form of request message which includes following information:

- The Request-line
- The analysis of source IP address, proxy and port
- The analysis of destination IP address, protocol, port and host
- The Requested URI (Uniform Resource Identifier)

- The Request method and Content
- The User-Agent header
- The Connection control header
- The Cache control header

The HTTP request method indicates the method to be performed on the resource identified by the **Requested URI (Uniform Resource Identifier)**. This method is case-sensitive and should be used in uppercase.

The HTTP request methods are:

| HTTP Request | Description |
|----------------|---|
| GET | Asks to get the resource at the requested URL. |
| POST | Asks the server to accept the body info attached. It is like GET request with extra info sent with the request. |
| HEAD | Asks for only the header part of whatever a GET would return. Just like GET but with no body. |
| TRACE | Asks for the loopback of the request message, for testing or troubleshooting. |
| PUT | Says to put the enclosed info (the body) at the requested URL. |
| DELETE | Says to delete the resource at the requested URL. |
| OPTIONS | Asks for a list of the HTTP methods to which the thing at the request URL can respond |

Example demonstrating Servlet Request

In this example, we will show how a parameter is passed to a Servlet in a request object from HTML page.

index.html

```
<form method="post" action="check">
Name <input type="text" name="user" >
<input type="submit" value="submit">
</form>
```

web.xml


```
<servlet>
    <servlet-name>check</servlet-name>
    <servlet-class>MyServlet</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>check</servlet-name>
    <url-pattern>/check</url-pattern>
</servlet-mapping>
```

MyServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class MyServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {

            String user=request.getParameter("user");
            out.println("<h2> Welcome "+user+"</h2>");
        } finally {
            out.close();
        }
    }
}
```

Output :



Introduction to Servlet Response

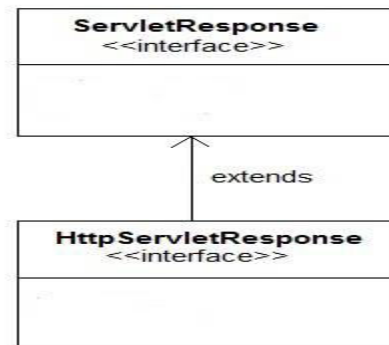
Servlet API provides two important interfaces **ServletResponse** and **HttpServletResponse** to assist in sending response to client.

Some Important Methods of ServletResponse

| Methods | Description |
|---|---|
| PrintWriter <code>getWriter()</code> | returns a PrintWriter object that can send character text to the client. |
| void <code>setBufferSize(int size)</code> | Sets the preferred buffer size for the body of the response |
| void <code>setContentLength(int len)</code> | Sets the length of the content body in the response In HTTP servlets, this method sets the HTTP Content-Length header |
| void <code>setContentType(String type)</code> | sets the content type of the response being sent to the client before sending the respond. |
| void <code>setBufferSize(int size)</code> | sets the preferred buffer size for the body of the response. |
| boolean <code>isCommitted()</code> | returns a boolean indicating if the response has been committed |
| void <code>setLocale(Locale loc)</code> | sets the locale of the response, if the response has not been committed yet. |

HttpServletResponse Interface

HttpServletResponse interface adds the methods that relates to the **HTTP** response.



Some Important Methods of HttpServletResponse

| Methods | Description |
|--|---|
| <code>void addCookie(Cookie cookie)</code> | adds the specified cookie to the response. |
| <code>void sendRedirect(String location)</code> | Sends a temporary redirect response to the client using the specified redirect location URL and clears the buffer |
| <code>int getStatus()</code> | gets the current status code of this response |
| <code>String getHeader(String name)</code> | gets the value of the response header with the given name. |
| <code>void setHeader(String name, String value)</code> | sets a response header with the given name and value |
| <code>void setStatus(int sc)</code> | sets the status code for this response |
| <code>void sendError(int sc, String msg)</code> | sends an error response to the client using the specified status and clears the buffer |

Files to be created :

- **index.html** will have form fields to get user information.
- **Validate.java** will validate the data entered by the user.
- **Welcome.java** will be the welcome page.
- **web.xml** , the deployment descriptor.

index.html

```
<form method="post" action="Validate">
Name:<input type="text" name="user" /><br/>
Password:<input type="password" name="pass" ><br/>
<input type="submit" value="submit">
</form>
```

Validate.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Validate extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            String name = request.getParameter("user");
            String password = request.getParameter("pass");

            if(password.equals("studytonight"))
            {
                RequestDispatcher rd = request.getRequestDispatcher("Welcome");
                rd.forward(request, response);
            }
            else
            {
                out.println("<font color='red'><b>You have entered incorrect password</b></font>");
                RequestDispatcher rd = request.getRequestDispatcher("index.html");
                rd.include(request, response);
            }
        }
    }
}
```

```

        }

    }finally {
        out.close();
    }

}

}

```

Welcome.java

```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class Welcome extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {

            out.println("<h2>Welcome user</h2>");
        } finally {
            out.close();
        }
    }
}

```

web.xml

```

<web-app>

    <servlet>
        <servlet-name>Validate</servlet-name>
        <servlet-class>Validate</servlet-class>
    </servlet>

    <servlet>
        <servlet-name>Welcome</servlet-name>
        <servlet-class>Welcome</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>Validate</servlet-name>

```

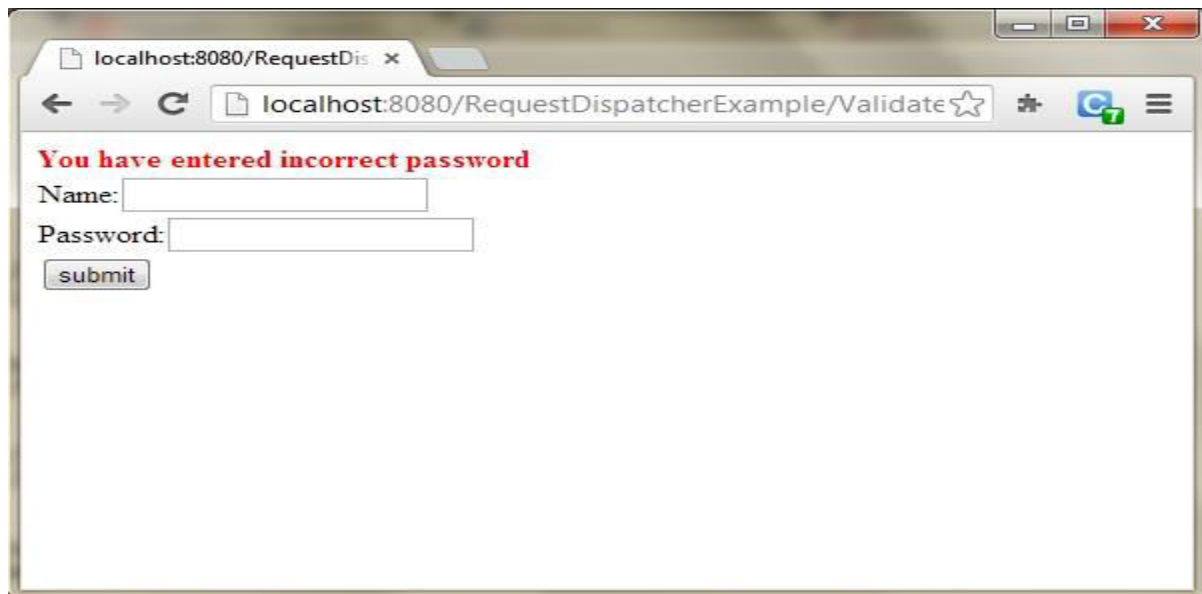
```
<url-pattern>/Validate</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>Welcome</servlet-name>
    <url-pattern>/Welcome</url-pattern>
</servlet-mapping>
<welcome-file-list>
    <welcome-file>index.html</welcome-file>
</welcome-file-list>
</web-app>
```

This will be the first screen. You can enter your Username and Password here.



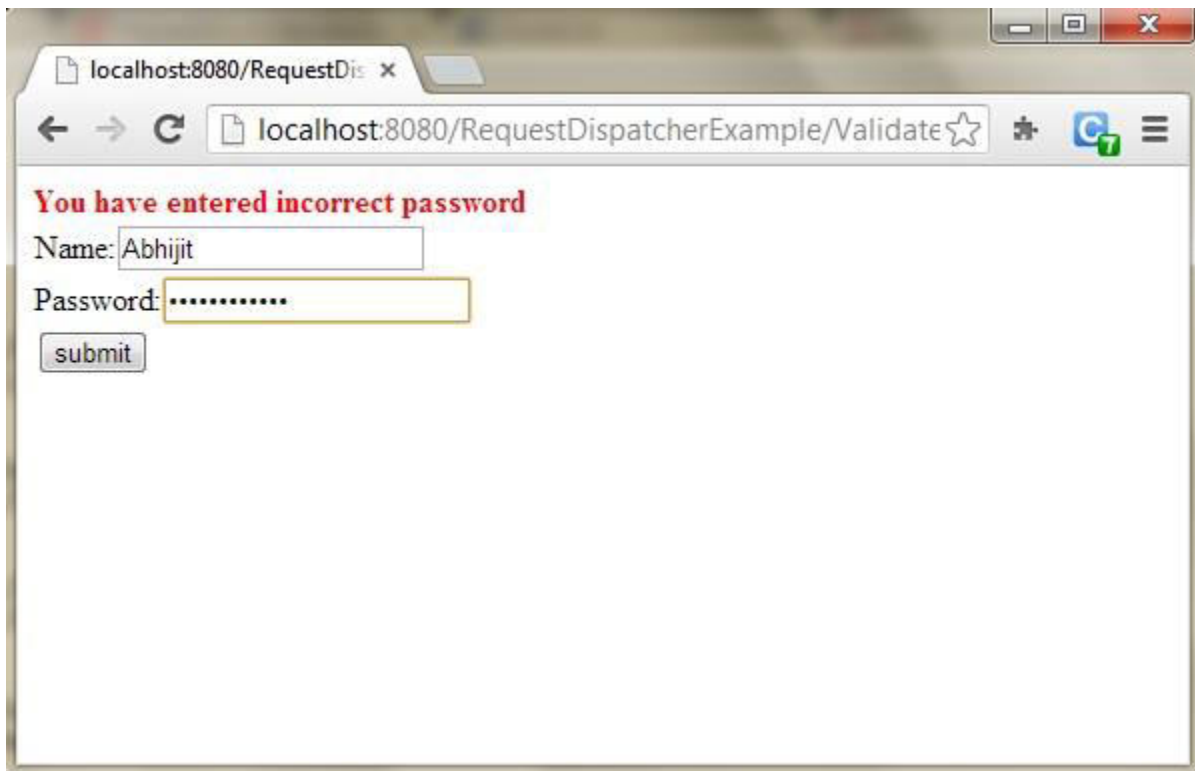
A screenshot of a web browser window. The address bar shows 'localhost:8080/RequestDispatcherExample/'. The page contains a form with two input fields: 'Name:' with the value 'Abhijit' and 'Password:' with two dots. Below the fields is a blue 'submit' button.

When you click on Submit, Password will be validated, if it is not 'studytonight', error message will be displayed.



A screenshot of a web browser window. The address bar shows 'localhost:8080/RequestDispatcherExample/Validate'. The page displays a red error message: 'You have entered incorrect password'. Below the message are the same 'Name:' and 'Password:' input fields and a 'submit' button as in the previous screenshot.

Enter any Username, but enter 'studytonight' as password.



A screenshot of a web browser window. The address bar shows the URL `localhost:8080/RequestDispatcherExample/Validate`. The page content displays a red error message: **You have entered incorrect password**. Below the message, there is a form with two input fields: "Name:" containing the text "Abhijit" and "Password:" containing a series of dots. A "submit" button is located below the password field.

Password will be successfully validated and you will be directed to the Welcome Servlet.



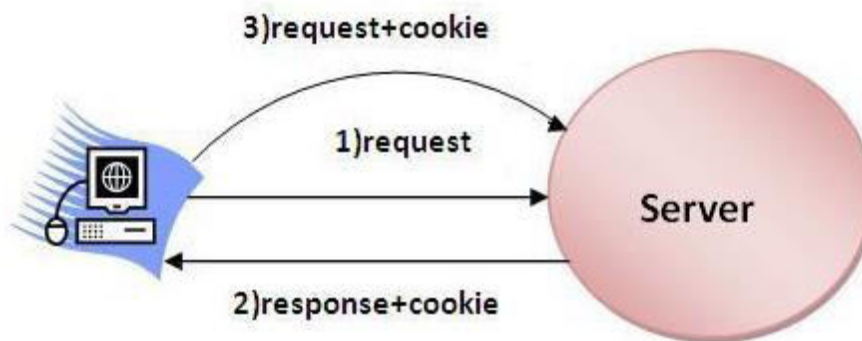
A screenshot of a web browser window. The address bar shows the URL `localhost:8080/RequestDispatcherExample/Validate`. The page content displays the text **Welcome user**.

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is **valid for single session** only. It is removed each time when user closes the browser.

Persistent cookie

It is **valid for multiple session**. It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor

Cookie()

Cookie(String name, String value)

Description

constructs a cookie.

constructs a cookie with a specified name and value.

Useful Methods of Cookie class

There are given some commonly used methods of the Cookie class. **Method**

public void setMaxAge(int expiry)

public String getName()

public String getValue()

public void setName(String name)

public void setValue(String value)

Description

Sets the maximum age of the cookie in seconds.

Returns the name of the cookie. The name cannot be changed after creation.

Returns the value of the cookie.

changes the name of the cookie.

changes the value of the cookie.

Other methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. **public void addCookie(Cookie ck);** method of HttpServletResponse interface is used to add cookie in response object.
2. **public Cookie[] getCookies();** method of HttpServletRequest interface is used to return all the cookies from the browser.

How to delete Cookie?

Let's see the simple code to delete cookie. It is mainly used to logout or signout the user.

```
Cookie ck=new Cookie("user","");//deleting value of cookie ck.setMaxAge(0);//changing the maximum age to 0 seconds response.addCookie(ck);//adding cookie in the response
```

How to get Cookies?

Let's see the simple code to get all the cookies. `Cookie ck[]=request.getCookies(); for(int i=0;i<ck.length;i++){ out.print("
" +ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie }`

Session Tracking:

Http is a *stateless* protocol, means that it can't persist the information. It always treats each request as a new request. In Http client makes a connection to the server, sends the request., gets the response, and closes the connection.

In session management client first make a request for any servlet or any page, the container receives the request and generate a unique session ID and gives it back to the client along with the response. This ID gets stores on the client machine. Thereafter when the client request again sends a request to the server then it also sends the session Id with the request. There the container sees the Id and sends back the request.

Session Tracking can be done in Four ways:

1. Hidden Form Fields:

2. Cookies

3. HttpSession

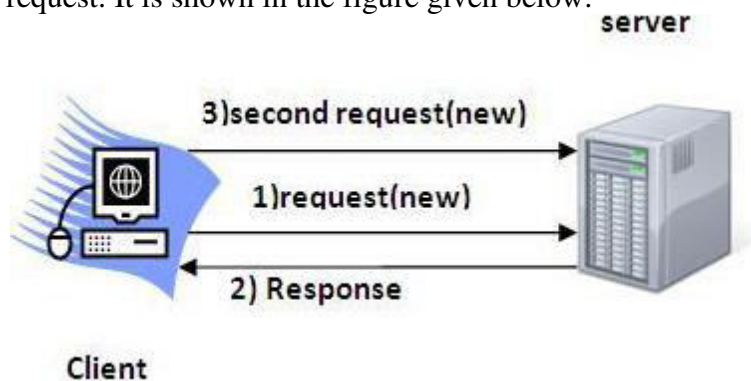
4. URL Rewriting

Session Tracking in Servlets

1. Session Tracking

2. Session Tracking Techniques

Session simply means a particular interval of time. **Session Tracking** is a way to maintain state (data) of an user. It is also known as **session management** in servlet. Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user requests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user. HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:



Why use Session Tracking?

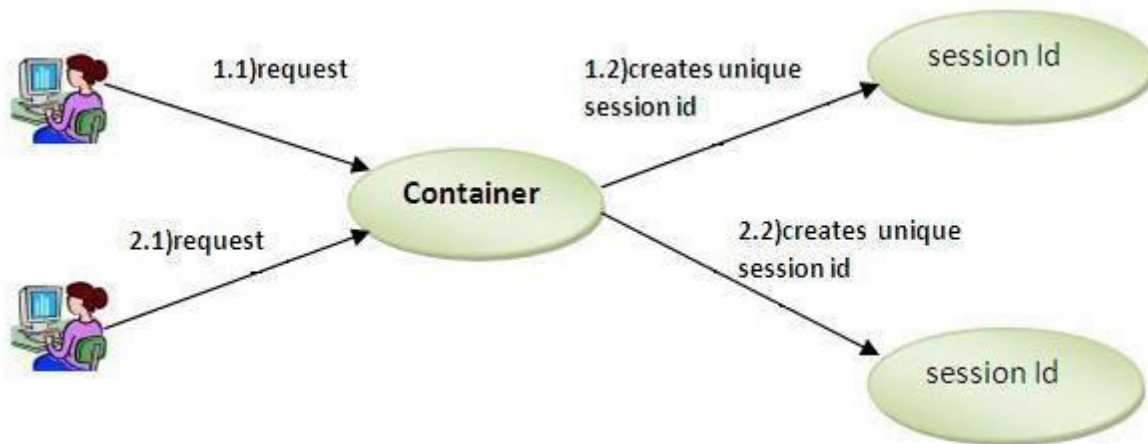
To recognize the user It is used to recognize the particular user.

HttpSession interface

1. HttpSession interface
2. How to get the HttpSession object
3. Commonly used methods of HttpSession interface
4. Example of using HttpSession

In such case, container creates a session id for each user. The container uses this id to identify the particular user. An object of HttpSession can be used to perform two tasks:

1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



How to get the HttpSession object ?

The `HttpServletRequest` interface provides two methods to get the object of `HttpSession`:

1. **`public HttpSession getSession()`**:Returns the current session associated with this request, or if the request does not have a session, creates one.
2. **`public HttpSession getSession(boolean create)`**:Returns the current `HttpSession` associated with this request or, if there is no current session and `create` is true, returns a new session

Commonly used methods of HttpSession interface

1. **`public String getId()`**:Returns a string containing the unique identifier value.
2. **`public long getCreationTime()`**:Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
3. **`public long getLastAccessedTime()`**:Returns the last time the client sent a request associated with this session, as the number of milliseconds since midnight January 1, 1970 GMT.
4. **`public void invalidate()`**:Invalidates this session then unbinds any objects bound to it.

Example of using `HttpSession`

In this example, we are setting the attribute in the session scope in one servlet and getting that value from the session scope in another servlet. To set the attribute in the session scope, we have used the `setAttribute()` method of `HttpSession` interface and to get the attribute, we have used the `getAttribute` method.

Introduction of Java Database Connectivity

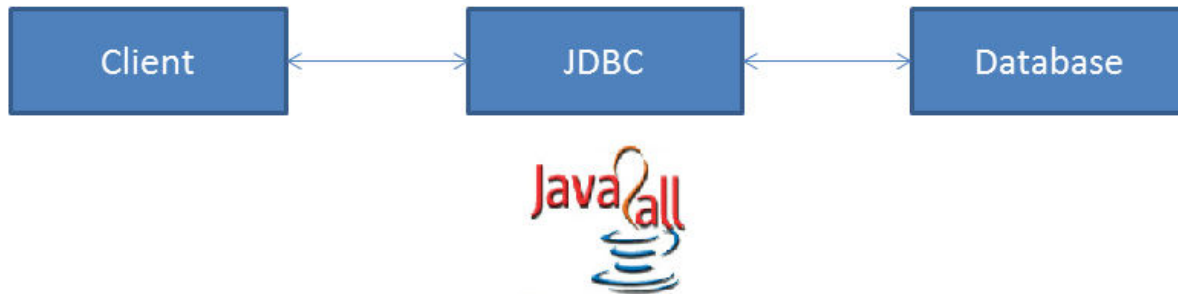
JDBC - Java Database Connectivity.

JDBC provides API or Protocol to interact with different databases.

With the help of JDBC driver we can connect with different types of databases.

Driver is must needed for connection establishment with any database.

A driver works as an interface between the client and a database server



JDBC have so many classes and interfaces that allow a java application to send request made by user to any specific DBMS(Data Base Management System).

JDBC supports a wide level of portability.

JDBC provides interfaces that are compatible with java application.

components and specification of JDBC:

Components of JDBC:

JDBC has four main components as under and with the help of these components java application can connect with database.

The JDBC API - it provides various methods and interfaces for easy communication with database.

The JDBC DriverManager - it loads database specific drivers in an application to establish connection with database.

The JDBC test suite - it will be used to test an operation being performed by JDBC drivers.

The JDBC-ODBC bridge - it connects database drivers to the database.

JDBC Specification:

Different version of JDBC has different specification as under.

JDBC 1.0 - it provides basic functionality of JDBC

JDBC 3.0 - it provides classes and interfaces in two packages(java.sql and javax.sql).

JDBC 4.0 - it provides so many extra features like

Auto loading of the driver interface.

Connection management

ROWID data type support.

Enhanced support for large object like BLOB(Binary Large Object) and CLOB(Character Large Object).

What Does JDBC Do?

Simply put, JDBC makes it possible to do three things:

1. establish a connection with a database
2. send SQL statements
3. process the results

JavaSoft provides three JDBC product components as part of the Java Developer's Kit (JDK):

1. . the JDBC driver manager,
2. . the JDBC driver test suite, and
3. . the JDBC-ODBC bridge.

The JDBC driver manager is the backbone of the JDB architecture. It actually is quite small and simple; its primary function is to connect Java applications to the correct JDBC driver and then get out of the way

JDBC Architecture:

As we all know now that driver is required to communicate with database.

JDBC API provides classes and interfaces to handle request made by user and response made by database.

Some of the important JDBC API are as under.

DriverManager

Driver

Connection

Statement

PreparedStatement

CallableStatement

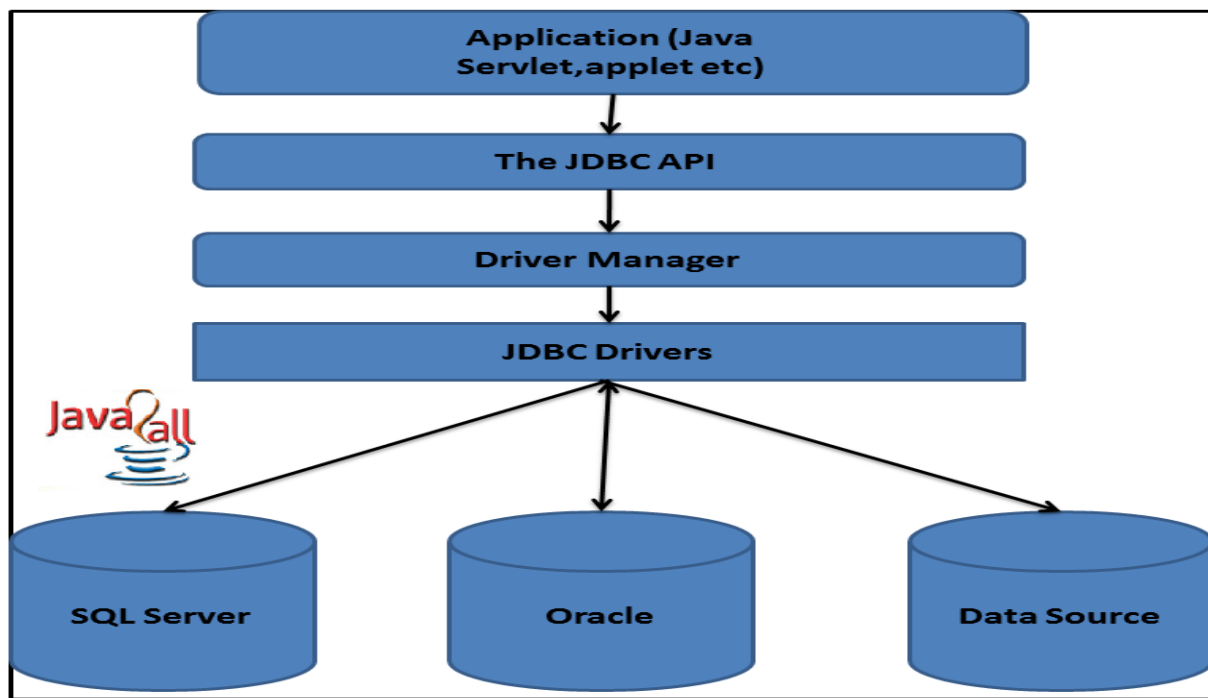
ResultSet

DatabaseMetaData

ResultSetMetaData

Here The DriverManager plays an important role in JDBC architecture.

It uses some database specific drivers to communicate our J2EE application to database.



As per the diagram first of all we have to program our application with JDBC API.
With the help of DriverManager class than we connect to a specific database with the help of specific database driver.

Java drivers require some library to communicate with the database

We have four different types of java drivers.

We will learn all that four drivers with architecture in next chapter.

Some drivers are pure java drivers and some are partial.

So with this kind of JDBC architecture we can communicate with specific database.

JDBC Drivers:

JDBC Driver Types:

There are four categories of drivers by which developer can apply a connection between Client (The JAVA application or an applet) to a DBMS.

(1) Type 1 Driver : JDBC-ODBC Bridge.

(2) Type 2 Driver : Native-API Driver (Partly Java driver).

(3) Type 3 Driver : Network-Protocol Driver (Pure Java driver for database Middleware).

(4) Type 4 Driver : Native-Protocol Driver (Pure Java driver directly connected to database).

(1) Type 1 Driver: JDBC-ODBC Bridge :-

The JDBC type 1 driver which is also known as a JDBC-ODBC Bridge is a convert JDBC methods into ODBC function calls.

Sun provides JDBC-ODBC Bridge driver by “sun.jdbc.odbc.JdbcOdbcDriver”.

The driver is a platform dependent because it uses ODBC which is depends on native libraries of the operating system and also the driver needs other installation for example, ODBC must be installed on the computer and the database must support ODBC driver.

Type 1 is the simplest compare to all other driver but it's a platform specific i.e. only on Microsoft platform.

For type-1 we create the DSN name.

Steps for DSN:

1. Start
2. Control panel
3. Administrator tools
4. Data source odbc
5. System dsn
6. Add
7. Microsoft oracle for odbc
8. Finish
9. ora

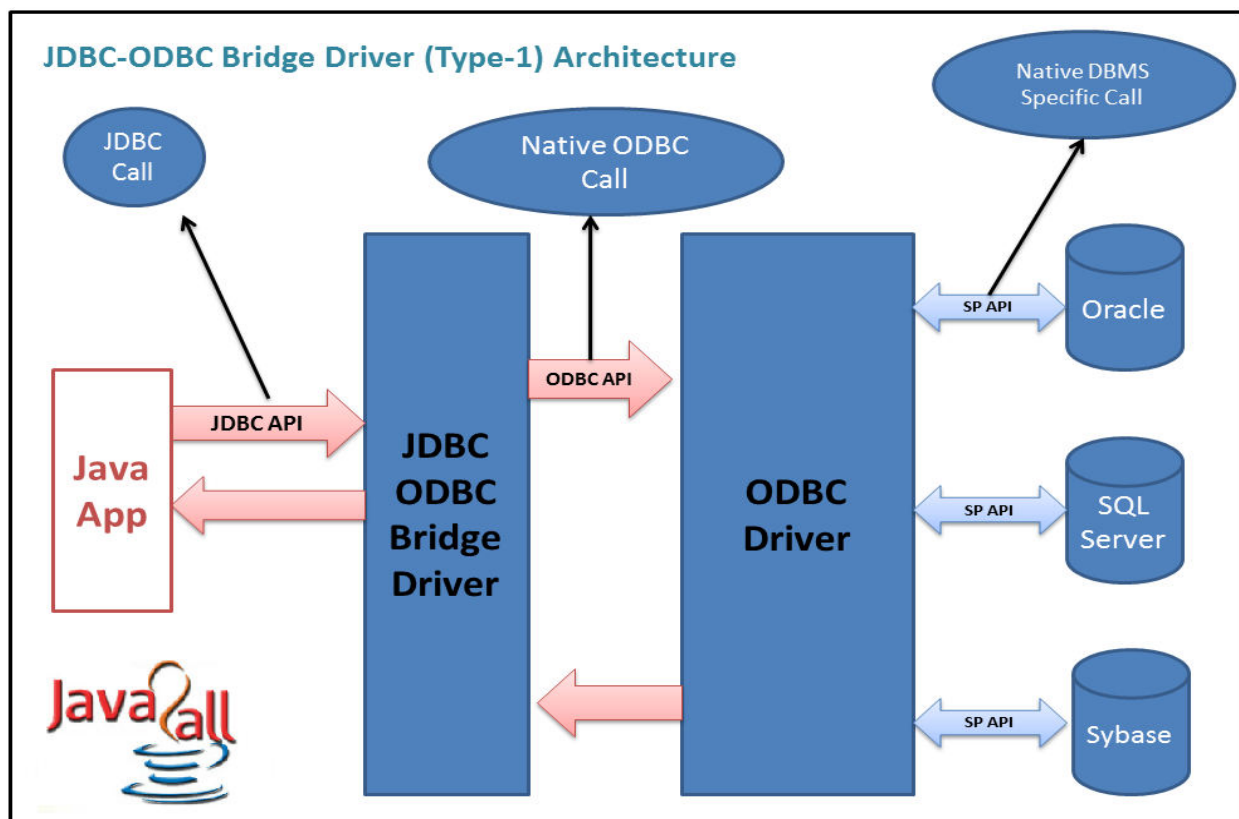
The JDBC-ODBC Bridge is use only when there is no PURE-JAVA driver available for a particular database.

The driver is a platform dependent because it uses ODBC which is depends on native libraries of the operating system and also the driver needs other installation for example, ODBC must be installed on the computer and the database must support ODBC driver.

Java DB is Oracle's supported distribution of the open source Apache Derby database. Its ease of use, standards compliance, full feature set, and small footprint make it the ideal database for Java developers. Java DB is written in the Java programming language, providing "write once, run anywhere" portability. It can be embedded in Java applications, requiring zero administration by the developer or user. It can also be used in client server mode. Java DB is fully transactional and provides a standard SQL interface.

The JDBC driver manager is the backbone of the JDB architecture. It actually is quite small and simple; its primary function is to connect Java applications to the correct JDBC driver and then get out of the way

Architecture Diagram :



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 1 Driver → ODBC Driver → Database library APIs → Database

Advantage:

- (1) Connect to almost any database on any system, for which ODBC driver is installed.
- (2) It's an easy for installation as well as easy(simplest) to use as compare the all other driver.

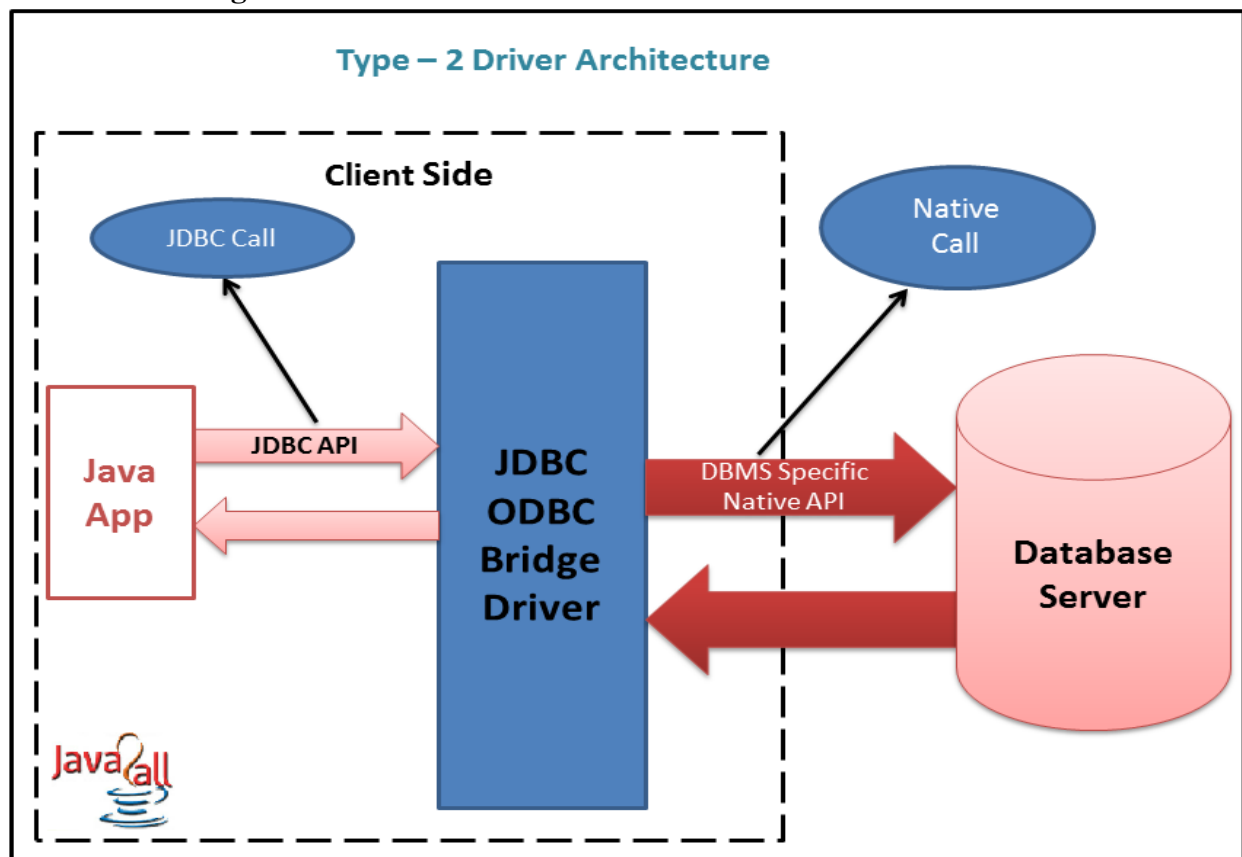
Disadvantage:

- (1) The ODBC Driver needs to be installed on the client machine.
- (2) It's a not a purely platform independent because its use ODBC which is depends on native libraries of the operating system on client machine.
- (3) Not suitable for applets because the ODBC driver needs to be installed on the client machine.

(2) Type 2 Driver: Native-API Driver (Partly Java driver) :-

The JDBC type 2 driver uses the libraries of the database which is available at client side and this driver converts the JDBC method calls into native calls of the database so this driver is also known as a Native-API driver.

Architecture Diagram :



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 2 Driver → Vendor Client Database library APIs → Database

Advantage:

(1) There is no implantation of JDBC-ODBC Bridge so it's faster than a type 1 driver; hence the performance is better as compare the type 1 driver (JDBC-ODBC Bridge).

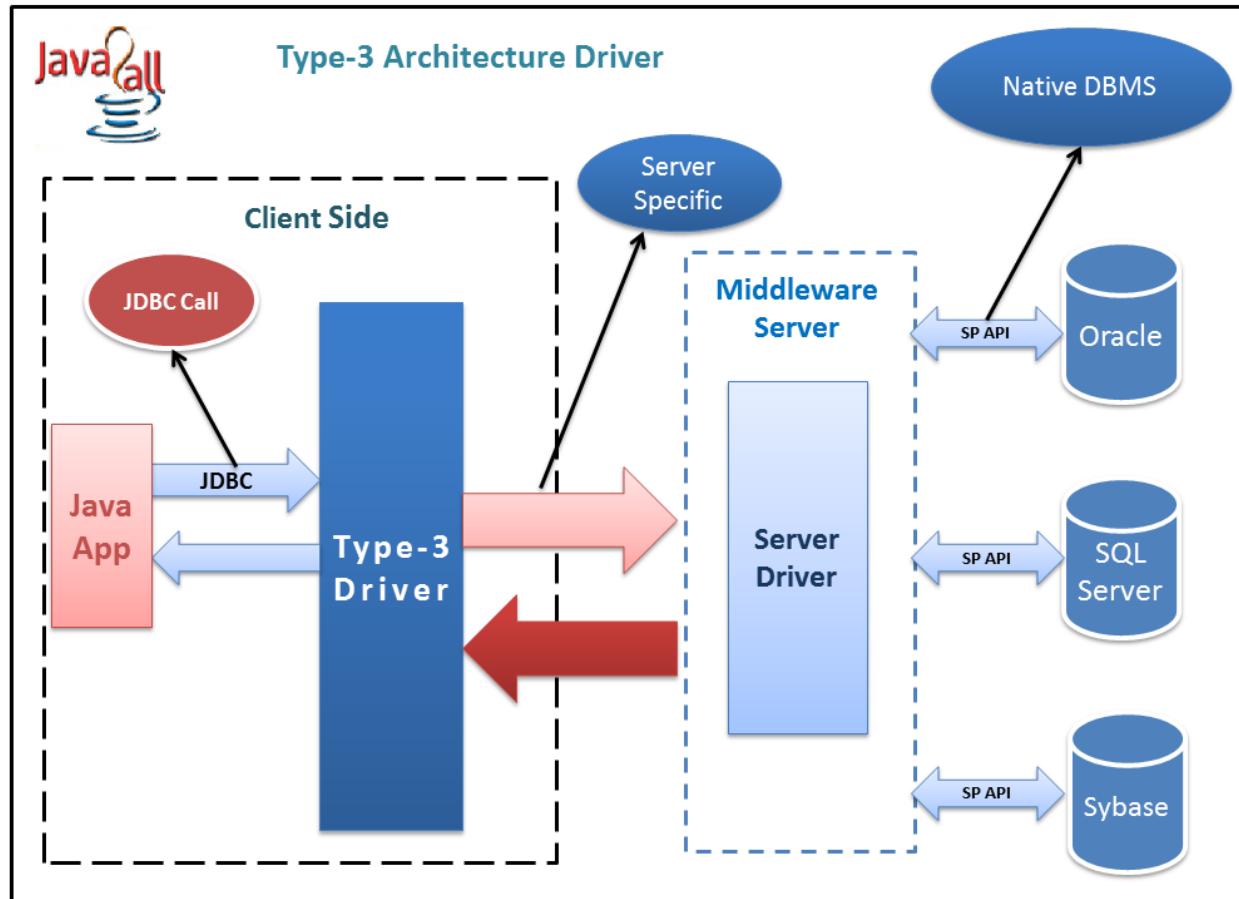
Disadvantage:

- (1) On the client machine require the extra installation because this driver uses the vendor client libraries.
- (2) The Client side software needed so cannot use such type of driver in the web-based application.
- (3) Not all databases have the client side library.
- (4) This driver supports all JAVA applications except applets.

(3) Type 3 Driver: Network-Protocol Driver (Pure Java driver for database Middleware) :-

The JDBC type 3 driver uses the middle tier(application server) between the calling program and the database and this middle tier converts JDBC method calls into the vendor specific database protocol and the same driver can be used for multiple databases also so it's also known as a Network-Protocol driver as well as a JAVA driver for database middleware.

Architecture Diagram:



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 3 Driver → Middleware (Server) → any Database

Advantage:

- (1) There is no need for the vendor database library on the client machine because the middleware is database independent and it communicates with client.
- (2) Type 3 driver can be used in any web application as well as on internet also because there is no any software require at client side.
- (3) A single driver can handle any database at client side so there is no need a separate driver for each database.
- (4) The middleware server can also provide the typical services such as connections, auditing, load balancing, logging etc.

Disadvantage:

- (1) An Extra layer added, may be time consuming.

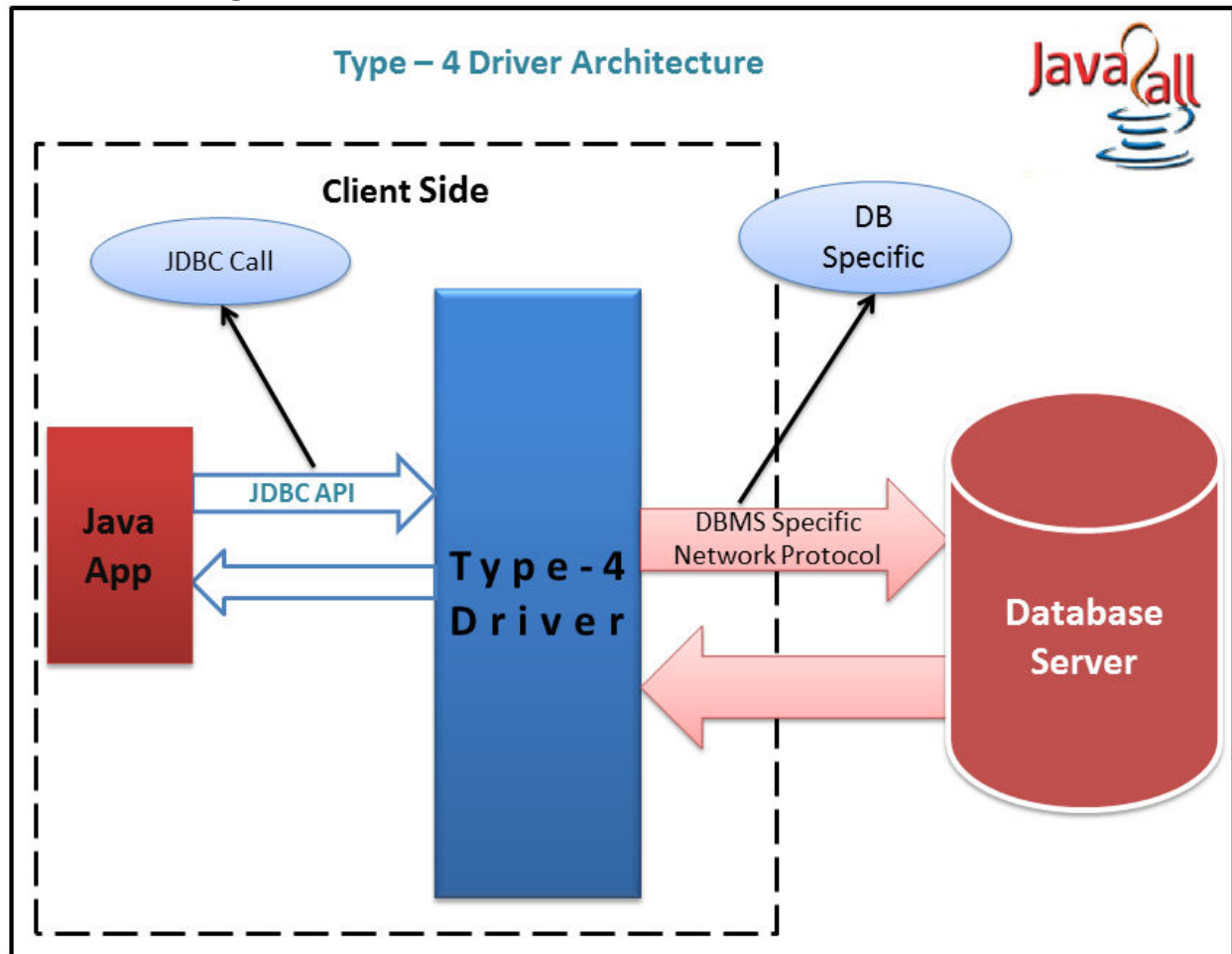
(2) At the middleware develop the database specific coding, may be increase complexity.

(4) Type 4 Driver: Native-Protocol Driver (Pure Java driver directly connected to database) :-

The JDBC type 4 driver converts JDBC method calls directly into the vendor specific database protocol and in between do not need to be converted any other formatted system so this is the fastest way to communicate quires to DBMS and it is completely written in JAVA because of that this is also known as the “direct to database Pure JAVA driver”.

If we are using type-4 driver in oracle then we need to add jar file to the class path because it was given by third party.

Architecture Diagram:



Process:

Java Application → JDBC APIs → JDBC Driver Manager → Type 4 Driver (Pure JAVA Driver) → Database Server

Advantage:

- (1) It's a 100% pure JAVA Driver so it's a platform independence.
- (2) No translation or middleware layers are used so consider as a faster than other drivers.
- (3) The all process of the application-to-database connection can manage by JVM so the debugging is also managed easily.

Disadvantage:

(1) There is a separate driver needed for each database at the client side.

(2) Drivers are Database dependent, as different database vendors use different network protocols.

JDBC APIs:

If any java application or an applet wants to connect with a database then there are various classes and interfaces available in java.sql package.

Depending on the requirements these classes and interfaces can be used.

Some of them are listed out below **Description**

which are used to perform the various tasks with database as well as for connection. **Class or Interface**

| | |
|----------------------------|---|
| Java.sql.Connection | Create a connection with specific database |
| Java.sql.DriverManager | The task of DriverManager is to manage the database driver |
| Java.sql.Statement | It executes SQL statements for particular connection and retrieve the results |
| Java.sql.PreparedStatement | It allows the programmer to create prepared SQL statements |
| Java.sql.CallableStatement | It executes stored procedures |
| Java.sql.ResultSet | This interface provides methods to get result row by row generated by SELECT statements |

INTRODUCTION TO JSP

Enrichment in server side programming is now a need of web application. We prefer component based, multithreaded client server application. A lot of server side technologies such as JSP, servlets, ASP, PHP are used.

Java Server Pages is a kind of scripting language in which we can embed Java code along with html elements.

Advantages of JSP:

- ☐ Jsp is useful for server side programming.
- ☐ Jsp can be used along with servlets. Hence business logic for any application can be developed using Jsp.
- ☐ Dynamic contents can be handled using Jsp because jsp allows scripting and element based programming.
- ☐ Jsp allows creating and using our own custom tag libraries. Hence any application specific requirements can be satisfied using custom tag libraries.
- ☐ Jsp is a specification and not a product. Hence any variety of applications can be developed.
- ☐ Jsp is essential component of J2ee. Hence using Jsp is possible to develop simple as well as complex applications.

Problem with Servlet:

In only one class, the servlet alone has to do various tasks such as:

- ☐ Acceptance of request
- ☐ Processing of request
- ☐ Handling of business logic
- ☐ Generation of response

Hence there are some problems that are associated with servlets:

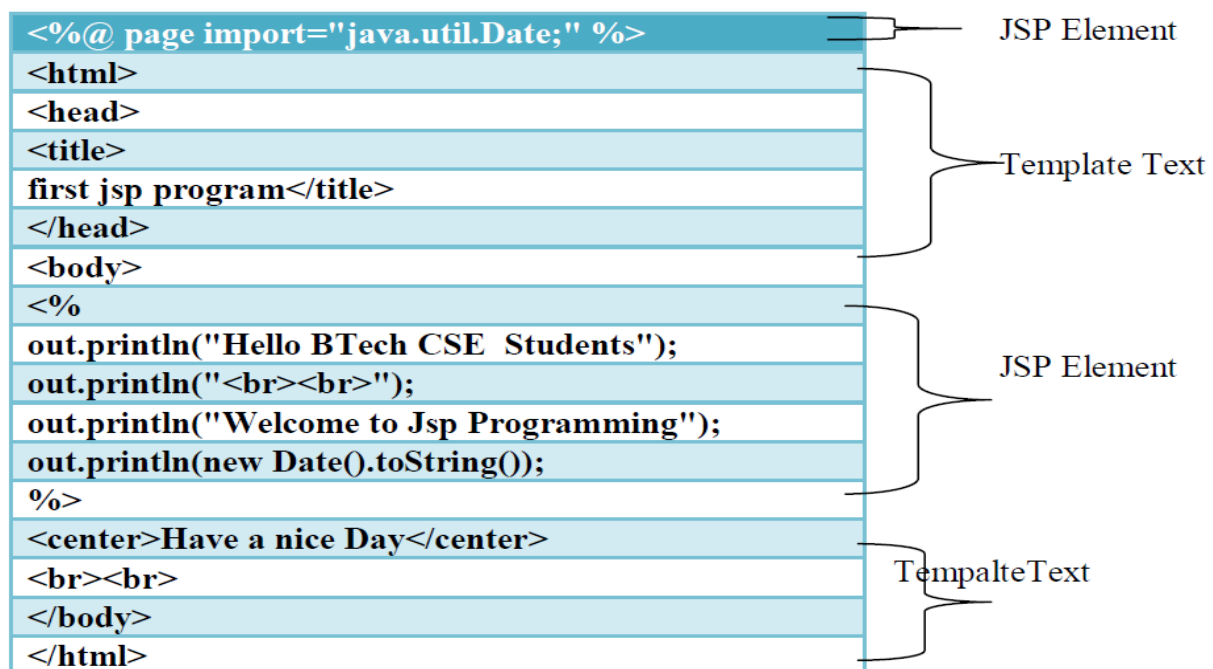
- ☐ For developing a servlet application, we need knowledge of Java as well as html code.
- ☐ While developing any web based application, look and feel of web based application needs to be changed then entire code needs to be changed and recompiled.
- ☐ There are some web page development tools available using which the developer can develop web based applications. But servlets don not support such tools. Even if such tools are used, we need to change embedded html code manually, which is time consuming and error prone

These problems associated with servlets are due to one and only one reason that is servlet has to handle all tasks of request processing. JSP is a technology that came up to overcome these problems. Jsp is a technology in which request processing, business logic and presentations are separated out.

ANATOMY OF JSP PAGE:

- ☐ JSP page is a simple web page which contains the **JSP elements** and **template text**.
- ☐ The template text can be scripting code such as Html, Xml or a simple plain text.
- ☐ Various Jsp elements can be action tags, custom tags, JSTL library elements. These JSP elements are responsible for generating dynamic contents.

Anatomy of JSP



When JSP request gets processed template text and JSP elements are merged together and sent to the browser as response.

JSP Processing:

JSP pages can be processed using JSP Container only. Following are the steps that need to be followed while processing the request for JSP page:

- ❑ Client makes a request for required JSP page to server. The server must have JSP container so that JSP request can be processed. For instance: let the client makes request for hello.jsp page.
- ❑ On receiving this request the JSP container searches and then reads the desired JSP page. Then this JSP page is converted to corresponding servlet.
- ❑ Basically any JSP page is a combination of template text and JSP elements.

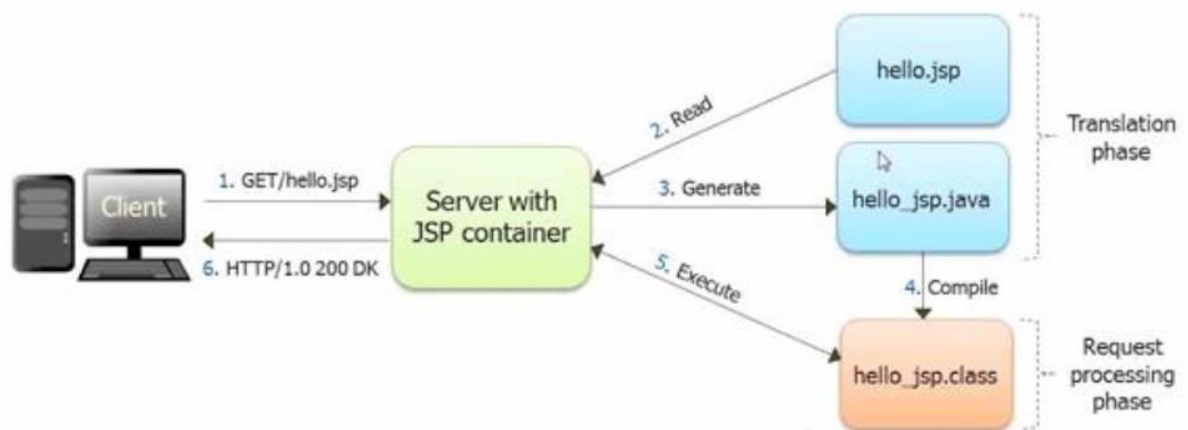
Every template text is converted to corresponding println statement.

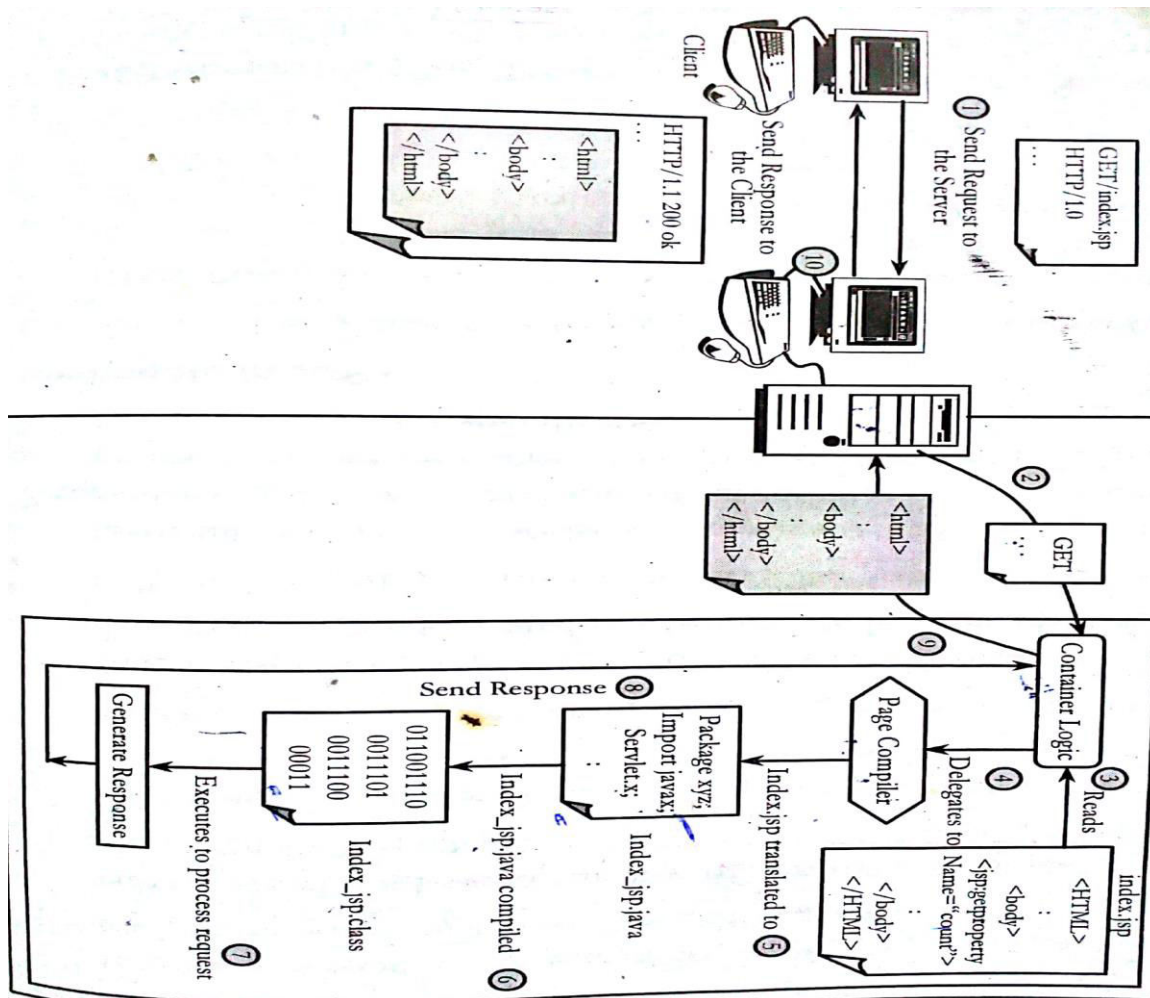
Every JSP element is converted into corresponding Java code.

This phase is called Translation phase, output of it is a servlet. for instance: hello.jsp is converted to hello_jsp.java

This servlet is compiled to generate servlet class file. Using this class response is generated. This phase is called request processing phase.

- ❑ The Jsp container thus executes servlet class file.
- ❑ A requested page is then returned to client as response.





JSP Declaration Tag

The JSP declaration tag is used *to declare fields and methods*.

The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet.

The JSP declaration element defines page-scope variables to store information or defines supporting methods that the rest of a JSP page may need. Declaration elements do not produce outputs

So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

1. `< %! field or method declaration % >`

Difference between JSP Scriptlet tag and Declaration tag

| Jsp Scriptlet Tag | Jsp Declaration Tag |
|--|---|
| The jsp scriptlet tag can only declare variables not methods. | The jsp declaration tag can declare variables as well as methods. |
| The declaration of scriptlet tag is placed inside the <code>_jspService()</code> method. | The declaration of jsp declaration tag is placed outside the <code>_jspService()</code> method. |

Example of JSP declaration tag that declares field

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
<html>
<body>
< %! int data = 50 ; % >
< %= "Value of the variable is:"+data % >
</body>
</html>
```

Example of JSP declaration tag that declares method

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

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

JSP directives

The jsp directives are messages that tells the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

- page directive
- include directive
- taglib directive

Syntax of JSP Directive

```
<%@ directive attribute="value" %>
```

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

```
<%@ page attribute= "value" %>
```

Attributes of JSP page directive

- import
- contentType
- extends
- info
- buffer
- language
- isThreadSafe
- autoFlush
- session
- pageEncoding
- errorPage
- isErrorPage

1)import

The import attribute is used to import class,interface or all the members of a package.It is similar to import

keyword in java class or interface.

Example of import attribute

```
<html>  
<body>  
<%@ page import = "java.util.Date" %>  
Today is: <%= new Date() %>  
</body>  
</html>
```

2)contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.The

default value is "text/html;charset=ISO-8859-1".

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4)errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected

to the error page.

Example of errorPage attribute

```
//index.jsp  
<html>  
<body>  
<%@ page errorPage= "myerrorpage.jsp" %>
```

```
<%= 100 / 0 %>
```

```
</body>
```

```
</html>
```

5)isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

Note: The exception object can only be used in the error page.

Example of isErrorPage attribute

```
//myerrorpage.jsp
```

```
<html>
```

```
<body>
```

```
<%@ page isErrorPage= "true" %>
```

```
Sorry an exception occurred!<br/>
```

```
The exception is: <%= exception %>
```

```
</body>
```

```
</html>
```

There are three types of directive tag –

| S.No. | Directive & Description |
|-------|---|
| 1 | Defines page-dependent attributes, such as scripting language, error page, and buffering requirements. <%@ page ... %> |
| 2 | Includes a file during the translation phase. <%@ include ... %> |
| 3 | Declares a tag library, containing custom actions, used in the page <%@ taglib ... %> |

Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The

include directive includes the original content of the included resource at page translation time (the jsp page is

translated only once so it will be better to include static resource).

Advantage of Include directive

Code Reusability

Syntax of include directive

```
<%@ include file= "resourceName" %>
```

Example of include directive

In this example, we are including the content of the header.html file. To run this example you must create an

header.html file.

```

<html>
<body>
<%@ include file= "header.html" %>
Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>

```

JSP Expressions

JSP expression tag

The code placed within **JSP expression tag** is *written to the output stream of the response*. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

Syntax of JSP expression tag

1. <%= statement %>

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

1. <html>
2. <body>
3. <%= "welcome to jsp" %>
4. </body>
5. </html>

JSP expressions are very powerful. They are evaluated by JSP container and values are printed on the web page. They allow display of dynamic contents on the web page. The expression to be evaluated is placed between <%= and %>. The equivalent JSP syntax for JSP expression is

```
<jsp:expression> jsp expression here </jsp:expression>
```

6. The example below first declares an integer i using JSP declarations and sets it to zero. It then uses JSP expression to print i, the square of i and 2i. After this i is incremented by 1. The same statements are then repeated two more times.

```

<%! int i %> <table>
<tr>
<th>number</th>
<th>square</th>
<th>power of two</th>
</tr>
<tr>
<td><%=i%></td>
<td><%=i*i%></td>
<td><%= java.lang.Math.pow(2,i++)%></td>
</tr>
<tr>

```

```

<td><%=i%></td>
<td><%=i*i%></td>
<td><%= java.lang.Math.pow(2,i++)%></td>
</tr>
<tr>
<td><%=i%></td>
<td><%=i*i%></td>
<td><%= java.lang.Math.pow(2,i++)%></td>
</tr>
</table>

```

7. The expression `<%=i%>` prints the value of i. The expression `<%=i*i%>` prints the square of i. The expression `<%= java.lang.Math.pow(2,i++)%>` prints 2i, and then increments i.
- 8.
9. The following is displayed on the web page as output from above page –

| number | square | power of two |
|--------|--------|--------------|
| 0 | 0 | 1.0 |
| 1 | 1 | 2.0 |
| 2 | 4 | 4.0 |

JSP code snippets

A JSP code snippet is a code sample that shows you how to add WebSphere Commerce functionality to your store. JSP code snippets may be added to a starter store, or to a store previously published and migrated. JSP code snippets are intended to help you: Quickly add a feature to your store, or add a feature that is not included in one of the starter stores. JSP code snippets use the JSP Standard Tag Library (JSTL). Each JSP code snippet is well commented, easy to read, easy to understand, and easy to customize.

- JSP code snippet: Display customization terms and conditions
The CustomizationTCDisplay.jsp file displays the customization information according to the display customization terms and conditions for the a user's current session logon ID, store ID, and the selected language ID.
- JSP code snippet for e-Marketing Spots (WebSphere Commerce Accelerator)
This eMarketingSpotDisplay.jsp is built as a sample snippet to display an e-Marketing Spot in a store page. This e-Marketing Spot code supports all types of Web activities.
- JSP code snippet for e-Marketing Spots (Management Center)
Use the WebServiceeMarketingSpotDisplay.jsp sample snippet to display an e-Marketing Spot in a store page. The code uses Web services to call the marketing runtime to get the data to display in the e-Marketing Spot. Use this snippet for e-Marketing Spots that are used in Web activities managed with the Management Center.
- JSP code snippet: Store catalog display
This JSP code snippet displays all available catalogs (master catalog and sales catalogs) associated with a store. It uses the StoreDataBean to retrieve all the CatalogDataBeans that contain the catalog information.
- JSP code snippet: Promotions display

This JSP code snippet displays all of the available discounts associated with a particular category or catalog entry. It uses the CalculationCodeListDataBean to retrieve all of the CalculationCodeDataBeans in order to get the required information.

- JSP code snippet: Promotion code form
This JSP code snippet displays the following information: the order list and the discounts applied, the promotion code form that allows customers to enter a promotion code, the list of promotion codes entered (for each code, a 'Remove' button is provided to allow customers to remove the code).
- Sample JSP code: e-mail template
Use this sample JSP code snippet as a starting point to create custom e-mail templates for marketing e-mail activities. Creating a template using this JSP code snippet is an alternative to having business users create a template using the WebSphere Commerce Accelerator or Management Center user interface.
- JSP code snippet: Catalog attachments display
This JSP code snippet displays product attachments on the pages in which you include this snippet.
- JSP code snippet: SecureThreeDACSTForm.jsp
This JSP code snippet displays the password input form a cardholder uses to enter his password in this page. After entering the password, the cardholder clicks 'Submit' to verify the Payer Authentication.
- JSP code snippet: SecureThreeDDisplay.jsp
This JSP code snippet displays waiting page when the server is processing a 3-D Secure request. It displays a password request page and asks the shopper to enter the password of the credit card if the credit card is enrolled for 3-D Secure service.
- JSP code snippet: SecureThreeDCTFError.jsp
This JSP code snippet is used to send null content back to the Visa Certificate Test Facility (CTF). (This file will only be used when doing CTF test with Visa.)
- JSP code snippet: SecureThreeDCacheDisplay.jsp
This JSP code snippet displays the SecureThreeDCacheCmd refresh link (such as 'stop', 'start', 'update', 'reset' button). Note that the function of caching of card range is not currently supported.
- JSP code snippet: SecureThreeDError.jsp
This JSP code snippet displays detailed error information for errors occurring during 3-D Secure service.
- Sample: EdpJspCodeSnippetText_en_US.properties file
The EdpJspCodeSnippetText_en_US.properties file provides translatable text for JSP snippets that display various payment methods.

JSP IMPLICIT OBJECTS

What is JSP Implicit object?

- JSP implicit objects are created during the translation phase of JSP to the servlet.
- These objects can be directly used in scriptlets that goes in the service method.
- They are created by the container automatically, and they can be accessed using objects.

There are 9 types of implicit objects available in the container:

These objects are created by JSP Engine during translation phase (while translating JSP to Servlet). They are being created inside service method so we can directly use them within **Scriptlet** without initializing and declaring them. There are total 9 implicit objects available in JSP.

Implicit Objects and their corresponding classes:

| | |
|-------------|--|
| out | javax.servlet.jsp.JspWriter |
| request | javax.servlet.http.HttpServletRequest |
| response | javax.servlet.http.HttpServletResponse |
| session | javax.servlet.http.HttpSession |
| application | javax.servlet.ServletContext |
| exception | javax.servlet.jsp.JspException |
| page | java.lang.Object |
| pageContext | javax.servlet.jsp.PageContext |
| config | javax.servlet.ServletConfig |

1. **Out:** This is used for writing content to the client (browser). It has several methods which can be used for properly formatting output message to the browser and for dealing with the buffer.
Read full article here » [OUT implicit object with examples.](#)
2. **Request:** The main purpose of request implicit object is to get the data on a JSP page which has been entered by user on the previous JSP page. While dealing with login and signup forms in JSP we often prompts user to fill in those details, this object is then used to get those entered details on an another JSP page (action page) for validation and other purposes.
Read full article here » [Request implicit object with examples.](#)
3. **Response:** It is basically used for modifying or dealing with the response which is being sent to the client(browser) after processing the request.
Read full article here » [Response implicit object with examples.](#)
4. **Session:** It is 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.
Read full article here » [Session implicit object with examples.](#)
5. **Application:** This is used for getting application-wide initialization parameters and to maintain useful data across whole JSP application.
Read full article here » [Application implicit object with examples.](#)
6. **Exception:** Exception implicit object is used in exception handling for displaying the error messages. This object is only available to the JSP pages, which has isErrorPage set to true.
Read full article here » [Exception implicit object with examples.](#)
7. **Page:** Page implicit object is a reference to the current Servlet instance (Converted Servlet, generated during translation phase from a JSP page). We can simply use **this** in

place of it. I'm not covering it in detail as it is rarely used and not a useful implicit object while building a JSP application.

8. **pageContext:** It is used for accessing page, request, application and session attributes. Read full article here » [pageContext implicit object with examples](#).
9. **Config:** This is a Servlet configuration object and mainly used for accessing getting configuration information such as servlet context, servlet name, configuration parameters etc.

Use Bean in JSP Page

With this example we are going to demonstrate how to use a Bean in a JSP page. JavaServer Pages (JSP) is a server-side programming technology that enables the creation of dynamic, platform-independent method for building Web-based applications. JSP have access to the entire family of Java APIs, including the JDBC API to access enterprise databases. In short, to use a Bean in a JSP page you should:

- Create a Java Bean. The Java Bean is a specially constructed Java class that provides a default, no-argument constructor, implements the Serializable interface and it has getter and setter methods for its properties.
- Create a jsp page, using the `<%code fragment%>` scriptlet. It can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- Use the useBean action to declare the JavaBean for use in the JSP page. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP.
- Use the getProperty action to access get methods and setProperty action to access set methods of the bean.

Let's take a look at the code snippets of a sample Bean and a JSP page that uses it, below:

SampleBean.java

```
01 package com.javacodegeeks.snippets.enterprise;
02
03 import java.util.Date;
04
05 public class SampleBean {
06
07     private String param1;
08     private Date param2 = new Date();
09
10     public String getParam1() {
11         return param1;
12     }
13     public void setParam1(String param1) {
14         this.param1 = param1;
15     }
```

```
16
17 public Date getParam2() {
18     return param2;
19 }
20 public void setParam2(Date param2) {
21     this.param2 = param2;
22 }
23
24 @Override
25 public String toString() {
26     return "SampleBean [param1=" + param1 + ", param2=" + param2 + "]";
27 }
28
29 }
```

UseBean.jsp

```
01 <%@ page language="java" contentType="text/html; charset=UTF-8" %>
02 <%@ page import="com.javacodegeeks.snippets.enterprise.SampleBean"%>
03
04 <html>
05
06 <head>
07     <title>Java Code Geeks Snippets - Use a Bean in JSP Page</title>
08 </head>
09
10 <body>
11
12     <jsp:useBean id="sampleBean"
13 class="com.javacodegeeks.snippets.enterprise.SampleBean" scope="session">
14         <%-- initialize bean properties --%>
15         <jsp:setProperty name="sampleBean" property="param1" value="value1" />
16     </jsp:useBean>
17
18     Sample Bean: <%= sampleBean %>
19
20     param1: <jsp:getProperty name="sampleBean" property="param1" />
21     param2: <jsp:getProperty name="sampleBean" property="param2" />
22
```


22 </body>

URL:

http://myhost:8080/jcgsnippets/UseBean.jsp

Output:

Sample Bean: SampleBean [param1=value1, param2=Thu Nov 17 21:28:03 EET 2017]
param1: value1 param2: Thu Nov 17 21:28:03 EET 2011

jsp:useBean action tag

The jsp:useBean action tag is used to locate or instantiate a bean class

If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of

bean is not created, it instantiates the bean.

```
<jsp:useBean id= "instanceName" scope= "page | request | session | application"  
class= "packageName.className" type= "packageName.className"  
beanName="packageName.className " >  
</jsp:useBean>
```

Session Tracking in JSP

Session Tracking :

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a new connection to the Web server and the server does not keep any record of previous client request. Session tracking is a mechanism that is used to maintain state about a series of requests from the same user (requests originating from the same browser) across some period of time. A session id is a unique token number assigned to a specific user for the duration of that user's session.

Need Of Session Tracking :

HTTP is a stateless protocol so When there is a series of continuous request and response from a same client to a server, the server cannot identify which client is sending request. If we want to maintain the conversational state, session tracking is needed. For example, in a shopping cart application a client keeps on adding items into his cart using multiple requests. When every request is made, the server should identify in which client's cart the item is to be added. So in this scenario, there is a certain need for session tracking.

Solution is, when a client makes a request it should introduce itself by providing unique identifier every time. There are four ways to maintain session between web client and web server. When we are writing java code inside the Jsp, we may get exception inside that. In order to handle the exception we need to write try/catch blocks many times in all the Jsp pages. To avoid the try ,catch block and to centralize the exception handling use `errorPage` and `IsErrorPage` attribute inside the JSP where you want to handle all the exceptions.

Methods to track session :

- Cookies
- URL Rewriting
- Hidden Fields
- Session API

Cookies :

Cookies mostly used for session tracking. Cookie is a key value pair of information, sent by the server to the browser. This should be saved by the browser in its space in the client computer. Whenever the browser sends a request to that server it sends the cookie along with it. Then the server can identify the client using the cookie.

This is not an effective way because many time browser does not support a cookie or users can opt to disable cookies using their browser preferences. In such case, the browser will not save the cookie at client computer and session tracking fails.

URL Rewriting :

Here is a simple URL which will pass two values using GET method. You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session. For Example Original URL: `http://javawebtutor.com/jsp/name` Rewritten URL: `http://javawebtutor.com/jsp/name?sessionId=12345`. When a request is made an additional parameter is appended with the `url.sessionid=12345`, the session identifier is attached as `sessionId=12345` which can be accessed at the web server to identify the client.

Hidden Form Fields :

HTML forms have an entry that looks like `<INPUT TYPE="HIDDEN" NAME="session" VALUE="...">`. This means that, when the form is submitted, the specified name and value are included in the GET or POST data. This can be used to store information about the session. However, it has the major disadvantage that it only works if every page is dynamically generated, since the whole point is that each session has a unique identifier.

The session Object :

Servlets provided HttpSession Interface will provides a way to identify a user across multiple request to a Web site and to store information about that user.

For JSPs, by default a session is automatically created for the request if one does not exist. So if your starting page is a JSP, a session would have already been created when you get the first request.

1. Setting Session :

Before we validate or check the existing session it is important to know that how we can set session in JSP. We need to use `session.setAttribute("ATTRIBUTE NAME", "ATTRIBUTE VALUE")` method for setting value in session. you can set as many attribute as you want.

2. Retrieving values from session attribute :

To retrieve value from session attribute we need to use following method. `session.getAttribute("attribute name");`

The information can be stored in the session for the current session by `setAttribute()` and then retrieve by `getAttribute()` method, whenever needed.

Setting Session : `session.setAttribute("username", name);`

Getting Session : `session.getAttribute("username")`

To understand a session let us create a very simple example of getting the name from the user and saving it in the session, we will retrieve this name from session on next page and display on the page.

index.jsp

```
?  
1 <%@ page isErrorPage="true" %>  
2 <html>  
3 <head>  
4 <title>Session Management Example</title>  
5 </head>  
6 <body>  
7 <form method="post" action="firstpage.jsp">  
8 <font size=5>Enter your name<input type="text" name="name"></font><br><br>  
9 <font size=5>Enter your password<input type="password" name="password">  
10</font><br><br>  
11<input type="submit" name="submit" value="submit">  
12</form>  
13</body>  
14</html>
```

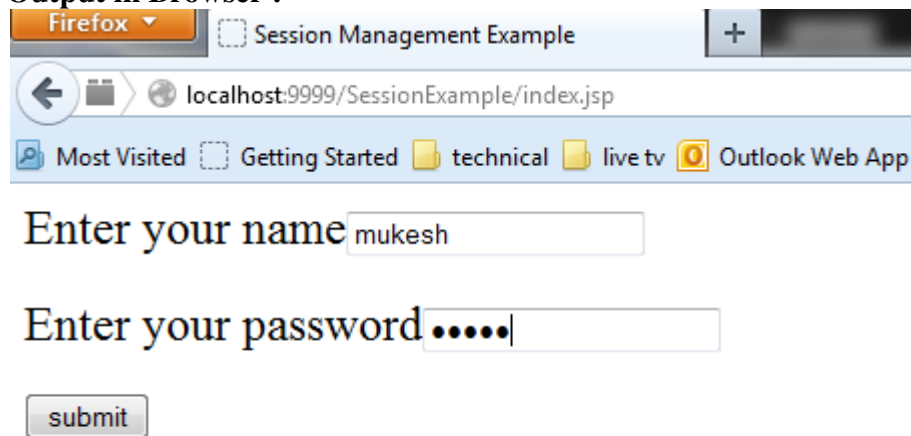
firstpage.jsp

```
?  
1 <%  
2 String name = request.getParameter("name");  
3 String password = request.getParameter("password");  
4 if (name.equals("mukesh") && password.equals("kumar")) {  
5 session.setAttribute("username", name);  
6 response.sendRedirect("secondpage.jsp");  
7 }  
8 else {  
9     response.sendRedirect("index.jsp");  
10 }  
11%>
```

secondpage.jsp

```
?  
1<html>  
2<head>  
3<title>Welcome in the program of session</title>  
4</head>  
5<body>  
6<font size = 5>Hello <%= session.getAttribute("username") %></font>  
7</body>  
8</html>
```

Output in Browser :



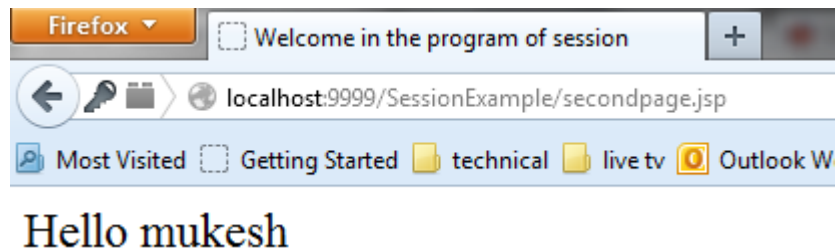
Firefox Session Management Example +

localhost:9999/SessionExample/index.jsp

Most Visited Getting Started technical live tv Outlook Web App

Enter your name

Enter your password



Connecting to DataBase using JSP

In this chapter, we will discuss how to access database with JSP. We assume you have good understanding on how JDBC application works. Before starting with database access through a JSP, make sure you have proper JDBC environment setup along with a database.

```
<%@ 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>SELECT Operation</title>
</head>
```

```

<body>
  <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://localhost/TEST"
    user = "root" password = "pass123"/>
  <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>

```

Access the above JSP, the following result will be displayed –

| Emp ID | First Name | Last Name | Age |
|--------|------------|-----------|-----|
| 100 | Zara | Ali | 18 |
| 101 | Mahnaz | Fatma | 25 |
| 102 | Zaid | Khan | 30 |
| 103 | Sumit | Mittal | 28 |

INSERT Operation

Following example shows how we can execute the SQL INSERT statement using JTSL in JSP programming –

```

<%@ 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>JINSERT Operation</title>
</head>

<body>
  <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://localhost/TEST"
    user = "root" password = "pass123"/>
  <sql:update dataSource = "${snapshot}" var = "result">
    INSERT INTO Employees VALUES (104, 2, 'Nuha', 'Ali');
  </sql:update>
  <sql:query dataSource = "${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>

```

Access the above JSP, the following result will be displayed –

Emp ID First Name Last Name Age

| | | | |
|-----|--------|--------|----|
| 100 | Zara | Ali | 18 |
| 101 | Mahnaz | Fatma | 25 |
| 102 | Zaid | Khan | 30 |
| 103 | Sumit | Mittal | 28 |
| 104 | Nuha | Ali | 2 |

UNIT-V JAVASCRIPT

Java Script

JavaScript is the premier client-side *interpreted scripting language*. It's widely used in tasks ranging from the validation of form data to the creation of complex user interfaces. **Dynamic HTML** is a combination of the content formatted using HTML, CSS, Scripting language and DOM. By combining all of these technologies, we can create interesting and interactive websites.

History of JavaScript:

Netscape initially introduced the language under the name **LiveScript** in an early beta release of Navigator 2.0 in 1995, and the focus was on form validation. After that, the language was renamed JavaScript. After Netscape introduced JavaScript in version 2.0 of their browser, Microsoft introduced a clone of JavaScript called **JScript** in Internet Explorer 3.0.

What a JavaScript can do?

JavaScript gives web developers a programming language for use in web pages & allows them to do the following:

- ☐ JavaScript gives HTML designers a programming tool
- ☐ JavaScript can be used to validate data
- ☐ JavaScript can read and write HTML elements
- ☐ Create pop-up windows
- ☐ Perform mathematical calculations on data
- ☐ React to events, such as a user rolling over an image or clicking a button
- ☐ Retrieve the current date and time from a user's computer or the last time a document was modified
- ☐ Determine the user's screen size, browser version, or screen resolution
- ☐ JavaScript can put dynamic text into an HTML page
- ☐ JavaScript can be used to create cookies

Advantages of JavaScript:

- ☐ Less server interaction
- ☐ Immediate feedback to the visitors
- ☐ Increased interactivity
- ☐ Richer interfaces
- ☐ Web surfers don't need a special plug-in to use your scripts
- ☐ Java Script is relatively secure.

Limitations of JavaScript:

We cannot treat JavaScript as a full-fledged programming language. It lacks some of the important features like:

- ☐ Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.

- ❑ JavaScript cannot be used for networking applications because there is no such support available.
- ❑ JavaScript doesn't have any multithreading or multiprocessing capabilities.
- ❑ If your script doesn't work then your page is useless.

Points to remember:

- ❑ JavaScript is case-sensitive
- ❑ Each line of code is terminated by a semicolon
- ❑ Variables are declared using the keyword **var**
- ❑ Scripts require neither a **main** function nor an **exit** condition. There are major differences between scripts and *proper* programs. Execution of a script starts with the first line of code & runs until there is no more code

JavaScript comments:

In JavaScript, each line of comment is preceded by two slashes and continues from that point to the end of the line.

//this is javascript comment

Block comments or Multiline comments: **/* */**

JavaScript is not the same as Java, which is a bigger programming language (although there are some similarities)

JavaScript and HTML Page

Having written some JavaScript, we need to include it in an HTML page. We can't execute these scripts from a command line, as the interpreter is part of the browser. The script is included in the web page and run by the browser, usually as soon as the page has been loaded. The browser is able to debug the script and can display errors.

Embedding JavaScript in HTML file:

If we are writing small scripts, or only use our scripts in few pages, then the easiest way is to include the script in the HTML code. The syntax is shown below:

```
<html>
<head>
<script language="javascript">
<!--
Javascript code here
// - - >
</head>
<body>
.....
</body>
</html>
```


JavaScript Programming Elements

- Variables, datatypes, operators
- Statements
- Arrays
- Functions
- Objects in JavaScript
- Exception Handling
- Events
- Dynamic HTML with JavaScript

VARIABLES

Like any programming language, JavaScript has variables. A variable is a name assigned to computer memory location to store data. As the name suggests, the value of the variable can vary, as the program runs. We can create a variable with the **var** statement:

var <variablename> = <some value>;

Example:

```
var sum = 0;
```

```
var str;
```

We can initialize a variable like this:

```
str = "hello";
```

Rules for variable names:

They must begin with a letter, digit or underscore character # We can't use spaces in names

Variable names are case sensitive # We can't use reserved word as a variable name.

Weakly Typed Language:

□ Most high-level languages, including C and Java, are **strongly typed**. That is, a variable must be declared before it is used, and its type must be included in its declaration. Once a variable is declared, its type cannot be changed.

□ As the JavaScript is **weakly typed** language, data types are not explicitly declared.

Example: var num;

```
num = 3;
```

```
num = "San Diego";
```

First, when the variable **num** is declared, it is empty. Its data type is actually the type **undefined**. Then we assign it to the number 3, so its data type is **numeric**. Next we reassign it to the string "San Diego", so the variable's type is now **string**.

Example:

```
<html>
```

```
<body>
```

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

```
var s;
```

```
s = "Hello";
```

```
alert(typeof s);
```

```
s = 54321;
```

```
alert(typeof s);  
</script> </body> </html>
```

DATATYPES

- JavaScript supports five primitive data types:

number string boolean undefined null.

- These types are referred to as *primitive types* because they are the basic building blocks from which more complex types can be built.
- Of the five, only **number**, **string**, and **boolean** are real data types in the sense of actually storing data.
- Undefined and null are types that arise under special circumstances.

Numeric Data Type:

□ These are numbers and can be integers (such as 2, 22 and 2000) or floating-point values (such as 23.42, -56.01, and 2E45).

□ Valid ways to specify numbers in JavaScript

10 177.5 -2.71 .333333e77 -1.7E12 3.E-5 128e+100

□ We can represent integers in one of the following 3 ways:

Decimal: The usual numbers which are having the base 10 are the decimal numbers

Octal: Octal literals begin with a leading zero, and they consist of digits from zero through seven. The following are all valid octal literals:

00 0777 024513600

HexaDecimal: Hexadecimal literals begin with a leading 0x, and they consist of digits from 0 through 9 and letters A through F. The following are all valid hexadecimal literals:

Scope of variables:

JavaScript has two scopes: global and local. A variable that is declared outside a function definition is a global variable, and its value is accessible and modifiable throughout your program. A variable that is declared inside a function definition is local. It is created and destroyed every time the function is executed, and it cannot be accessed by any code outside the function. JavaScript does not support block scope (in which a set of braces `{ . . }` defines a new scope), except in the special case of block-scoped variables.

Scope in JavaScript

A local variable can have the same name as a global variable, but it is entirely separate; changing the value of one variable has no effect on the other. Only the local version has meaning inside the function in which it is declared.

JavaScriptCopy

```
// Global definition of aCentaur.  
var aCentaur = "a horse with rider,";
```

```
// A local aCentaur variable is declared in this function.
function antiquities(){

    var aCentaur = "A centaur is probably a mounted Scythian warrior";
}

antiquities();
aCentaur += " as seen from a distance by a naive innocent.";

document.write(aCentaur);

// Output: "a horse with rider, as seen from a distance by a naive innocent."
```

In JavaScript, variables are evaluated as if they were declared at the beginning of the scope they exist in. Sometimes this results in unexpected behavior, as shown here.

JavaScriptCopy

```
var aNumber = 100;
tweak();

function tweak(){

    // This prints "undefined", because aNumber is also defined locally below.
    document.write(aNumber);

    if (false)
    {
        var aNumber = 123;
    }
}
```

When JavaScript executes a function, it first looks for all variable declarations, for example, `var someVariable;`. It creates the variables with an initial value of `undefined`. If a variable is declared with a value, for example, `var someVariable = "something";`, then it still initially has the value `undefined` and takes on the declared value only when the line that contains the declaration is executed.

JavaScript processes all variable declarations before executing any code, whether the declaration is inside a conditional block or other construct. Once JavaScript has found all the variables, it executes the code in the function. If a variable is implicitly declared inside a function - that is, if

it appears on the left side of an assignment expression but has not been declared with `var` - it is created as a global variable.

In JavaScript, an inner (nested) function stores references to the local variables that are present in the same scope as the function itself, even after the function returns. This set of references is called a closure. In the following example, the second call to the inner function outputs the same message ("Hello Bill") as the first call, because the input parameter for the outer function, `name`, is a local variable that is stored in the closure for the inner function.

JavaScriptCopy

```
function send(name) {
  // Local variable 'name' is stored in the closure
  // for the inner function.
  return function () {
    sendHi(name);
  }
}

function sendHi(msg) {
  console.log('Hello ' + msg);
}

var func = send('Bill');
func();
// Output:
// Hello Bill
sendHi('Pete');
// Output:
// Hello Pete
func();
// Output:
// Hello Bill
```

Block-scoped variables

Internet Explorer 11 introduces support for `let` and `const`, which are block-scoped variables. For these variables, the braces `{ . . }` define a new scope. When you set one of these variables to a particular value, the value applies only to the scope in which it is set.

The following example illustrates the use of `let` and block-scoping.

```
let x = 10;
var y = 10;
{
```

```

let x = 5;
var y = 5;
{
  let x = 2;
  var y = 2;
  document.write("x: " + x + "<br/>");
  document.write("y: " + y + "<br/>");
  // Output:
  // x: 2
  // y: 2
}
document.write("x: " + x + "<br/>");
document.write("y: " + y + "<br/>");
// Output:
// x: 5
// y: 2
}

document.write("x: " + x + "<br/>");
document.write("y: " + y + "<br/>");
// Output:
// x: 10
// y: 2

```

Understanding variable scopes in Javascript

In JavaScript, there are two types of scopes

1. Global Scope – Scope outside the outermost function attached to Window
2. Local Scope – Inside the function being executed

Let's look at the code below. We have a global variable defined in first line in global scope. Then we have a local variable defined inside the function fun().

```

var globalVar = "This is a global variable";

function fun(){
  var localVar = "This is a local variable";
  console.log(globalVar);
  console.log(localVar);
}

fun();

// OUTPUT
// This is a global variable
// This is a local variable

```

When we execute the function fun(), the output shows that both global as well as local variables are accessible inside the function as we are able to console.log them. This shows that inside the function we have access to both global variables (declared outside the function) and local variables (declared inside the function). Let's move the console.log statements outside the

function and put them just after calling the function.

```
var globalVar = "This is a global variable";

function fun(){
  var localVar = "This is a local variable";
}

fun();
console.log(globalVar);
console.log(localVar);
// OUTPUT
// This is a global variable
// Uncaught ReferenceError: localVar is not defined
```

We are still able to see the value of global variable, but for local variable console.log throws an error. This is because now the console.log statements are present in global scope where they have access to global variables but cannot access the local variables.

Word of caution: Whenever you are declaring variables, always use the prefix var. If you don't use the var keyword, then the variables are by default created in the global scope. For instance, in the above example, let's just remove the keyword var before the declaration of localVar.

```
var globalVar = "This is a global variable";

function fun(){
  localVar = "This is a local variable";
}

fun();
console.log(globalVar);
console.log(localVar);
// OUTPUT
// This is a global variable
// This is a local variable
```

We are now able to console.log the local variable as well because the localVar was created in the global scope as we missed the keyword var while declaring it. What really happened is that as we didn't use the var keyword, JavaScript first searched the localVar in local scope, then in the global scope. As there was no existing global variable by that name, so it created a new global variable. One of the most asked questions in interviews is the scenario where the global as well as local variable has the same name. Let's see what happens then.

```
var name = "Geeks for geeks 1";

function fun(){
  var name = "Geeks for geeks 2";
}

fun();
console.log(name);
// OUTPUT
// Geeks for geeks 1
```

In this example, we have declared a local as well as global variable "name". What matters here is the scope in which we are accessing it. In the above example, we are accessing it in global scope, so it will output the global variable as local variable is not present in its scope. Let's move the console.log statement inside the function fun().

```

var name = "Geeks for geeks 1";

function fun(){
  var name = "Geeks for geeks 2";
  console.log(name);
}

fun();

// OUTPUT
// Geeks for geeks 2

```

Inside the function fun(), both the local as well as global variables are accessible. But when we console.log the variable name, firstly JavaScript tries to find a local variable in the current scope. It finds the local variable and outputs it. Otherwise it would have search for the variable “name” in the outer scope (which in this case is global scope).

What if we want to access the global variable instead of local one here. Well, the window object comes to our rescue. All the global variables are attached to window object and thus we can access the global variable name as shown in example below.

```

var name = "Geeks for geeks 1";

function fun(){
  var name = "Geeks for geeks 2";
  console.log(window.name);
}

fun();

// OUTPUT
// Geeks for geeks 1

```

After discussing about scopes in JavaScript, guessing the output of below code fragments should be a cakewalk.

```

function fun(){
  function fun2(){
    i = 100;
  }
  fun2();
  console.log(i);
}
fun();

// OUTPUT
// 100

```

```

function fun(){
  function fun2(){
    var i = 100;
  }
  fun2();
  console.log(i);
}
fun();

// OUTPUT
// Uncaught ReferenceError: i is not defined

```

In the first example as we didn’t use the keyword var, the variable “i” was assumed to be declared in global scope and thus the output was 100. In the second example, “i” became a local variable and thus was not accessible outside the scope of that function.

Let's look at another example.

```
function fun(){
  if(true){
    var i = 100;
  }
  console.log(i);
}
fun();
// OUTPUT
// 100
```

Well, the output should have been a ReferenceError, right? Wrong! The if block does not create a new scope in JavaScript. Only a new function creates a new scope. Always remember, JavaScript has function scope and not block scope. So in the above example, all the code inside the function fun() is in the same scope and thus the output is 100.

After understanding the concept of scopes in JavaScript, let's look at practical example of these concepts. Suppose you want to implement something like “private variables of class in C++/Java” in JavaScript. Checkout the code snippet below for the exact code you would need to write.

```
var person = function () {
  // Private
  var name = "GeeksForGeeks";
  return {
    getName : function () {
      return name;
    },
    setName : function (newName) {
      name = newName;
    }
  };
}();
console.log(person.name);           // Undefined
console.log(person.getName());      // "GeeksForGeeks"
person.setName("GeeksQuiz");
console.log(person.getName());      // "GeeksQuiz"
```

In the above code, you have a variable “name” which acts as a private variable as it is not directly accessible outside the scope of function person(). But the variable “name” is accessible by the functions inside the scope of function person(). This lets us create the getter and setter function to access the variables, thereby implementing the data hiding concept of classes in OOPS.

Conditional Structure and Operators

JavaScript Arithmetic Operators

Arithmetic operators are used to perform arithmetic between variables and/or values. Given that y=5, the table below explains the arithmetic operators:

| Operator | Description | Example | Result |
|----------|------------------------------|---------|--------|
| + | Addition | x=y+2 | x=7 |
| - | Subtraction | x=y-2 | x=3 |
| * | Multiplication | x=y*2 | x=10 |
| / | Division | x=y/2 | x=2.5 |
| % | Modulus (division remainder) | x=y%2 | x=1 |
| ++ | Increment | x=++y | x=6 |
| -- | Decrement | x--y | x=4 |

JavaScript Assignment Operators

Assignment operators are used to assign values to JavaScript variables. Given that $x=10$ and $y=5$, the table below explains the assignment operators:

| Operator | Example | Same As | Result |
|----------|---------|----------|--------|
| = | $x=y$ | | $x=5$ |
| += | $x+=y$ | $x=x+y$ | $x=15$ |
| -= | $x-=y$ | $x=x-y$ | $x=5$ |
| *= | $x*=y$ | $x=x*y$ | $x=50$ |
| /= | $x/=y$ | $x=x/y$ | $x=2$ |
| %= | $x\%=y$ | $x=x\%y$ | $x=0$ |

Comparison Operators Comparison operators are used in logical statements to determine equality or difference between variables or values. Given that $x=5$, the table below explains the comparison operators:

| Operator | Description | Example |
|----------|--------------------------------------|---------------------------------------|
| == | is equal to | $x==8$ is false |
| === | is exactly equal to (value and type) | $x===5$ is true $x==="5"$ is false |
| != | is not equal | $x!=8$ is true |
| > | is greater than | $x>8$ is false |
| < | is less than | $x<8$ is true |
| >= | is greater than or equal to | $x>=8$ is false |
| <= | is less than or equal to | $x<=8$ is true |

Logical Operators

Logical operators are used to determine the logic between variables or values. Given that $x=6$ and $y=3$, the table below explains the logical operators:

| Operator | Description | Example |
|----------|-------------|-----------------------------------|
| && | and | $(x < 10 \ \&\& \ y > 1)$ is true |
| | or | $(x===5 \ \ y===5)$ is false |
| ! | not | $!(x===y)$ is true |

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

`variablename=(condition)?value1:value2`

Example

```
greeting=(visitor=="PRES")?"Dear President ":"Dear ";
```

Conditional Statements

Very often when you write code, you want to perform different actions for different decisions. You can use conditional statements in your code to do this.

In JavaScript we have the following conditional statements:

- if statement - use this statement to execute some code only if a specified condition is true
- if...else statement - use this statement to execute some code if the condition is true and another code if the condition is false
- if...else if...else statement - use this statement to select one of many blocks of code to be executed
- switch statement - use this statement to select one of many blocks of code to be executed

If Statement

Use the if statement to execute some code only if a specified condition is true.

Syntax

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

Note that if is written in lowercase letters. Using uppercase letters (IF) will generate a JavaScript error!

Example

```
<script type="text/javascript">  
//Write a "Good morning" greeting if  
//the time is less than 10
```

```
var d=new Date();  
var time=d.getHours();  
  
if (time<10)  
{  
  document.write("<b>Good morning</b>");  
}  
</script>
```

If...else Statement

Use the if...else statement to execute some code if a condition is true and another code if the condition is not true.

Syntax

```
if (condition)  
{  
  code to be executed if condition is true  
}  
else  
{  
  code to be executed if condition is not true  
}
```

```
}
```

Example

```
<script type="text/javascript">  
//If the time is less than 10, you will get a "Good morning" greeting.  
//Otherwise you will get a "Good day" greeting.
```

```
var d = new Date();  
var time = d.getHours();  
  
if (time < 10)  
{  
    document.write("Good morning!");  
}  
else  
{  
    document.write("Good day!");  
}  
</script>
```

If...else if...else Statement

Use the if....else if...else statement to select one of several blocks of code to be executed.

Syntax

```
if (condition1)  
{  
    code to be executed if condition1 is true  
}  
else if (condition2)  
{  
    code to be executed if condition2 is true  
}  
else  
{  
    code to be executed if condition1 and condition2 are not true  
}
```

Example

```
script type="text/javascript">  
var d = new Date()  
var time = d.getHours()  
if (time<10)  
{  
    document.write("<b>Good morning</b>");  
}
```

```
else if (time>10 && time<16)
{
    document.write("<b>Good day</b>");
}
else
{
    document.write("<b>Hello World!</b>");
}
</script>
```

The JavaScript Switch Statement

Use the switch statement to select one of many blocks of code to be executed.

Syntax

```
switch(n)
{
case 1:
    execute code block 1
    break;
case 2:
    execute code block 2
    break;
default:
    code to be executed if n is different from case 1 and 2
}
```

JavaScript Popup Boxes

JavaScript has three kind of popup boxes: Alert box, Confirm box, and Prompt box.

Alert Box

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

Syntax

```
alert("sometext");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_alert()
{
alert("I am an alert box!");
}
</script>
</head>
<body>
<input type="button" onclick="show_alert()" value="Show alert box" />
</body>
</html>
```

Confirm Box

A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed. If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax

```
confirm("sometext");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_confirm()
{
var r=confirm("Press a button");
if (r==true)
{
alert("You pressed OK!");
}
else
{
alert("You pressed Cancel!");
}
}
</script>
</head>
<body>
<input type="button" onclick="show_confirm()" value="Show confirm box" />
</body>
</html>
```

Prompt Box

A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value. If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax

```
prompt("sometext","defaultvalue");
```

Example

```
<html>
<head>
<script type="text/javascript">
function show_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
{
document.write("Hello " + name + "! How are you today?");
}
}
</script>
</head>
<body>

<input type="button" onclick="show_prompt()" value="Show prompt box" />

</body>
</html>
```

JavaScript Loops

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal lines in a script we can use loops to perform a task like this.

In JavaScript, there are two different kind of loops:

- **for** - loops through a block of code a specified number of times
- **while** - loops through a block of code while a specified condition is true

The for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

```
for (variable=startvalue;variable<=endvalue;variable=variable+increment)
{
code to be executed
}
```

Example

The example below defines a loop that starts with `i=0`. The loop will continue to run as long as `i` is less than, or equal to 5. `i` will increase by 1 each time the loop runs.

Note: The increment parameter could also be negative, and the `<=` could be any comparing statement.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=5;i++)
{
document.write("The number is " + i);
document.write("<br />");
}
</script>
</body>
</html>
```

JavaScript While Loop

Loops execute a block of code a specified number of times, or while a specified condition is true.

The while Loop

The while loop loops through a block of code while a specified condition is true.

Syntax

```
while (variable<=endvalue)
{
    code to be executed
}
```

Note: The `<=` could be any comparing operator.

Example

The example below defines a loop that starts with `i=0`. The loop will continue to run as long as `i` is less than, or equal to 5. `i` will increase by 1 each time the loop runs:

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
while (i<=5)
{
  document.write("The number is " + i);
  document.write("<br />");
  i++;
}
</script>
</body>
</html>
```

The do...while Loop

The do...while loop is a variant of the while loop. This loop will execute the block of code ONCE, and then it will repeat the loop as long as the specified condition is true.

Syntax

```
do
{
  code to be executed
}
while (variable<=endvalue);
```

Example

The example below uses a do...while loop. The do...while loop will always be executed at least once, even if the condition is false, because the statements are executed before the condition is tested:

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
do
{
  document.write("The number is " + i);
  document.write("<br />");
  i++;
}
while (i<=5);
</script>
</body>
</html>
```

The break Statement

The break statement will break the loop and continue executing the code that follows after the loop (if any).

Example

```
<html>
<body>
<script type="text/javascript">
var i=0;
for (i=0;i<=10;i++)
{
  if (i==3)
  {
    break;
  }
  document.write("The number is " + i);
  document.write("<br />");
}
</script>
</body>
</html>
```

The continue Statement

The continue statement will break the current loop and continue with the next value.

Example

```
<html>
<body>
<script type="text/javascript">
var i=0
for (i=0;i<=10;i++)
{
  if (i==3)
  {
    continue;
  }
  document.write("The number is " + i);
  document.write("<br />");
}
</script>
</body>
</html>
```

JavaScript For...In Statement

The for...in statement loops through the elements of an array or through the properties of an object.

Syntax

```
for (variable in object)
{
  code to be executed
}
```

Note: The code in the body of the for...in loop is executed once for each element/property.

Note: The variable argument can be a named variable, an array element, or a property of an object.

Example

Use the for...in statement to loop through an array:

Example

```
<html>
<body>

<script type="text/javascript">
var x;
var mycars = new Array();
mycars[0] = "Saab";
mycars[1] = "Volvo";
mycars[2] = "BMW";

for (x in mycars)
{
    document.write(mycars[x] + "<br />");
}
</script>

</body>
</html>
```

FUNCTIONS

A function is a piece of code that performs a specific task. The function will be executed by an event or by a call to that function. We can call a function from anywhere within the page (or even from other pages if the function is embedded in an external **.js** file). JavaScript has a lot of builtin functions.

Defining functions:

JavaScript function definition consists of the **function** keyword, followed by the name of the function.

A list of arguments to the function are enclosed in parentheses and separated by commas. The statements within the function are enclosed in curly braces { }.

Syntax:

```
function functionname(var1,var2,...,varX)
{
    some code
}
```

Parameter Passing:

Not every function accepts parameters. When a function receives a value as a parameter, that value is given a name and can be accessed using that name in the function. The names of parameters are taken from the function definition and are applied in the order in which parameters are passed in.

- Primitive data types are passed by value in JavaScript. This means that a copy is made of a variable when it is passed to a function, so any manipulation of a parameter holding primitive data in the body of the function leaves the value of the original variable untouched.

- Unlike primitive data types, composite types such as arrays and objects are passed by reference rather than value.

Examining the function call:

In JavaScript parameters are passed as arrays. Every function has two properties that can be used to find information about the parameters:

functionname.arguments

This is an array of parameters that have been passed

functionname.arguments.length

This is the number of parameters that have been passed into the function

Example:

```
<html>
<body>
<script language="javascript">
function fun(a,b){
var msg = fun.arguments[0]+".." +fun.arguments[1]; //referring a,b values
alert(msg);
}
fun(10,20); //function call
fun("abc","vit"); //function call
</script>
</body>
</html>
```

Returning values

The **return** statement is used to specify the value that is returned from the function. So functions that are going to return a value must use the **return** statement.

Example:

```
function prod(a,b)
{
x=a*b; return x;
}
```

Scoping Rules:

Programming languages usually impose rules, called scoping, which determine how a variable can be accessed. JavaScript is no exception. In JavaScript variables can be either *local* or *global*.

global

Global scoping means that a variable is available to all parts of the program. Such variables are declared outside of any function.

local

Local variables are declared inside a function. They can only be used by that function.

GLOBAL FUNCTIONS

| | | |
|-----------------------|---|--|
| >parseInt() | <p>>Converts the string argument to an integer and returns the value. If the string cannot be converted, it returns NaN. Like parseFloat(), this method should handle strings starting with numbers and peel off what it needs, but other mixed strings will not be converted.</p> | <pre>>var x; x = parseInt("-53"); // x is -53 x = parseInt("33.01568"); // x is 33 x = parseInt("47.6k-red-dog"); // x is 47 x = parseInt("a567.34"); // x is NaN x = parseInt("won't work"); // x is NaN</pre> |
| >unescape() | <p>>Takes a hexadecimal string value containing some characters of the form %xx and returns the ISO-Latin-1 ASCII equivalent of the passed values.</p> | <pre>>Var aString="O%27Neill%20%26%20Sons"; aString = unescape(aString); // aString = "O'Neill & Sons" aString = unescape("%64%56%26%23"); // aString = "dV&#"</pre> |

| Method | Description | Example |
|-------------------------|--|---|
| >escape() | >Takes a string and returns a string where all non-alphanumeric characters such as spaces, tabs, and special characters have been replaced with their hexadecimal equivalents in the form %xx. | <pre>>var aString="O'Neill & Sons"; // aString = "O'Neill & Sons" aString = escape(aString); // aString="O%27Neill%20%26%20Sons"</pre> |
| >eval() | >Takes a string and executes it as JavaScript code. | <pre>>var x; var aString = "5+9"; x = aString; // x contains the string "5+9" x = eval(aString); // x will contain the number 14</pre> |
| >isFinite() | >Returns a Boolean indicating whether its number argument is finite. | <pre>>var x; x = isFinite('56'); // x is true x = isFinite(Infinity) // x is false</pre> |
| >isNaN() | >Returns a Boolean indicating whether its number argument is NaN . | <pre>>var x; x = isNaN('56'); // x is False x = isNaN(0/0) // x is true x = isNaN(NaN); // x is true</pre> |
| >parseFloat() | >Converts the string argument to a floating-point number and returns the value. If the string cannot be converted, it returns NaN . The method should handle strings starting with numbers and peel off what it needs, but other mixed strings will not be converted. | <pre>>var x; x = parseFloat("33.01568"); // x is 33.01568 x = parseFloat("47.6k-red-dog"); // x is 47.6 x = parseFloat("a567.34"); // x is NaN x = parseFloat("won't work"); // x is NaN</pre> |

Event Handlers

An event occurs when something happens in a browser window. The kinds of events that might occur are due to:

- A document loading
- The user clicking a mouse button
- The browser screen changing size

When a function is assigned to an event handler, that function is run when that event occurs.

A handler that is assigned from a script used the syntax '[element].[event] = [function];', where [element] is a page element, [event] is the name of the selected event and [function] is the name of the function that occurs when the event takes place.

What is an Event ?

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.

When the page loads, it is called an event. When the user clicks a button, that click too is an event. Other examples include events like pressing any key, closing a window, resizing a window, etc.

Developers can use these events to execute JavaScript coded responses, which cause buttons to close windows, messages to be displayed to users, data to be validated, and virtually any other type of response imaginable.

Events are a part of the Document Object Model (DOM) Level 3 and every HTML element contains a set of events which can trigger JavaScript Code.

Please go through this small tutorial for a better understanding [HTML Event Reference](#). Here we will see a few examples to understand a relation between Event and JavaScript –

onclick Event Type

This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type.

Example

Try the following example.

```
<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>
```

Output

onsubmit Event type

onsubmit is an event that occurs when you try to submit a form. You can put your form validation against this event type.

Example

The following example shows how to use onsubmit. Here we are calling a **validate()** function before submitting a form data to the webserver. If **validate()** function returns true, the form will be submitted, otherwise it will not submit the data.

Try the following example.

```
<html>
<head>

  <script type="text/javascript">
    <!--
      function validation() {
        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>
```

onmouseover and onmouseout

These two event types will help you create nice effects with images or even with text as well. The **onmouseover** event triggers when you bring your mouse over any element and the **onmouseout** triggers when you move your mouse out from that element. Try the following example.

```
<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>
```

Output

HTML 5 Standard Events

The standard HTML 5 events are listed here for your reference. Here script indicates a Javascript function to be executed against that event.

| Event handler | Applies to: | Triggered when: |
|--------------------|--|--|
| onAbort | Image | The loading of the image is cancelled. |
| onBlur | Button, Checkbox, 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 | Select, Text, TextArea | The data in the form element is changed by the user. |
| onClick | Button, Checkbox, Link, Radio, Reset, Submit | The object is clicked on. |
| onDbClick | Document, Link | The object is double-clicked on. |
| onError | Image | A JavaScript error occurs. |
| onFocus | Button, Checkbox, Password, Radio, Reset, Select, Submit, Text, TextArea | The object in question gains focus (e.g. by clicking on it or pressing the TAB key). |
| onKeyDown | Image, Link, TextArea | The user presses a key. |
| onKeyPress | Image, Link, TextArea | The user presses or holds down a key. |
| onKeyUp | Image, Link, TextArea | The user releases a key. |
| onLoad | Image, Window | The whole page has finished loading. |
| onMouseDown | Button, 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, 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. |

Document Object Model (DOM) :

*The **DOM** defines what properties of a document can be retrieved and changed, and the methods that can be performed. (or)*

*The Browser **DOM** specifies how JavaScript (and other programming languages) can be used to access information about a document. Then you can perform calculations and decisions based on the values retrieved from the document using JavaScript*

Example: We can retrieve properties such as the value of the **height** attribute of any image, the **href** attribute of any link, or the length of a password entered into a text box in a form. Meanwhile, methods allow us to perform actions such as **reset()** or **submit()** methods on a form that allows you to reset or submit a form.

DOM

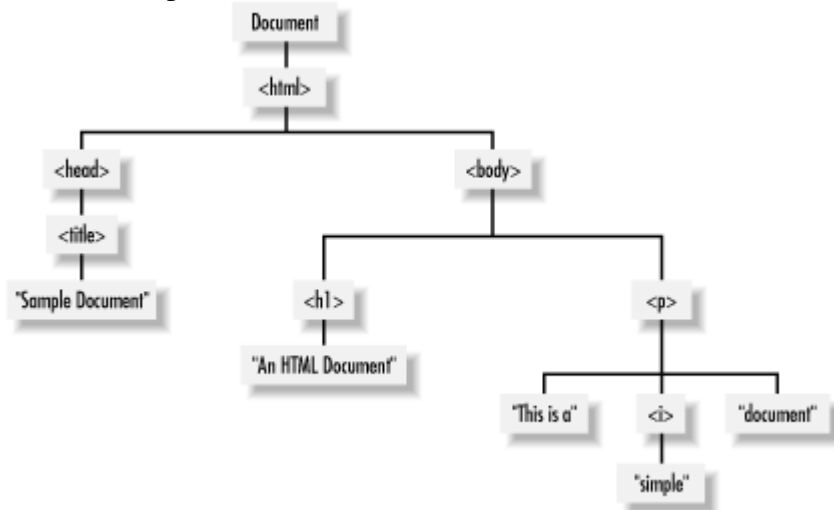
- The Document Object Model (DOM) is an API that allows programs to interact with HTML (or XML) documents
- The primary function of the Document Object Model is to view, access, and change the structure of an HTML document separate from the content contained within it.
- The DOM will provide you with methods and properties to retrieve, modify, update, and delete parts of the document you are working on. The properties of the Document Object Model are used to describe the web page or document and the methods of the Document Object Model are used for working with parts of the web page.
- In DOM, HTML document is represented in tree like structure. It constructs a hierarchical tree structure for a HTML document to traverse and to manipulate the document.
- For example,

```
<html>
<head>
  <title>Sample Document</title>
</head>
<body>
  <h1>An HTML Document</h1>
  <p>This is a <i>simple</i> document.
</body>
</html>
```

DOM Hierarchy

- The objects in the web page follow a strict hierarchy, where the **window** object is the very top level. Because **window** is the top level “root” object it can be omitted in the address syntax. For instance, the **window.document.bgColor** property, which stores the value of the window’s current background color, can be addressed simply as **document.bgColor**
- Several of the DOM objects have properties that contain an array of elements in that web page. For example, with **document.images[]**, the **images[]** array is a property of the document object that will store the URL address of each image contained on that web page. The URL of the first image in the HTML code is stored in the array at **document.images[0]**

The DOM representation of this document is as follows:



The node directly above a node is the *parent* of that node. The nodes one level directly below another node are the *children* of that node. Nodes at the same level, and with the same parent, are *siblings*. The set of nodes any number of levels below another node are the *descendants* of that node.

Types of nodes

- There are many types of nodes in the DOM document tree that specifies what kind of node it is. Every Object in the DOM document tree has properties and methods defined by the Node host object.

The following table lists the non method properties of Node object.

TABLE 5.2: Non-method properties of Node instances.

| Property | Description |
|------------------------------|---|
| <code>nodeType</code> | Number representing the type of node (<code>Element</code> , <code>Comment</code> , etc.). |
| <code>nodeName</code> | String providing a name for this Node (form of name depends on the <code>nodeType</code> ; see text). |
| <code>parentNode</code> | Reference to object that is this node's parent. |
| <code>childNodes</code> | Acts like a read-only array containing this node's child nodes. Has <code>length</code> 0 if this node has no children. |
| <code>previousSibling</code> | Previous sibling of this node, or <code>null</code> if no previous sibling exists. |
| <code>nextSibling</code> | Next sibling of this node, or <code>null</code> if no next sibling exists. |
| <code>attributes</code> | Acts like a read-only array containing <code>Attr</code> instances representing this node's attributes. |

The following table lists the node types commonly encountered in HTML documents and the `nodeType` value for each one.

| Node Type | <code>nodeType</code> constant | <code>nodeType</code> value |
|-----------|--------------------------------|-----------------------------|
|-----------|--------------------------------|-----------------------------|

| | | |
|------------------|-----------------------------|----|
| Element | Node.ELEMENT_NODE | 1 |
| Text | Node.TEXT_NODE | 3 |
| Document | Node.DOCUMENT_NODE | 9 |
| Comment | Node.COMMENT_NODE | 8 |
| DocumentFragment | Node.DOCUMENT_FRAGMENT_NODE | 11 |
| Attr | Node.ATTRIBUTE_NODE | 2 |

The following table lists the method properties of Node object.

TABLE 5.4: Method properties of Node instances.

| Method | Functionality |
|---------------------------------------|---|
| <code>hasAttributes()</code> | Returns Boolean indicating whether or not this node has attributes. |
| <code>hasChildNodes()</code> | Returns Boolean indicating whether or not this node has children. |
| <code>appendChild(Node)</code> | Adds the argument Node to the end of the list of children of this node. |
| <code>insertBefore(Node, Node)</code> | Adds the first argument Node in the list of children of this node immediately before the second argument Node (or at end of child list if second argument is <code>null</code>). |
| <code>removeChild(Node)</code> | Removes the argument Node from this node's list of children. |
| <code>replaceChild(Node, Node)</code> | In the list of children of this node, replace the second argument Node with the first. |

Traversing a Document: Counting the number of Tags

The DOM represents an HTML document as a tree of Node objects. With any tree structure, one of the most common things to do is traverse the tree, examining each node of the tree in turn.

The following program shows one way to do this.

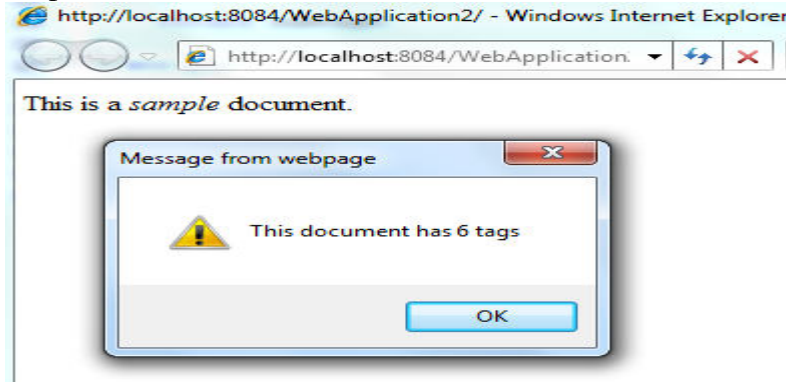
```

<html>
<head>
<script>
function countTags(n)
{
    // n is a Node
    var numtags = 0;           // Initialize the tag counter
    if (n.nodeType == 1 ) // Check if n is an Element
        numtags++;           // Increment the counter if so
    var children = n.childNodes; // Now get all children of n
    for(var i=0; i < children.length; i++)
    { // Loop through the children
        numtags += countTags(children[i]); // Recurse on each one
    }
    return numtags;           // Return the total number of tags
}
</script>
</head>
<body onload="alert('This document has ' + countTags(document) + ' tags')">
This is a <i>sample</i> document.

```

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

Output



Finding Specific Elements in a Document

The ability to traverse all nodes in a document tree gives us the power to find specific nodes. When programming with the DOM API, it is quite common to need a particular node within the document or a list of nodes of a specific type within the document.

You can use `getElementById()` and `getElementsByName()` methods of Document Object to obtain a list of any type of HTML element. For example, to find all the tables within a document, you'd do this:

```
var tables = document.getElementsByTagName("table");
```

This code finds `<table>` tags and returns elements in the order in which they appear in the document.

`getElementById()` to find a specific element whereas `getElementsByName()` returns an array of elements rather than a single element.

The following program illustrates this.

```
<html>
  <head>
    <script type="text/javascript">
      function f1()
      {
        alert(document.getElementById("p1").nodeName);
      }
    </script>
  </head>
  <body>
    <p id="p1"> Program is coded to find paragraph element is present in the document or not
    using
      its id attribute</p>
```

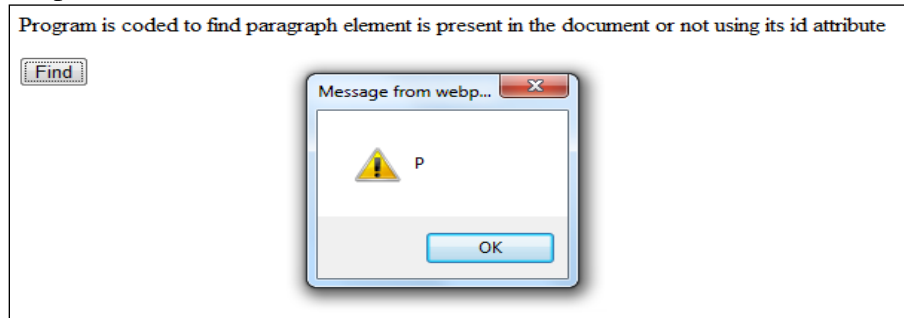
```

        <input type="button" value="Find" onclick="f1();" />

    </body>
</html>

```

Output



In the above program the method `getElementById()` finds the specific element and `nodeName` is used property return the specific element name.

Modifying a Document: Reversing the nodes of a document

DOM API lies in the features that allow you to use JavaScript to dynamically modify documents. The following examples demonstrate the basic techniques of modifying documents and illustrate some of the possibilities.

The following example includes a JavaScript function named `reverse()`, a sample document, and an HTML button that, when pressed, calls the `reverse()` function, passing it the node that represents the `<body>` element of the document. The `reverse()` function loops backward through the children of the supplied node and uses the `removeChild()` and `appendChild()` methods of the Node object to reverse the order of those children.

```

<html>
  <head><title>Reverse</title>
  <script>
    function reverse(n) {      // Reverse the order of the children of Node n
      var kids = n.childNodes; // Get the list of children
      var numkids = kids.length; // Figure out how many children there are
      for(var i = numkids-1; i >= 0; i--) { // Loop backward through the children
        var c = n.removeChild(kids[i]); // Remove a child
        n.appendChild(c);              // Put it back at its new position
      }
    }
  </script>
</head>
<body>
  <pre>
    paragraph #1
    paragraph #2
    paragraph #3
  </pre>

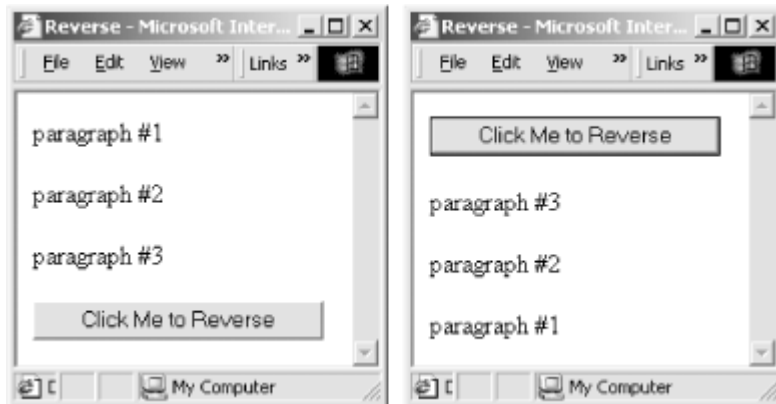
```

```

    <form>
      <input type="button" value="Click Me to reverse" onclick="reverse(document.body);"/>
    </form>
  </body>
</html>

```

when the user clicks the button, the order of the paragraphs and of the button are reversed.



Changing element Style

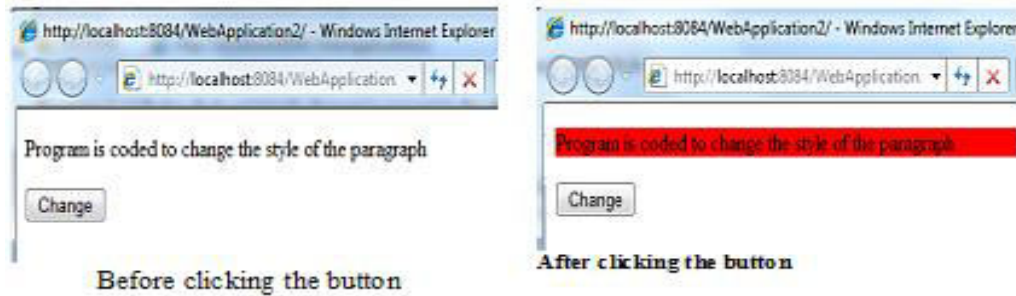
The following program illustrates how to change the element style using DOM properties and methods.

```

<html>
  <head>
    <script type="text/javascript">
      function f1()
      {
        document.getElementById("p1").style.backgroundColor="red";
      }
    </script>
  </head>
  <body>
    <p id="p1"> Program is coded to change the style of the paragraph</p>
    <input type="button" value="Change" onclick="f1();" />
  </body>
</html>

```

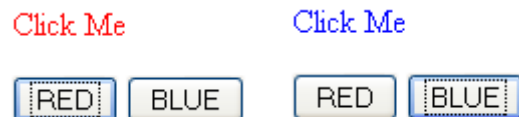
The method `getElementById()` gets the paragraph element in the document and the property `style` is used to change the background color of the paragraph to “red” as shown below.



Changing element style

```
<html>
<head>
<script type="text/javascript">
function f1()
{
    var o=document.getElementById("p1");
    o.style.color="red";
}
function f2()
{
    var o=document.getElementById("p1");
    o.style.color="blue";
}
</script>
</head>
<body>
<p id="p1">
    Click Me
</p>
<form>
<input type="button" id="b1" value="RED" onclick=f1() />
<input type="button" id="b2" value="BLUE" onclick=f2() />
</form>
</body>
</html>
```

Output

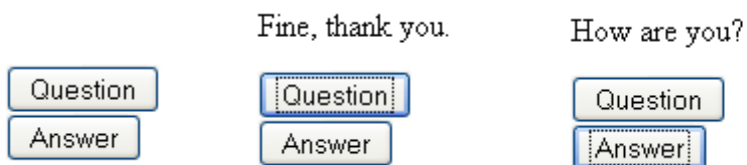


Changing HTML Content

This page shows a example of how to change a HTML page's content

```
<html>
<head>
<script type="text/javascript">
function f1()
{
    document.getElementById("p1").childNodes[0].nodeValue="Fine, thank you.";
}
function f2()
{
    document.getElementById("p1").childNodes[0].nodeValue="How are you?";
}
</script>
</head>
<body id="p1">
<pre>
<input type="button" id="b1" value="Question" onclick=f1() />
<input type="button" id="b2" value="Answer" onclick=f2() />
</pre>
</body>
</html>
```

After pressing the Question button, it adds the content, "How are you?" to the HTML document and after pressing the Answer button, it replaces the content "How are you?" with "Fine, thank you"



Removing Element from HTML documents

```
<html>
<head>
<script type="text/javascript">
function f1()
{
    var node=document.getElementById("p1");
    node.removeChild(node.childNodes[0]);
}
</script>
</head>
<body >
```

```

<pre id="p1"><input type="button" id="b1" value="Question" />
<input type="button" id="b2" value="Remove" onclick=f1() />
Example for Removing an element from HTML document.
</pre>
</body>
</html>

```

After pressing the “Remove” button, the element “Question” is removed from the document.



Example for Removing an element from HTML document.



Example for Removing an element from HTML document.

Server-side Programming: Servlet

The combination of

- HTML
- JavaScript
- DOM

is sometimes referred to as Dynamic HTML (DHTML). Web pages that include scripting are often called dynamic pages. Similarly, web server response can be static or dynamic

- **Static: HTML document is retrieved** from the file system by the server and the same returned to the client.
- **Dynamic:** In server, a **HTML document is generated** by a program in response to an HTTP request

BUILT-IN OBJECTS IN JAVASCRIPT

Javascript has many built-in objects which possess the capability of performing many tasks. Hence sometimes Javascript is referred to as an ***Object based programming language***.

Now we will discuss few commonly used objects of javascript along with their attributes & behaviors.

THE WINDOW OBJECT

PROPERTIES:

frames[]

array of frames stored in the order in which they are defined in the document

frames.length

number of frames

self

current window

opener

the window (if any) which opened the current window

parent

parent of the current window if using a frameset

status

message in the status bar

defaultStatus

default message for the status bar

name

the name of the window if it was created using the open() method and a name was specified

location

this object contains the full URL address of the document that is currently loaded in the browser, and assigning a new value to this will load the specified URL into the browser.

A typical URL address may comprise these parts:

Protocol: // host / pathname ? #hash

We can use the following properties of location object to extract individual pieces of information from URL

window.location.href

window.location.protocol

window.location.host

window.location.pathname

window.location.hash

onunload

This object can be used to specify the name of a function to be called when the user exits the web page.

METHODS:

alert("string")

opens box containing the message

confirm("string")

displays a message box with OK and CANCEL buttons

prompt("string")

displays a prompt window with field for the user to enter a text string

blur()

remove focus from current window

focus()

give focus to current window

scroll(x,y)

move the current window to the chosen x,y location

open("URL", "name", "options string")

The open() method has 3 arguments:

☐ URL to load in the pop-up window

☐ Name for the pop-up

☐ List of options

THE DOCUMENT OBJECT

A document is a web page that is being either displayed or created. The document has a number of properties that can be accessed by JavaScript programs and used to manipulate the content of the page.

PROPERTIES:

bgColor

Background color of the document

Example: write a javascript that designs 3 buttons “red”, “green”, and “yellow”. When ever the button is clicked, the document color should change accordingly

```
<html>
<head>
<script language="javascript">
function changecolor(s)
{
window.document.bgColor=s;
}
</script>
</head>
<body>
<form>
<input type="button" value="red" onclick="changecolor('red')">
<input type="button" value="green" onclick="changecolor('green')">
<input type="button" value="yellow" onclick="changecolor('yellow')">
</form>
</body> </html>
```

fgColor

Foreground color of the document

title

Title of the current document

location

This object contains the full URL address of the document that is currently loaded in the browser, and assigning a new value to this will load the specified URL into the browser.

Example:

```
<html>
<body>
<script language="javascript">
document.title="ACEcet";
function fun(){
document.location="page1.html";
}
</script>
<input type="button" value="change url" onclick="fun()">
</body>
</html>
```

Object that provides information about date and time when a webpage was last modified. This data is usually supplied to document.lastModified from the HTTP file header that is sent by the web server

Example:

```
<html>
<body>
<script language="javascript">
```

window.status = "Last updated " + document.lastModified;

</script>

<h1> ACECET</h1>

Ace engineering college

</body>

</html>

linkColor, vlinkColor,alinkColor

These can be used to set the colors for the various types of links

forms[] array of forms on the current page

forms.length

the number of form objects on the page

links[]

array of links in the current page in the order in which they appear in the document

anchors[]

an array of anchors. Any named point inside an HTML document is an anchor. Anchors are create using . These will be commonly used for moving around inside a large page.

The anchors property is an array of these names in the order in which they appear in the HTML document. Anchors can be accessed like this: document.anchors[0]

images[]

an array of images

applets[]

an array of applets

cookie

object that stores information about cookie

Methods:

write("string")

write an arbitrary string to the HTML document

writeln("string")

write a string to the HTML document and terminate it with a newline character. HTML pages can be created on the fly using JavaScript. This is done using the write or writeln methods of the document object.

Example:

document.write("<body>");

document.write("<h1>ACECET</h1>");

document.write("<form>");

clear()

clear the current document

close()

close the current document

String Object Methods

| Method | Description |
|---------------|---|
| charAt() | Returns the character at the specified index |
| charCodeAt() | Returns the Unicode of the character at the specified index |
| concat() | Joins two or more strings, and returns a copy of the joined strings |
| indexOf() | Returns the position of the first found occurrence of a specified value in a string |
| lastIndexOf() | Returns the position of the last found occurrence of a specified value in a string |

| | |
|---------------|--|
| match() | Searches for a match between a regular expression and a string, and returns the matches |
| replace() | Searches for a match between a substring (or regular expression) and a string, and replaces the matched substring with a new substring |
| search() | Searches for a match between a regular expression and a string, and returns the position of the match |
| slice() | Extracts a part of a string and returns a new string |
| split() | Splits a string into an array of substrings |
| substr() | Extracts the characters from a string, beginning at a specified start position, and through the specified number of character |
| substring() | Extracts the characters from a string, between two specified indices |
| toLowerCase() | Converts a string to lowercase letters |
| toUpperCase() | Converts a string to uppercase letters |
| valueOf() | Returns the primitive value of a String object |

DATE OBJECT

This object is used to obtain the date and time. This date and time is based on computer's local time (system's time) or it can be based on GMT. This GMT is also known as UTC i.e. Universal Coordinated Time. This is basically a world time standard. Following are the commonly used methods of Date object:

Meaning

Method

| | |
|----------------------|---|
| getTime() | It returns the number of milliseconds. This value is the difference between the current time and the time value from 1st January 1970 |
| getDate() | Returns the current date based on computers local time |
| getUTCDate() | Returns the current date obtained from UTC |
| getDay() | Returns the current day. The day number is from 0 to 6 i.e. from Sunday to Saturday |
| getUTCDay() | Returns the current day based on UTC. The day number is from 0 to 6 |
| getHours() | Returns the hour value ranging from 0 to 23 |
| getUTCHours() | Returns the hour value ranging from 0 to 23, based on UTC timing zone |
| getMinutes() | Returns the minute value ranging from 0 to 59 |
| getUTCMinutes() | Returns the minute value ranging from 0 to 59, based on UTC |
| getSeconds() | Returns the seconds value ranging from 0 to 59 |
| getUTCSeconds() | Returns the seconds value ranging from 0 to 59, based on UTC |
| getMilliseconds() | Returns the milliseconds value ranging from 0 to 999, based on local time |
| getUTCMilliseconds() | Returns the milliseconds value ranging from 0 to 999, based on UTC |
| setDate(value) | This is used to set the Date |

setHour(hr,min,sec,ms)

This is used to set the Hour

Example:

```
<html>
<head>
<title>Date Object</title>
</head>
<body>
<script type="text/javascript">
var d = new Date();
document.write("The Date is: "+d.toString()+"<br>");
document.write("Today date is: "+d.getDate()+"<br>");
document.write("UTC date is: "+d.getUTCDate()+"<br>");
document.write("Minutes: "+d.getMinutes()+"<br>");
document.write("UTC Minutes: "+d.getUTCMinutes()+"<br>");
</script>
</body> </html>
```

MATH OBJECT

For performing the mathematical computations there are some useful methods available from math object.

- ☐ sqrt(num)
- ☐ abs(num)
- ☐ ceil(num)
- ☐ floor(num)
- ☐ log(num)
- ☐ pow(a,b)
- ☐ min(a,b)
- ☐ max(a,b)
- ☐ sin(num)
- ☐ cos(num)
- ☐ tan(num)
- ☐ exp(num)
- ☐ asin(value)
- ☐ acos(value)
- ☐ atan(value)

random() - returns a psuedorandom number between 1 to 1

round(value)

In addition to the above methods, it has several properties (Numeric constants) like:

Math.E Euler constant

Math.PI 3.14159

Math.SQRT_2 The square root of 2

Math.SQRT1_2 The square root of ½

Math.LN2 Log of 2

Math.LN10 Log of 10

Example:

```

<html>
<body>
<script language="javascript">
var n = prompt("enter any number");
alert("square root is "+Math.sqrt(n));
</script>
</body>
</html>

```

FORM OBJECT

The **window.document.forms** object contains an array of all the forms in a HTML document, indexed in the order in which they appear in the HTML code.

For example, **window.document.forms[0]** addresses the first form to appear in the HTML code of a web page.

If the **id** attribute of the **<form>** element has been assigned a value, then the form can be addressed by name.

For example, a form named **reg** can be addressed as **document.forms.reg**

All the attributes assigned in the **<form>** tag can be accessed as properties of that form object.

Example:

```

<html>
<head>
<script language="javascript">
window.onload = fun;
function fun()
{
var msg = "Form name: " + document.forms.reg.id;
msg += "\nMethod: " + document.forms.reg.method;
msg += "\nAction: " + document.forms.reg.action;
window.alert(msg);
}
</script>
</head>
<body>
<form id="reg" method="post" action=mailto:abc@xyz.com>
Name: <input type="text" size=10> <br>
Age: <input type="text" size=5> <br>
</form>
</body>
</html>

```

Form elements:

The elements of the form are held in the array **window.document.forms[.elements[]**

The properties of the form elements can be accessed and set using Javascript.

The elements of the form are held in the array **window.document.forms[.elements[]**

The properties of the form elements can be accessed and set using Javascript

JavaScript Form Validation

1. JavaScript form validation
2. Example of JavaScript validation
3. JavaScript email validation

It is important to validate the form submitted by the user because it can have inappropriate values. So validation is must.

The JavaScript provides you the facility to validate the form on the client side so processing will be fast than server-side validation. So, most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile number etc fields.

Form validation normally used to occur at the server, after the client had entered all the necessary data and then pressed the Submit button. If the data entered by a client was incorrect or was simply missing, the server would have to send all the data back to the client and request that the form be resubmitted with correct information. This was really a lengthy process which used to put a lot of burden on the server.

JavaScript provides a way to validate form's data on the client's computer before sending it to the web server. Form validation generally performs two functions.

- **Basic Validation** – First of all, the form must be checked to make sure all the mandatory fields are filled in. It would require just a loop through each field in the form and check for data.
- **Data Format Validation** – Secondly, the data that is entered must be checked for correct form and value. Your code must include appropriate logic to test correctness of data.

Example

We will take an example to understand the process of validation. Here is a simple form in html format.

```
<html>

<head>
  <title>Form Validation</title>

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

```
<!--  
    // Form validation code will come here.  
    //-->  
</script>  
  
</head>  
  
<body>  
    <form action="/cgi-bin/test.cgi" name="myForm" onsubmit="return(validate());">  
        <table cellpadding="2" cellspacing="2" border="1">  
  
            <tr>  
                <td align="right">Name</td>  
                <td><input type="text" name="Name" /></td>  
            </tr>  
  
            <tr>  
                <td align="right">EMail</td>  
                <td><input type="text" name="EMail" /></td>  
            </tr>  
  
            <tr>  
                <td align="right">Zip Code</td>  
                <td><input type="text" name="Zip" /></td>  
            </tr>  
  
            <tr>  
                <td align="right">Country</td>  
                <td>  
                    <select name="Country">  
                        <option value="-1" selected>[choose yours]</option>  
                        <option value="1">USA</option>  
                        <option value="2">UK</option>  
                        <option value="3">INDIA</option>  
                    </select>  
                </td>  
            </tr>  
  
            <tr>  
                <td align="right"></td>  
                <td><input type="submit" value="Submit" /></td>  
            </tr>  
  
        </table>  
    </form>
```

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

Output

Basic Form Validation

First let us see how to do a basic form validation. In the above form, we are calling **validate()** to validate data when **onsubmit** event is occurring. The following code shows the implementation of this validate() function.

```
<script type="text/javascript">
<!--
// Form validation code will come here.
function validate()
{

    if( document.myForm.Name.value == "" )
    {
        alert( "Please provide your name!" );
        document.myForm.Name.focus() ;
        return false;
    }

    if( document.myForm.Email.value == "" )
    {
        alert( "Please provide your Email!" );
        document.myForm.Email.focus() ;
        return false;
    }

    if( document.myForm.Zip.value == "" ||
    isNaN( document.myForm.Zip.value ) ||
    document.myForm.Zip.value.length != 5 )
    {
        alert( "Please provide a zip in the format #####." );
        document.myForm.Zip.focus() ;
        return false;
    }

    if( document.myForm.Country.value == "-1" )
    {
        alert( "Please provide your country!" );
        return false;
    }
    return( true );
}
```

```
//-->  
</script>
```

Data Format Validation

Now we will see how we can validate our entered form data before submitting it to the web server.

The following example shows how to validate an entered email address. An email address must contain at least a '@' sign and a dot (.). Also, the '@' must not be the first character of the email address, and the last dot must at least be one character after the '@' sign.

Example

Try the following code for email validation.

```
<script type="text/javascript">  
  
<!--  
    function validateEmail()  
    {  
        var emailID = document.myForm.EMail.value;  
        atpos = emailID.indexOf("@");  
        dotpos = emailID.lastIndexOf(".");  
        if (atpos < 1 || ( dotpos - atpos < 2 ))  
        {  
            alert("Please enter correct email ID")  
            document.myForm.EMail.focus() ;  
            return false;  
        }  
        return( true );  
    }  
  
    //-->  
</script>
```

Simple AJAX Application:

AJAX is an acronym for **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not any new programming language, but simply a new technique for creating better, faster, and more interactive web applications. AJAX uses JavaScript to send and receive data between a web browser and a web server.

The AJAX technique makes web pages more responsive by exchanging data with the web server behind the scenes, instead of reloading an entire web page each time a user makes a change. The best example of AJAX implementation is Yahoo mails. Another example of AJAX implementation is populating the city and state based upon the zip entered in a TextBox or partial page update.

It uses asynchronous data transfer (HTTP requests) between the clients browser and the web server, allowing web pages to request small bits of information from the server instead of whole pages to be reloaded.

Bases of AJAX

AJAX is very simple technology based on the following open standards:

- **JavaScript**
- **XML**
- **HTML**
- **CSS**

AJAX stands for Asynchronous JavaScript and XML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS, and Java Script.

Ajax uses XHTML for content, CSS for presentation, along with Document Object

- Model and JavaScript for dynamic content display. Conventional web applications transmit information to and from the sever using
- synchronous requests. It means you fill out a form, hit submit, and get directed to a new page with new information from the server. With AJAX, when you hit submit, JavaScript will make a request to the server,
- interpret the results, and update the current screen. In the purest sense, the user would never know that anything was even transmitted to the server. XML is commonly used as the format for receiving server data, although any

- format, including plain text, can be used. AJAX is a web browser technology independent of web server software.
- A user can continue to use the application while the client program requests information from the server in the background. Intuitive and natural user interaction. Clicking is not required, mouse movement
- is a sufficient event trigger.
- Data-driven as opposed to page-driven

AJAX cannot work independently. It is used in combination with other technologies to create interactive webpages. JavaScript Loosely typed scripting language.

- JavaScript function is called when an event occurs in a page.
- Glue for the whole AJAX operation.
- DOM API for accessing and manipulating structured documents.
- Represents the structure of XML and HTML documents.
- CSS Allows for a clear separation of the presentation style from the content and may be changed programmatically by JavaScript.

XMLHttpRequest : JavaScript object that performs asynchronous interaction with the server.

About Simple Ajax Example

This example will present a form to the user and ask the user to enter his/her name. After entering the name user can press the "Say Hello" button. Then an Ajax call will send the user name on the server to a php file. The php file will return greeting message with the current server date and time. This example will show you how to make Ajax call to a server-side script and then get the data from the server.

Demo: Simple Ajax Example demo

The application will display the form. User can then enter then name and press the "Say Hello" button as shown below in the screen shot.

Simple Ajax Example

Enter your name and then press "Say Hello Button"

Enter your name:

Enter the name and press the Say Button. Application should send the user name on server and then display the response as shown below:

Simple Ajax Example

Enter your name and then press "Say Hello Button"

Enter your name:

Welcome Deepak!

Request received on: Tuesday Sep 07th, 2010, 12:09:24

Writing Simple Ajax Example

We have to write the required JavaScript code to make the server call (to call sayhello.php) by passing the user name. Here is the complete code of the simpleajaxexampledemo.html file:

```
<html>
<head>
<title>Simple Ajax Example</title>
<script language="Javascript">
function postRequest(strURL) {
    var xmlhttp;
    if (window.XMLHttpRequest) { // Mozilla, Safari, ...
        var xmlhttp = new XMLHttpRequest();
    }else if (window.ActiveXObject) { // IE
        var xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }
    xmlhttp.open('POST', strURL, true);
    xmlhttp.setRequestHeader
('Content-Type', 'application/x-www-form-urlencoded');
    xmlhttp.onreadystatechange = function() {
        if (xmlhttp.readyState == 4) {
            updatepage(xmlhttp.responseText);
        }
    }
    xmlhttp.send(strURL);
}
function updatepage(str){
    document.getElementById("result").innerHTML =
    "<font color='red' size='5'>" + str + "</font>";
}
function SayHello(){
    var usr=window.document.f1.username.value;
    var rnd = Math.random();
    var url="sayhello.php?id="+rnd+"&usr="+usr;
    postRequest(url);
}
```


CONTENT RESORUCE

TEXT BOOKS:

1. Web Technologies , Uttam K Roy, Oxford University Press
2. The Complte Reference PHP – Steven Holzer, Tata McGraw-Hill.
2. Web Programming , building internet applications, Chris Bates 2 nd edition, Wiley Dreamtech
3. Java Server Pages – Hans Bergsten, SPD O;Reilly.

Java Script, D.Flanagan, O,Reilly, SPD.

5. Beginning Web Programming – Jon Duckett WROX.
6. Programming world wide web, R.W. Sebesta, Fourth Edition, Pearson.
7. Internet and World Wide Web – How to program, Dietel and Nieto, Pearson.

WEB SITES:

1. https://www.w3schools.com/jquery/html_append.asp
2. https://www.w3schools.com/xml/xml_attributes.asp
3. https://www.freebsd.org/doc/en_US.ISO8859-1/books/fdp-primer/xml-primer-elements.html
4. <http://www.w3schoolas.com/history%20of%20php.php>
5. <https://www.w3schools.com/js/default.asp>
6. https://www.w3schools.com/js/js_ajax_intro.asp
7. <https://www.w3schools.com/sql/default.asp>
8. <https://www.javatpoint.com/session-tracking-in-servlets>
9. <https://www.tutorialspoint.com/servlets/servlets-session-tracking.htm>
10. https://www.tutorialspoint.com/jsp/jsp_implicit_objects.htm
11. <https://www.javatpoint.com/jsp-implicit-objects>
12. <https://beginnersbook.com/2013/11/jsp-implicit-objects/>
13. <http://www.cs.uofs.edu/~bi/eclipse/servlet-jdbc>
14. <http://www.java2s.com/Code/Java/Servlets/JDBCandServlet.htm>
15. https://www.tutorialspoint.com/xml/xml_tags.htm
16. <http://api.jquery.com/append/>
17. https://www.w3schools.com/jquery/html_append.asp