UNIT-4

INTRODUCTION TO JSP: The Anatomy 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 database in JSP. Client-side Scripting: Introduction to JavaScript, JavaScript language – declaring variables, scope of variables, functions. event handlers (onClick, onSubmit etc.), Document Object Model, Form validation.

JAVA SCRIPT

1.JavaScript language:

JavaScript scripting language, which facilitates a disciplined approach to designing computer programs that enhance the functionality and appearance of web pages. JavaScript serves two purposes- it introduces client-side scripting, which makes web pages more dynamic and interactive, and it provides the programming foundation for the more complex server side developments.

JavaScript contains a standard library of objects, like **Array**, **Date**, and **Math**, and a core set of language elements like **operators**, **control structures**, and **statements**.

- **Client-side:** It supplies objects to control a browser and its Document Object Model (DOM). Like if client-side extensions allow an application to place elements on an HTML form and respond to user events such as **mouse clicks**, **form input**, and **page navigation**.
- **Server-side:** It supplies objects relevant to running JavaScript on a server. Like if the server-side extensions allow an application to communicate with a database, and provide continuity of information from one invocation to another of the application, or perform file manipulations on a server.

2. JavaScript Variables:

❖ Declaring Javascript variables:

Like many other programming languages, JavaScript has variables. Variables can be thought of as named containers. You can place data into these containers and then refer to the data simply by naming the container.

Before you use a variable in a JavaScript program, you must declare it. Variables are declared with the **var** keyword as follows.

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

You can also declare multiple variables with the same **var** keyword as follows −

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

```
</script>
```

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

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

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

Note − Use the **var** keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.

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

3. JavaScript Variable Scope:

The scope of a variable is the region of your program in which it is defined. JavaScript variables have only two scopes.

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

Within the body of a function, a local variable takes precedence over a global variable with the same name. If you declare a local variable or function parameter with the same name as a global variable, you effectively hide the global variable. Take a look into the following example.

Ex: <html>

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

This produces the following result −

Output: local

4. JavaScript Variable Names:

While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name.S For example, **break** or **boolean** variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, **123test** is an invalid variable name but **_123test** is a valid one.
- JavaScript variable names are case-sensitive. For example, **Name** and **name** are two different variables.

Ex:
```
<html>
      <body>
        <script language=”javacsript”>
            var x = 10;
            var y = 20;
           var z=x+y;
           document.writeln(“<h1>”+z+”</h1>”);
         </script>
      </body>
  </html>
```

Ex: Global variable:
```
 <html>
     <body>
       <script language=”javascript”>
         var data=200;//gloabal variable
        function a(){
          document.writeln(data);
        }
        function b(){
         document.writeln(data);
        }
       a();//calling JavaScript function
       b();
  </script>
 </body>
</html>
```

Output: 200 200

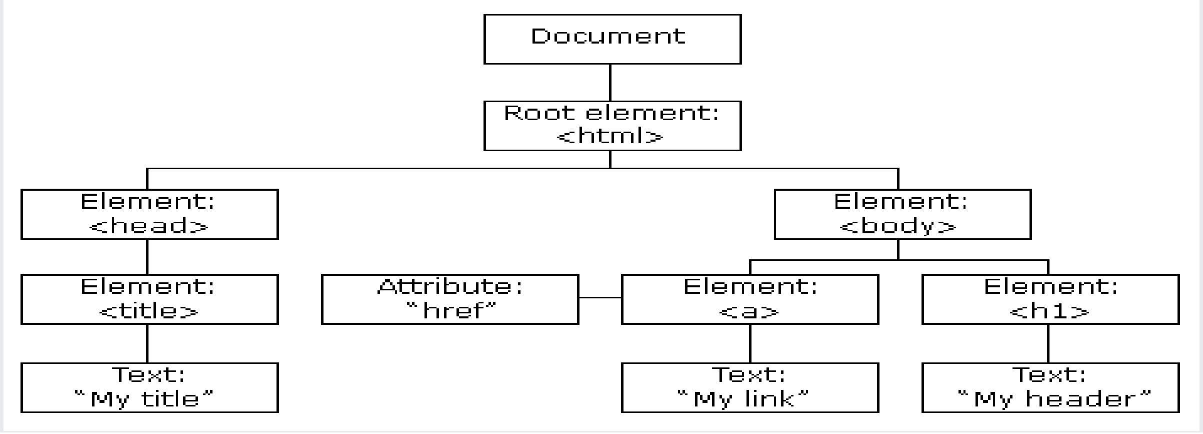
5. JavaScript - Document Object Model or DOM:

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page. The DOM is a W3C (World Wide Web Consortium) standard.The DOM defines a standard for accessing documents:

"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."

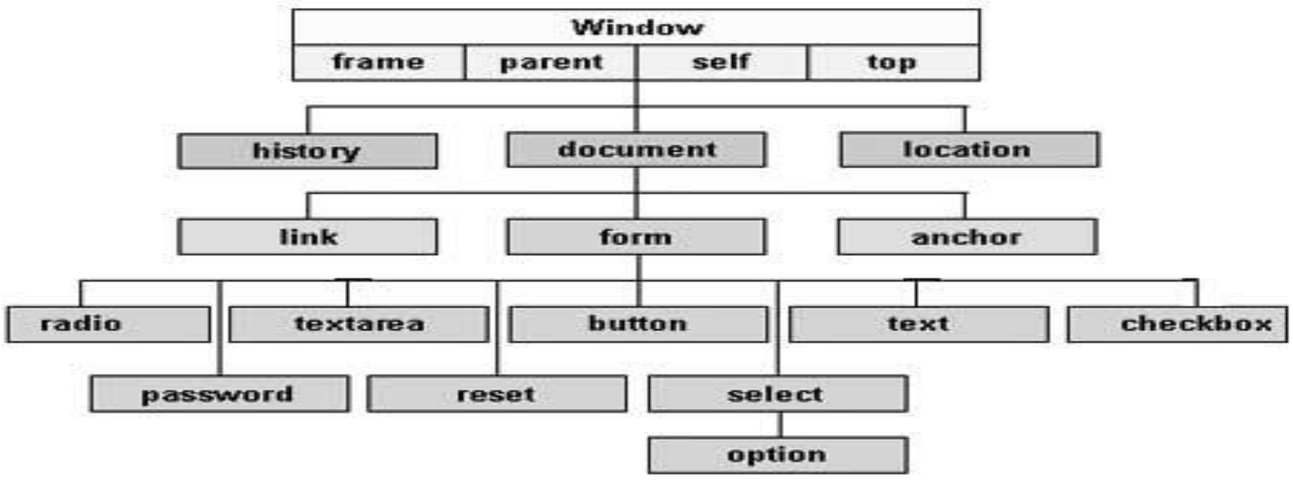
The **HTML DOM** model is constructed as a tree of **Objects**:

The HTML DOM Tree of Objects



The way a document content is accessed and modified is called the **Document Object Model**, or **DOM**. The Objects are organized in a hierarchy. This hierarchical structure applies to the organization of objects in a Web document.

- **Window object** − Top of the hierarchy. It is the outmost element of the object hierarchy.
- **Document object** − Each HTML document that gets loaded into a window becomes a document object. The document contains the contents of the page.
- **Form object** − Everything enclosed in the <form>...</form> tags sets the form object.
- **Form control elements** − The form object contains all the elements defined for that object such as text fields, buttons, radio buttons, and checkboxes.



The HTML DOM is a standard **object** model and **programming interface** for HTML. It defines:

- The HTML elements as **objects**
- The **properties** of all HTML elements
- The **methods** to access all HTML elements
- The **events** for all HTML elements

In other words: **The HTML DOM is a standard for how to get, change, add, or delete HTML elements.**

❖ **JavaScript - HTML DOM Methods**

HTML DOM methods are **actions** you can perform (on HTML Elements).
HTML DOM properties are **values** (of HTML Elements) that you can set or change.

❖ **The DOM Programming Interface:**

- The HTML DOM can be accessed with JavaScript (and with other programming languages).
- In the DOM, all HTML elements are defined as **objects**.
- The programming interface is the properties and methods of each object.

A **property** is a value that you can get or set (like changing the content of an HTML element).

A **method** is an action you can do (like add or deleting an HTML element).

Example:

The following example changes the content (the innerHTML) of the <p> element with id="demo":

Example

```
<html>
<body>

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

<script>
document.getElementById("demo").innerHTML = "HelloWorld!";
</script>

</body>
</html>
```

Output: HelloWorld

In the example above, getElementById is a **method**, while innerHTML is a **property**.

➢ **The getElementById Method**

The most common way to access an HTML element is to use the id of the element.In the example above the getElementById method used id="demo" to find the element.

➢ **The innerHTML Property**

The easiest way to get the content of an element is by using the innerHTML property.The innerHTML property is useful for getting or replacing the content of HTML elements.The innerHTML property can be used to get or change any HTML element, including <html> and <body>.

6. JavaScript - Form Validation:

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.

❖ **Basic Validation :**

It is important to validate the form submitted by the user because it can have inappropriate values. So, validation is must to authenticate user.

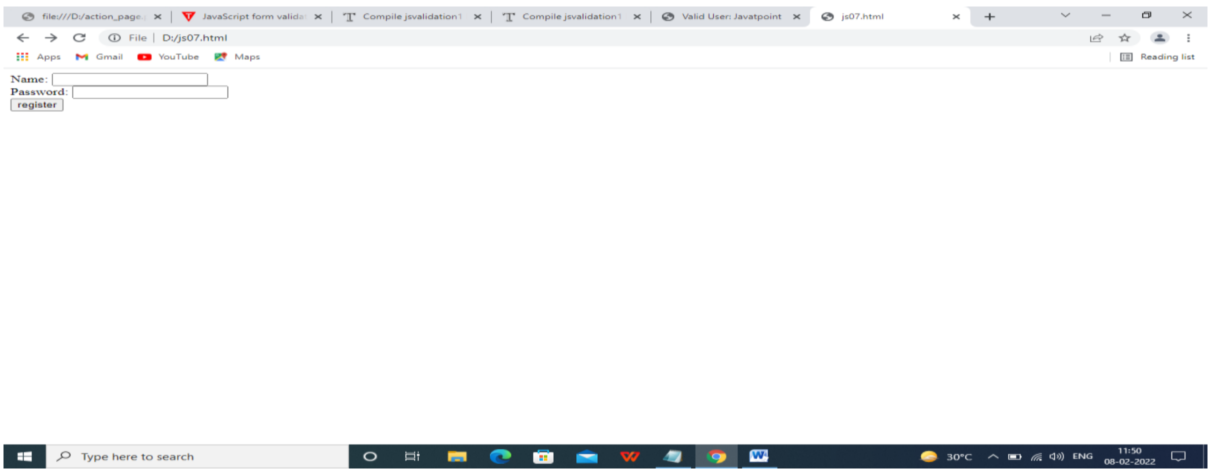
JavaScript provides facility to validate the form on the client-side so data processing will be faster than server-side validation. Most of the web developers prefer JavaScript form validation.

Through JavaScript, we can validate name, password, email, date, mobile numbers and more fields.We will take an example to understand the process of validation. Here is a simple form in html format.

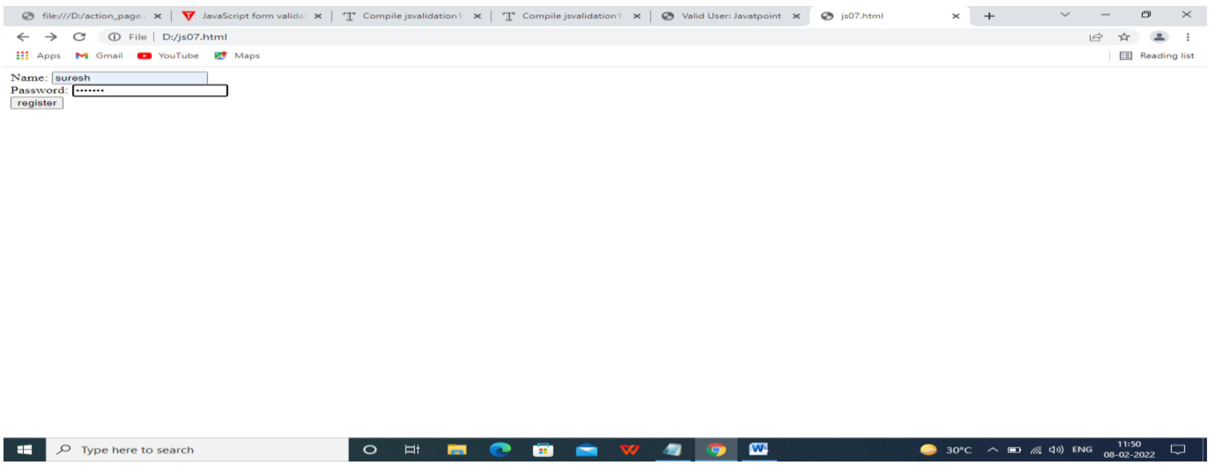
Ex:

```
<html>

<body><script language="javascript">

function validateform(){

var name=document.myform.name.value;

var password=document.myform.password.value;

if (name==null || name==""){

  alert("Name can't be blank");

  return false;

}else if(password.length<6){

  alert("Password must be at least 6 characters long.");

  return false;
  }  }

</script>

<body>

<form name="myform" method="post"
action="http://www.javatpoint.com/javascriptpages/valid.jsp"onsubmit="returnvalidateform
()" >

Name: <input type="text" name="name"><br/>

Password: <input type="password" name="password"><br/>

<input type="submit" value="register">

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

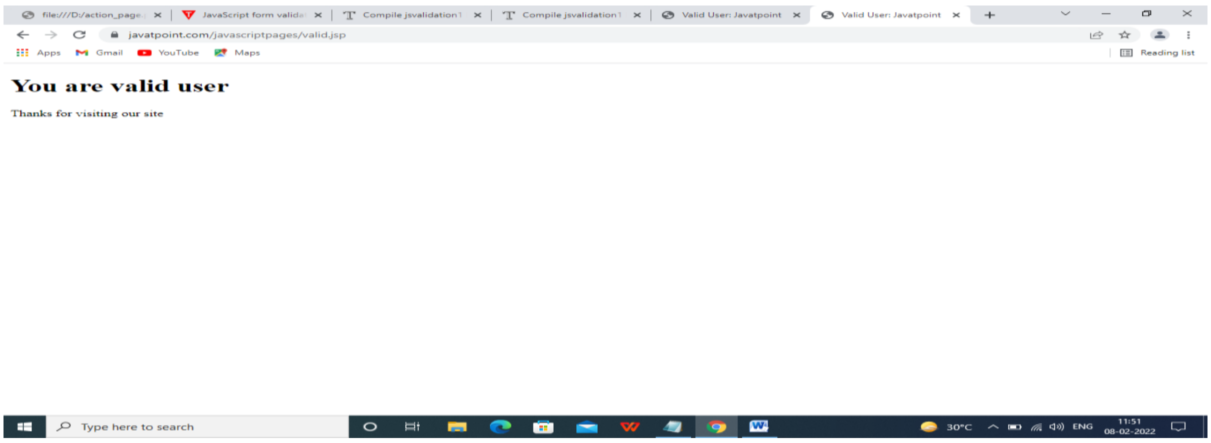
Output:



```
Name: 
Password: 
register
```



```
Name: suresh
Password: •••••••
register
```


You are valid user

Thanks for visiting our site

❖ Data Format Validation:

Data validation is the process of ensuring that user input is clean, correct, and useful.

Typical validation tasks are:

- has the user filled in all required fields?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Most often, the purpose of data validation is to ensure correct user input. Validation can be defined by many different methods, and deployed in many different ways.

- **Server side validation** is performed by a web server, after input has been sent to the server.
- **Client side validation** is performed by a web browser, before input is sent to a web server.

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>
```

7. Javascript EVENT HANDLERS:

JavaScript's interaction with HTML is handled through events that occur when the user or the browser manipulates a page.When the page loads, it is called an event. 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.

❖ onclick Event Type:

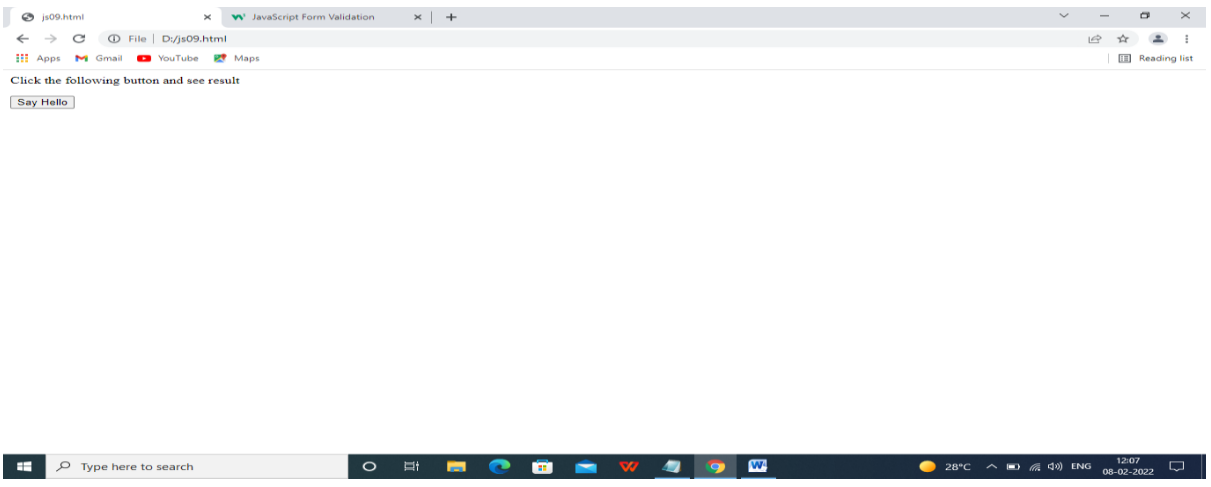
This is the most frequently used event type which occurs when a user clicks the left button of his mouse. You can put your validation, warning etc., against this event type. One of the most common event is onclick. When the user clicks a specific item with the mouse, the onclick event fires.

Example:

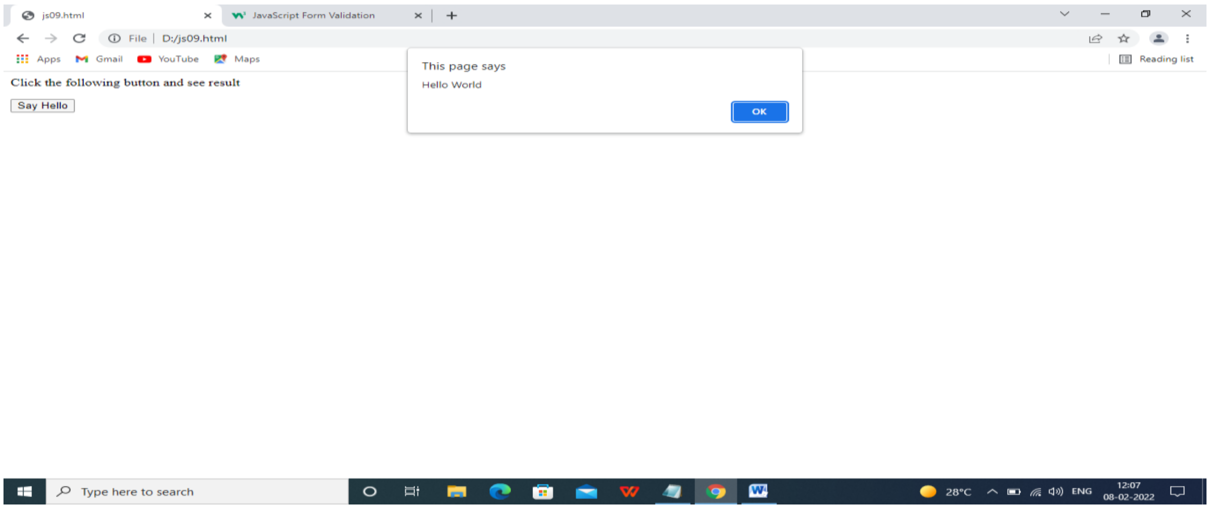
Try the following example.

```
<html>
  <head>
    <script language= "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:


Click the following button and see result

If you click on the sayHello button . The below message displays.



Click the following button and see result

Say Hello

This page says

Hello World

OK

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

associates the script directly with the input element. Inline scripting like this is often used to pass a value associated with the clicked element to an event handler.

❖ **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:

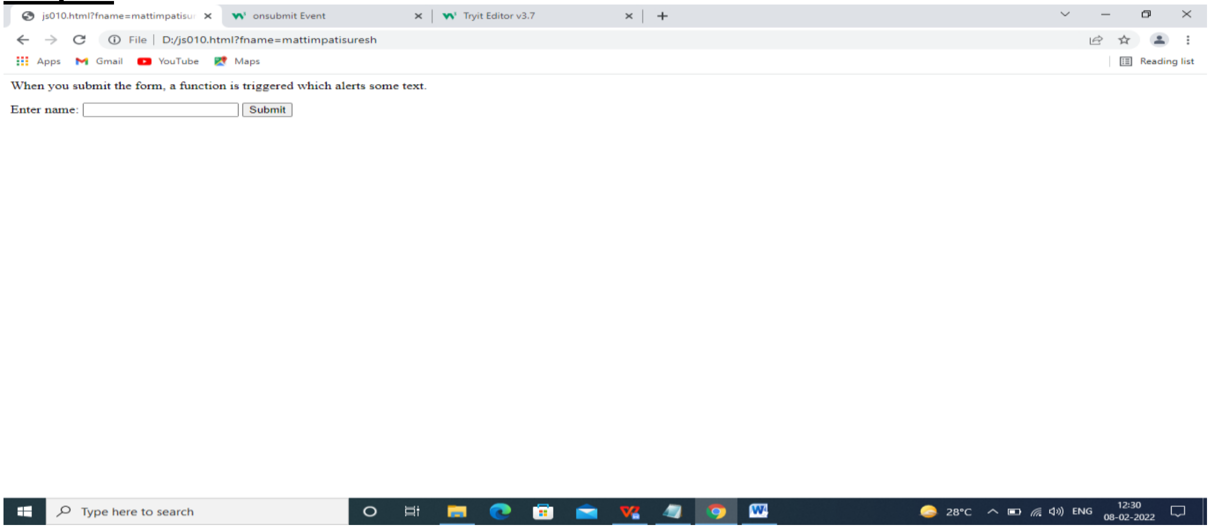
The following example shows how to use onsubmit. Here we are calling a **myFunction()** function before submitting a form data to the webserver. If **myFunction()** function returns the message " The form was submitted", otherwise it will not submit the data.

Try the following example.

Program:

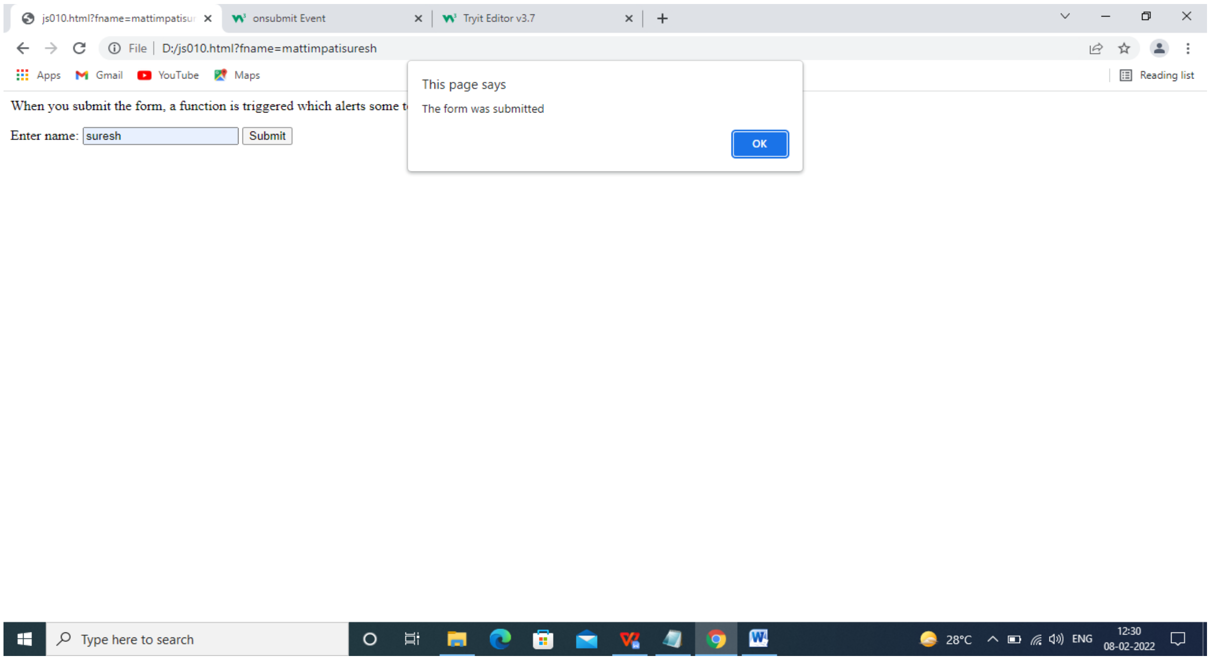
```
<html>
   <body>
         <p>When you submit the form, a function is triggered which alerts some text.</p>
         <form action="" onsubmit="myFunction()">
          Enter name: <input type="text" name="fname">
          <input type="submit" value="Submit">
         </form>
<script language="javascript">
function myFunction() {
  alert("The form was submitted");
}
</script>

</body>
</html>
```

Output:



When you submit the form, a function is triggered which alerts some text.

Enter name: Submit



When you submit the form, a function is triggered which alerts some t

Enter name: suresh Submit

This page says

The form was submitted

OK

8. **Javascript Functions**:

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it). A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. It helps programmers in writing modular codes. Functions allow a programmer to divide a big program into a number of small and manageable functions.

❖ **JavaScript Function Syntax:**

A JavaScript function is defined with the function keyword, followed by a **name**, followed by parentheses **()**.Function names can contain letters, digits, underscores, and dollar signs (same rules as variables).

The parentheses may include parameter names separated by commas:(*parameter1, parameter2, ...*)**The code to be executed, by the function, is placed inside curly brackets: {}.** There is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Syntax:

```
function name(parameter1, parameter2, parameter3)
{
  // codetobeexecuted
}
```

- Function **parameters** are listed inside the parentheses () in the function definition.
- Function **arguments** are the **values** received by the function when it is invoked.
- Inside the function, the arguments (the parameters) behave as local variables.

Ex:

```
<html>
<body>
<h2>JavaScript Functions</h2>
<p>This example calls a function which performs a calculation and returns the result:</p>
<script language="javascript">
var x = myFunction(4, 3);
function myFunction(a, b)
{
  return a * b;
}
document.writeln(x);
</script>
</body>
</html>
```

Output:

```
JavaScript Functions

This example calls a function which performs a calculation and returns the result:

12
```

- **The return Statement:**

A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

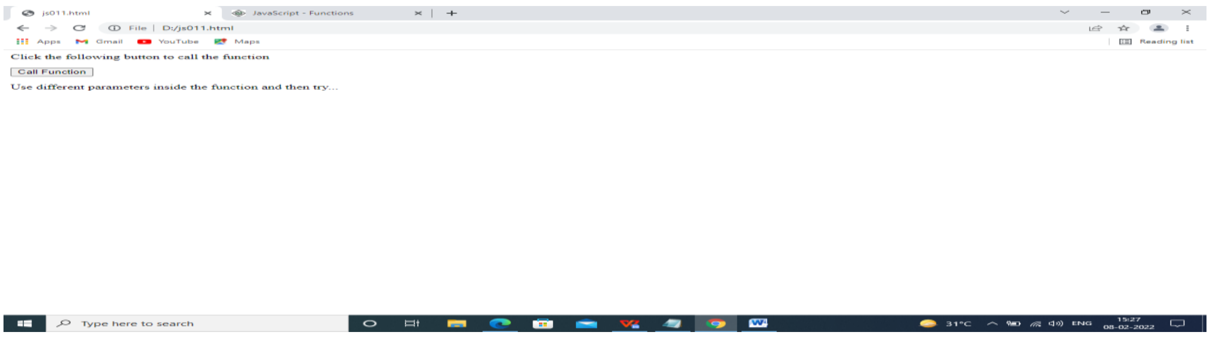
For example, you can pass two numbers in a function and then you can expect the function to return their multiplication in your calling program.

Example:

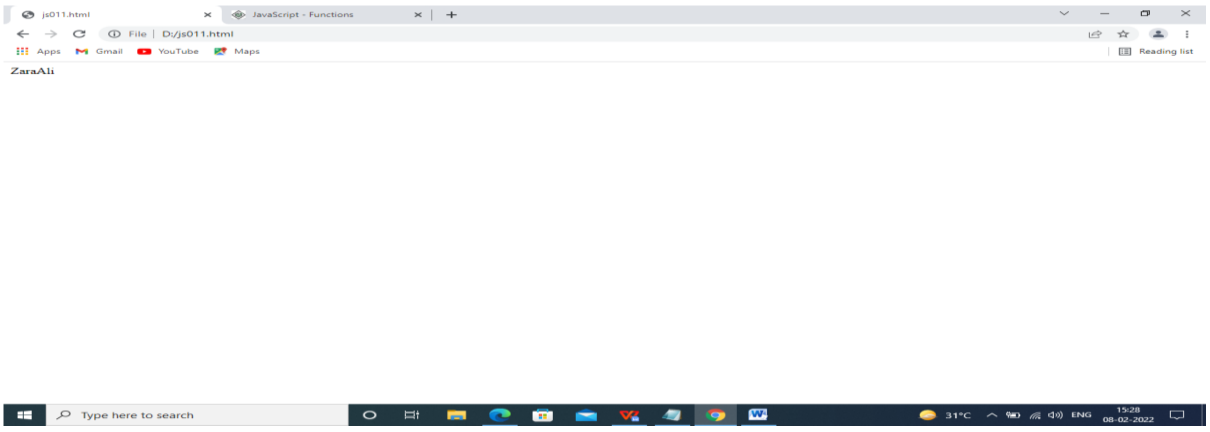
Try the following example. It defines a function that takes two parameters and concatenates them before returning the resultant in the calling program.

```
<html>
  <head>
    <script type = "text/javascript">
      function concatenate(first, last) {
        var full;
        full = first + last;
        return full;
      }
      function secondFunction() {
        var result;
        result = concatenate('Zara', 'Ali');
        document.write (result );
      }
    </script>
  </head>
   <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type = "button" onclick = "secondFunction()" value = "Call Function">
    </form>
    <p>Use different parameters inside the function and then try...</p>
 </body></html>
```

Output



```
Click the following button to call the function
[Call Function]
Use different parameters inside the function and then try...
```


ZaraAli

INTRODUCTION TO JSP

The Servlet technology and Java Server Pages (JSP) are the two main technologies for developing java Web applications. When first introduced by Sun Microsystems in 1996, the Servlet technology was considered superior to the reigning Common Gateway Interface (CGI) because servlets stay in memory after they service the first requests. Subsequent requests for the same servlet do not require instantiation of the servlet‘s class therefore enabling better response time.

Servlets has several difficulties to the web developer:

1. The code for a servlet becomes difficult to understand for the programmer.
2. The HTML content of web page is difficult if not impossible for a web designer to understand or design.
3. This is hard to program and even small changes in the presentation, such as the page‘s background color, will require the servlet to be recompiled. Any changes in the HTML content require the rebuilding of the whole servlet.
4. It's hard to take advantage of web-page development tools when designing the application interface. If such tools are used to develop the web page layout, the generated HTML must then be manually embedded into the servlet code, a process which is time consuming, error prone, and extremely boring.

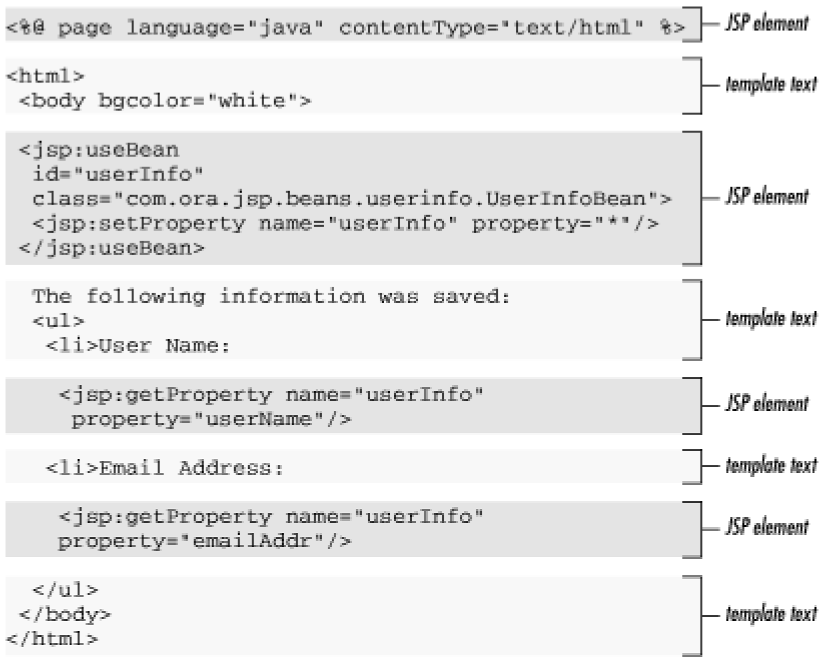
JSP solves these problems by giving a way to include java code into an HTML page using scriptlets. This way the HTML code remains intact and easily accessible to web designers, but the page can sill perform its task.

In late 1999, Sun Microsystems added a new element to the collection of Enterprise Java tools: JavaServer Pages (JSP). JavaServer Pages are built on top of Java servlets and designed to increase the efficiency in which programmers, and even nonprogrammers, can create web content.

Finally, The JSP is a server side technology. It is used to design web applications in order to generate dynamic response. For developing single page web applications, we use only JSP technology. For developing medium and large scale web applications, we use both servlet and JSP technology.

2. **The Anatomy of a JSP Page or JSP CODE:**

A JSP page is simply a regular web page with JSP elements for generating the parts of the page that differ for each request, as shown in below figure:



```
<%@ page language="java" contentType="text/html" %>
<html>
 <body bgcolor="white">
  <jsp:useBean
   id="userInfo"
   class="com.ora.jsp.beans.userinfo.UserInfoBean">
   <jsp:setProperty name="userInfo" property="*"/>
  </jsp:useBean>
   The following information was saved:
   <ul>
    <li>User Name:
     <jsp:getProperty name="userInfo"
      property="userName"/>
    <li>Email Address:
     <jsp:getProperty name="userInfo"
     property="emailAddr"/>
   </ul>
  </body>
</html>
```

Everything in the page that is not a JSP element is called *template text* . Template text can really be any text: HTML, WML, XML, or even plain text. Since HTML is by far the most common web page language in use today, most of the descriptions and examples in this book are HTML-based, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser.

When a JSP page request is processed, the template text and the dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

%@page import =”java util.Date;”% → JSP Element

<html>
<head>
<title>
First JSP program</title>
</head> → Template text

<body>
<%
out.println(“Hello MCA students”);
 out.println(“<br><br>”); → JSP Element

```
out.println("welcome to JSP programming");
out.println("new Date().tpoString());
%>
<center> Have a nice day</center>
<br><br>
</body>
</html>
```

$\longrightarrow$ Template texrt

When JSP request gets processed template text and JSP elements are merged together and sent to the browser as response.

<u>3. JSP Processing:</u>
Once you have a JSP capable web-server or application server, you need to know the following information about it:
• Where to place the files
• How to access the files from your browser (with an http: prefix, not as file:)
You should be able to create a simple file, such as

```
<HTML>
<BODY>
Hello, world
</BODY>
</HTML>
```

Know where to place this file and how to see it in your browser with an http:// prefix. Since this step is different for each web-server, you would need to see the web-server documentation to find out how this is done. Once you have completed this step, proceed to the next.

- **<u>Your first JSP</u>**

JSP simply puts Java inside HTML pages. You can take any existing HTML page and change its extension to ".jsp" instead of ".html". In fact, this is the perfect exercise for your first JSP. Take the HTML file you used in the previous exercise. Change its extension from ".html" to ".jsp". Now load the new file, with the ".jsp" extension, in your browser.

You will see the same output, but it will take longer! But only the first time. If you reload it again, it will load normally.

What is happening behind the scenes is that your JSP is being turned into a Java file, compiled and loaded. This compilation only happens once, so after the first load, the file doesn't take long to load anymore. (But everytime you change the JSP file, it will be recompiled again.)

Of course, it is not very useful to just write HTML pages with a .jsp extension! We now proceed to see what makes JSP so useful. Adding dynamic content via expressions

As we saw in the previous section, any HTML file can be turned into a JSP file by changing its extension to .jsp. Of course, what makes JSP useful is the ability to embed Java. Put the following text in a file with .jsp extension (let us call it hello.jsp), place it in your JSP directory, and view it in a browser.

```
<HTML>
<BODY>
Hello! The time is now
</BODY>
</HTML>
```

Notice that each time you reload the page in the browser, it comes up with the current time. The character sequences

<%= and%> enclose Java expressions, which are evaluated at run time.

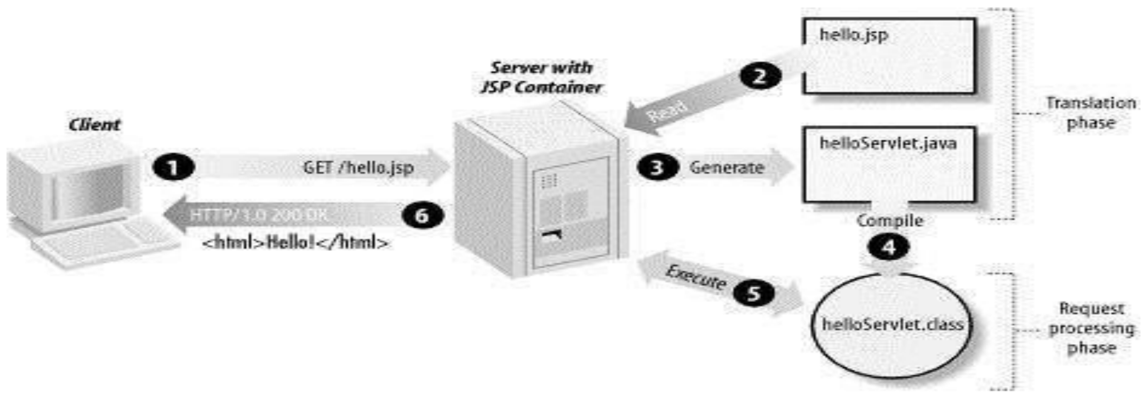
This is what makes it possible to use JSP to generate dynamic HTML pages that change in response to user actions or vary from user to user.

❖ JSP processing:

The following steps explain how the web server creates the Webpage using JSP −

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behaviour of the page.
- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is further passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram −



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

4. JSP DECLARATIONS:

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. So it doesn't get memory at each request.

Syntax of JSP declaration tag

The syntax of the declaration tag is as follows:

```
<%!  field or method declaration %>
```

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>
```

5. 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
- isELIgnored
- 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 i
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".

Example of contentType attribute

```
<html>
<body>
<%@ page contentType=application/msword %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

---

3)extends

The extends attribute defines the parent class that will be inherited by the generated servlet.It is rarely used.

4)info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

Example of info attribute

```
<html>
<body>
<%@ page info="composed by Sonoo Jaiswal" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

The web container will create a method getServletInfo() in the resulting servlet.For example:

```
public String getServletInfo() {
  return "composed by Sonoo Jaiswal";
}
```

5)buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page.The default size of the buffer is 8Kb.

Example of buffer attribute

```
<html>
<body>
<%@ page buffer="16kb" %>
Today is: <%= new java.util.Date() %>
</body>
</html>
```

6)language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

7)isELIgnored

We can ignore the Expression Language (EL) in jsp by the isELIgnored attribute. By default its value is fal
Expression Language is enabled by default. We see Expression Language later.

```
<%@ page isELIgnored="true" %>//Now EL will be ignored
```

8)isThreadSafe

Servlet and JSP both are multithreaded.If you want to control this behaviour of JSP page
can use isThreadSafe attribute of page directive.The value of isThreadSafe value is true.If you make it fals
web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a re
before passing another request to it.If you make the value of isThreadSafe attribute like:

<%@ page isThreadSafe="false" %>

The web container in such a case, will generate the servlet as:

```
public class SimplePage_jsp extends HttpJspBase
  implements SingleThreadModel{
.......
}
```

9)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>
```

10)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 occured!<br/>
The exception is: <%= exception %>
</body>
</html>
```

❖ 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>
```

Note: The include directive includes the original content, so the actual page size grows at runtime.

❖ JSP Taglib directive:

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags. In the custom tag section we will use this tag so it will be better to learn it in custom tag.

Syntax JSP Taglib directive

```
<%@ taglib uri="uriofthetaglibrary" prefix="prefixoftaglibrary" %>
```

Example of JSP Taglib directive

In this example, we are using our tag named currentDate. To use this tag we must specify the taglib directive so the container may get information about the tag.

```
<html>
<body>
  <%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
  <mytag:currentDate/>
  </body>
</html>
```

6. 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

```
<%=  statement %>
```

Example of JSP expression tag

In this example of jsp expression tag, we are simply displaying a welcome message.

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Note: Do not end your statement with semicolon in case of expression tag.

Example of JSP expression tag that prints current time

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
<html>
<body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

Example of JSP expression tag that prints the user name:

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

```
<html>
<body>
<form action="welcome.jsp">
<input type="text" name="uname"><br/>
<input type="submit" value="go">
</form>
</body>
</html>
```

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
</body>
</html>
```

7.JSP Implicit Objects:

There are **9 jsp implicit objects**. These objects are *created by the web container* that are available to all the jsp pages.The available implicit objects are out, request, config, session, application etc.

A list of the 9 implicit objects is given below:

1. JSP out implicit object:

Object	Type
Out	JspWriter
Request	HttpServletRequest
Response	HttpServletResponse
Config	ServletConfig
Application	ServletContext
Session	HttpSession
pageContext	PageContext
Page	Object
Exception	Throwable

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter. In case of servlet you need to write:

```
PrintWriter out=response.getWriter();
```

But in JSP, you don't need to write this code.

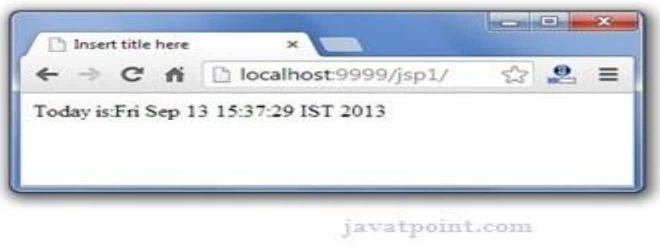
Example of out implicit object

In this example we are simply displaying date and time.

index.jsp

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

Output



```
Today is:Fri Sep 13 15:37:29 IST 2013
```

2. **JSP request implicit object:**

The **JSP request** is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc.It can also be used to set, get and remove attributes from the jsp request scope.

Let's see the simple example of request implicit object where we are printing the name of the user with welcome message.

Example of JSP request implicit object

index.html

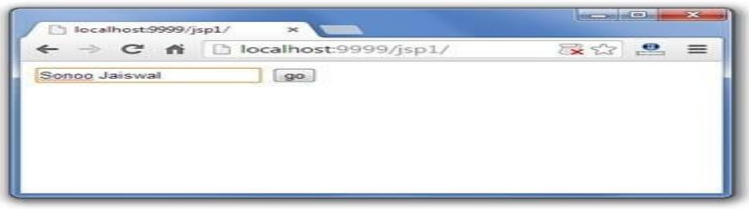
```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
```

```
</form>
```

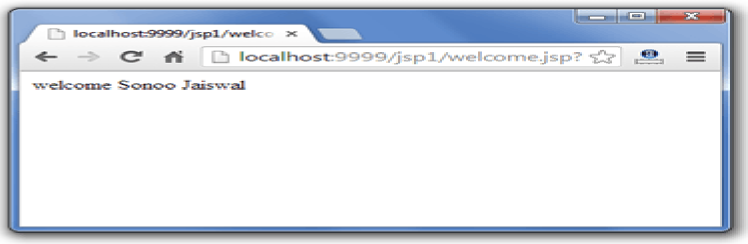
welcome.jsp

```
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
```

Output



```
Sonoo Jaiswal
```



```
welcome Sonoo Jaiswal
```

3) JSP response implicit object:

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request.

It can be used to add or manipulate response such as redirect response to another resource, send error etc.

Let's see the example of response implicit object where we are redirecting the response to the Google.

Example of response implicit object

index.html

```
<form action="welcome.jsp">
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```

Output

4.<u>JSP config implicit object:</u>

In JSP, config is an implicit object of type *ServletConfig*. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page. Generally, it is used to get initialization parameter from the web.xml file.

<u>5) JSP application implicit object</u>

In JSP, application is an implicit object of type *ServletContext*. The instance of ServletContext is created only once by the web container when application or project is deployed on the server.

This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope. This initialization parameter can be used by all jsp pages.

<u>6) session implicit object</u>

In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.

<u>7) pageContext implicit object</u>

In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes:

- page
- request
- session
- application

In JSP, page scope is the default scope.

8) page implicit object:

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as:

Object page=this;

For using this object it must be cast to Servlet type.For example:

```
<% (HttpServlet)page.log("message"); %>
```

Since, it is of type Object it is less used because you can use this object directly in jsp.For example:

```
<% this.log("message"); %>
```

9) exception implicit object:

In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be used to print the exception. But it can only be used in error pages.It is better to learn it after page directive.

Example of exception implicit object:

error.jsp

```
<%@ page isErrorPage="true" %>
<html>
<body>

Sorry following exception occured:<%= exception %>

</body>
</html>
```

9. JSP code snippets:

A JSP code snippet is a code sample that shows you how to add WebSphere Commerce functionality to the 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 the 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.

10. Using Beans in JSP Pages:

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.
Following are the unique characteristics that distinguish a JavaBean from other Java classes

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "**getter**" and "**setter**" methods for the properties.

❖ JavaBeans Properties:

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be **read, write, read only**, or **write only**. JavaBean properties are accessed through two methods in the JavaBean's implementation class −

S.No.	Method & Description
1	get**PropertyName**() For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property. This method is called accessor.
2	set**PropertyName**() For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

JavaBeans Example

Consider a student class with few properties −

```
package com.tutorialspoint;

public class StudentsBean implements java.io.Serializable {
   private String firstName = null;
   private String lastName = null;
   private int age = 0;

   public StudentsBean() {
   }
   public String getFirstName(){
      return firstName;
   }
   public String getLastName(){
      return lastName;
   }
   public int getAge(){
      return age;
   }
   public void setFirstName(String firstName){
      this.firstName = firstName;
   }
   public void setLastName(String lastName){
      this.lastName = lastName;
   }
   public void setAge(Integer age){
      this.age = age;
   }
}
```

- ❖ **Accessing JavaBeans**

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows −

<jsp:useBean id = "bean's name" scope = "bean's scope" typeSpec/>

Here values for the scope attribute can be a **page, request, session** or **application based** on your requirement. The value of the **id** attribute may be any value as a long as it is a unique name among other **useBean declarations** in the same JSP.

Following example shows how to use the useBean action −

```
<html>
   <head>
      <title>useBean Example</title>
   </head>

   <body>
      <jsp:useBean id = "date" class = "java.util.Date" />
      <p>The date/time is <%= date %>
   </body>
```

```
</html>
```

You will receive the following result − −

The date/time is Thu Sep 30 11:18:11 GST 2010

❖ **Accessing JavaBeans Properties:**

Along with **<jsp:useBean...>** action, you can use the **<jsp:getProperty/>** action to access the get methods and the **<jsp:setProperty/>** action to access the set methods. Here is the full syntax −

```
<jsp:useBean id = "id" class = "bean's class" scope = "bean's scope">
  <jsp:setProperty name = "bean's id" property = "property name"
    value = "value"/>
  <jsp:getProperty name = "bean's id" property = "property name"/>
  ..........
</jsp:useBean>
```

The name attribute references the id of a JavaBean previously introduced to the JSP by the useBean action. The property attribute is the name of the **get** or the **set** methods that should be invoked.

Following example shows how to access the data using the above syntax −

```
<html>
  <head>
    <title>get and set properties Example</title>
  </head>

  <body>
    <jsp:useBean id = "students" class = "com.tutorialspoint.StudentsBean">
      <jsp:setProperty name = "students" property = "firstName" value = "Zara"/>
      <jsp:setProperty name = "students" property = "lastName" value = "Ali"/>
      <jsp:setProperty name = "students" property = "age" value = "10"/>
    </jsp:useBean>

    <p>Student First Name:
      <jsp:getProperty name = "students" property = "firstName"/>
    </p>

    <p>Student Last Name:
      <jsp:getProperty name = "students" property = "lastName"/>
    </p>

    <p>Student Age:
      <jsp:getProperty name = "students" property = "age"/>
    </p>

  </body>
</html>
```

Let us make the **StudentsBean.class** available in CLASSPATH. Access the above JSP. the following result will be displayed −

Student First Name: Zara

Student Last Name: Ali

Student Age: 10

11. JSP - Session Tracking:

Here we discuss session tracking in JSP. HTTP is a "stateless" protocol which means each time a client retrieves a Webpage, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

❖ Maintaining Session Between Web Client And Server:

Let us now discuss a few options to maintain the session between the Web Client and the Web Server −

- **Cookies**

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

- **Hidden Form Fields**

A web server can send a hidden HTML form field along with a unique session ID as follows

<input type = "hidden" name = "sessionid" value = "12345">

This entry means that, when the form is submitted, the specified name and value are automatically included in the **GET** or the **POST** data. Each time the web browser sends the request back, the **session_id** value can be used to keep the track of different web browsers.

This can be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

- **URL Rewriting**

You can append some extra data at the end of each URL. This data identifies the session; the server can associate that session identifier with the data it has stored about that session.

For example, with **http://tutorialspoint.com/file.htm;sessionid=12345**, the session identifier is attached as **sessionid = 12345** which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and works for the browsers when they don't support cookies. The drawback here is that you will have to generate every URL dynamically to assign a session ID though page is a simple static HTML page.

- **The session Object**

Apart from the above mentioned options, JSP makes use of the servlet provided HttpSession Interface. This interface provides a way to identify a user across.

- a one page request or
- visit to a website or
- store information about that user

By default, JSPs have session tracking enabled and a new HttpSession object is instantiated for each new client automatically. Disabling session tracking requires explicitly turning it off by setting the page directive session attribute to false as follows −
<%@ page session = "false" %>

The JSP engine exposes the HttpSession object to the JSP author through the implicit **session** object. Since **session** object is already provided to the JSP programmer, the programmer can immediately begin storing and retrieving data from the object without any initialization or **getSession()**.

Here is a summary of important methods available through the session object −

S.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last time the client sent a request associated with the this session, as the number of milliseconds since midnight January 1, 1970 GMT.
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval, in seconds, that the servlet container will keep this session open between client accesses.

- **Session Tracking Example**

This example describes how to use the HttpSession object to find out the creation time and the last-accessed time for a session. We would associate a new session with the request if one does not already exist.

```
<%@ page import = "java.io.*,java.util.*" %>
<%
   // Get session creation time.
   Date createTime = new Date(session.getCreationTime());

   // Get last access time of this Webpage.
   Date lastAccessTime = new Date(session.getLastAccessedTime());

   String title = "Welcome Back to my website";
```

```
  Integer visitCount = new Integer(0);
  String visitCountKey = new String("visitCount");
  String userIDKey = new String("userID");
  String userID = new String("ABCD");

  // Check if this is new comer on your Webpage.
  if (session.isNew() ){
    title = "Welcome to my website";
    session.setAttribute(userIDKey, userID);
    session.setAttribute(visitCountKey,  visitCount);
  }
  visitCount = (Integer)session.getAttribute(visitCountKey);
  visitCount = visitCount + 1;
  userID = (String)session.getAttribute(userIDKey);
  session.setAttribute(visitCountKey,  visitCount);
%>

<html>
  <head>
    <title>Session Tracking</title>
  </head>

  <body>
    <center>
      <h1>Session Tracking</h1>
    </center>

    <table border = "1" align = "center">
      <tr bgcolor = "#949494">
        <th>Session info</th>
        <th>Value</th>
      </tr>
      <tr>
        <td>id</td>
        <td><% out.print( session.getId()); %></td>
      </tr>
      <tr>
        <td>Creation Time</td>
        <td><% out.print(createTime); %></td>
      </tr>
      <tr>
        <td>Time of Last Access</td>
        <td><% out.print(lastAccessTime); %></td>
      </tr>
      <tr>
        <td>User ID</td>
        <td><% out.print(userID); %></td>
      </tr>
      <tr>
        <td>Number of visits</td>
        <td><% out.print(visitCount); %></td>
      </tr>
    </table>

  </body>
</html>
```

Now put the above code in **main.jsp** and try to access ***http://localhost:8080/main.jsp***. Once you run the URL, you will receive the following result –

Welcome to my website

Session Information

Session info	**value**
Id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	0

Now try to run the same JSP for the second time, you will receive the following result.

Welcome Back to my website

Session Information

info type	**value**
Id	0AE3EC93FF44E3C525B4351B77ABB2D5
Creation Time	Tue Jun 08 17:26:40 GMT+04:00 2010
Time of Last Access	Tue Jun 08 17:26:40 GMT+04:00 2010
User ID	ABCD
Number of visits	1

12. <u>connecting to database in JSP:</u>

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

To start with basic concept, let us create a table and create a few records in that table as follows −

❖ **Create Table:**

To create the **Employees** table in the EMP database, use the following steps −

Step 1

Open a **Command Prompt** and change to the installation directory as follows −

```
C:\>
C:\>cd Program Files\MySQL\bin
C:\Program Files\MySQL\bin>
```

Step 2

Login to the database as follows −

```
C:\Program Files\MySQL\bin>mysql -u root -p
Enter password: ********
mysql>
```

Step 3

Create the **Employee** table in the **TEST** database as follows − −

```
mysql> use TEST;
mysql> create table Employees
  (
    id int not null,
    age int not null,
    first varchar (255),
    last varchar (255)
  );
Query OK, 0 rows affected (0.08 sec)
mysql>
```

❖ **Create Data Records:**

Let us now create a few records in the **Employee** table as follows − −

```
mysql> INSERT INTO Employees VALUES (100, 18, 'Zara', 'Ali');
Query OK, 1 row affected (0.05 sec)

mysql> INSERT INTO Employees VALUES (101, 25, 'Mahnaz', 'Fatma');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO Employees VALUES (102, 30, 'Zaid', 'Khan');
Query OK, 1 row affected (0.00 sec)
```

```
mysql> INSERT INTO Employees VALUES (103, 28, 'Sumit', 'Mittal');
Query OK, 1 row affected (0.00 sec)

mysql>
```

- **<u>SELECT Operation:</u>**

Following example shows how we can execute the **SQL SELECT** 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>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

❖ **DELETE Operation:**

Following example shows how we can execute the **SQL DELETE** 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>DELETE Operation</title>
  </head>

  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TEST"
      user = "root" password = "pass123"/>

    <c:set var = "empId" value = "103"/>
    <sql:update dataSource = "${snapshot}" var = "count">
      DELETE FROM Employees WHERE Id = ?
      <sql:param value = "${empId}" />
    </sql:update>

    <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>

    <table border = "1" width = "100%">
      <tr>
        <th>Emp ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
      </tr>

      <c:forEach var = "row" items = "${result.rows}">
        <tr>
          <td><c:out value = "${row.id}"/></td>
          <td><c:out value = "${row.first}"/></td>
          <td><c:out value = "${row.last}"/></td>
          <td><c:out value = "${row.age}"/></td>
        </tr>
      </c:forEach>
    </table>

  </body>
</html>
```

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

- **UPDATE Operation:** Following example shows how we can execute the SQL UPDATE 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>DELETE Operation</title>
  </head>

  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TEST"
      user = "root" password = "pass123"/>

    <c:set var = "empId" value = "102"/>

    <sql:update dataSource = "${snapshot}" var = "count">
      UPDATE Employees SET WHERE last = 'Ali'
      <sql:param value = "${empId}" />
    </sql:update>

    <sql:query dataSource = "${snapshot}" var = "result">
      SELECT * from Employees;
    </sql:query>

    <table border = "1" width = "100%">
      <tr>
        <th>Emp ID</th>
        <th>First Name</th>
        <th>Last Name</th>
        <th>Age</th>
      </tr>

      <c:forEach var = "row" items = "${result.rows}">
        <tr>
          <td><c:out value = "${row.id}"/></td>
          <td><c:out value = "${row.first}"/></td>
          <td><c:out value = "${row.last}"/></td>
          <td><c:out value = "${row.age}"/></td>
        </tr>
      </c:forEach>
    </table>

  </body>
</html>
```

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	Ali	30

******************* ALL THE BEST **********************************