*****JSTL (JSP Standard Tag Library)*****

Question :- What is wrong in using JSP Scriptlet Tags?

Ans:-

using scriptlet code produces page source code that contains a mix of HTML tags and Java statements. There are several reasons why this mixing of programming styles is not optimal.

The primary reason why you shouldn't mix scriptlet and tag-based code is the readability issue, which applies both to humans and computers.

Additionally, scriptlet code can be more difficult for a HTML programmer to produce. A HTML programmer must learn the syntax of Java to produce scriptlet code.

Example:-

```
<html>
 <head>
  <title>Count to 10 in JSP scriptlet</title>
 </head>
 <body>
            <%
                        for(int i=1;i<=10;i++)
                        {%>
                                    <%=i%><br/>
                        <%
                        }
            %>
 </body>
</html>
```

Question :- How JSTL Fixes JSP Scriptlet's Shortcomings.

Ans:-

JSTL allows the human programmer to look at a program that consists entirely of HTML and HTML-like tags.

With JSTL, the HTML programmer is already programming in a familiar tag-based

syntax.

Consider the following example, which shows how to count from 1 to 10 using JSTL rather than scriptlet code.

```
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
 <head>
  <title>Count to 10 Example (using JSTL)</title>
 </head>

 <body>
  <c:forEach var="i" begin="1" end="10" step="1">
   <c:out value="${i}" />

   <br />
  </c:forEach>
 </body>
</html>
```

When you examine the preceding source code, you can see that the JSP page consists entirely of tags. The code makes use of HTML tags such <head> and
. The use of tags is not confined just to HTML tags; this code also makes use of JSTL tags such as <c:forEach> and <c:out>.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

1. Fast Development JSTL provides many tags that simplify the JSP.
2. Code Reusability We can use the JSTL tags on various pages.
3. No need to use scriptlet tag It avoids the use of scriptlet tag.

Question:- Disadvantages Of JSTL?

Ans:- Following are some disadvantages of JSTL.

1) Overhead

Both JSP scriptlet and JSTL code are translated to a Java servlet for execution. Because JSP scriptlet code is really just Java embedded in JSP files, there is relatively

little overhead in the scriptlet code that is generated. This is not the case with tag libraries such as JSTL. Large JSP files that make use of JSTL will generate a great deal of overhead code. This means that more server-processing power is required to run JSTL-based sites than pure JSP scriptlet sites.

Another disadvantage of this overhead is that the size of a Java class is limited. Large complex JSP pages that make extensive use of JSTL and other tag libraries can quickly exceed this class-size limitation. If you run into this limitation, you will need to split your JSP page into multiple files.

2) Not as Extensive as JSP Scriptlet

JSP scriptlet code is really just Java code that is inserted into the servlet that the Web server generates, so JSP scriptlet code can make use of nearly any class that Java includes. This gives great flexibility in terms of what you can actually include in this JSP code of your page.

JSTL is not so flexible. JSTL code is a new programming language that is not nearly as evolved as Java itself. This means that if you are to create a pure JSTL implementation, you do not have access to the number of classes that you would if you were using a JSP scriptlet.

Although this is a limitation, it should not hinder you too much. JSP pages should usually contain only presentation-level code. Although JSP scriptlets give you access to the entire JDK, do you really want to be opening socket connections from your JSP pages? Such things are usually left to JavaBeans.

3) May Seem Burdensome for Experienced Programmers

Without a doubt, JSTL makes programming easier for a programmer with only an HTML skill set. However, for the advanced Java programmer who already knows how to use Java well, JSTL often is viewed as simply one more programming language to learn. Plus, if JSTL just gets translated into Java code anyway, an advanced programmer may wonder why he does not simply do that himself and save the time.

If a programmer is already familiar with Java, JSP scriptlet is not terribly hard to learn. This can make the equivalent JSTL code look larger and more complex. After all, all JSTL code is valid XML. This alone means extra typing for your programmers because it takes more keystrokes to produce an XML tag than does its scriptlet counterpart.

Question:- JSTL Tag Libraries.

Ans:-

Classification of The JSTL Tags

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −

- **Core Tags**
- **Formatting tags**
- **SQL tags**
- **XML tags**
- **JSTL Functions**

Core Tag

- **<c:out>**

The **<c:out>** tag displays the result of an expression. This is almost similar to the way **<%= %>** works. The difference here is that **<c:out>** tag lets you use the simpler **"."** notation to access properties. For example, to access customer.address.street,

use the tag **<c:out value = "customer.address.street"/>**.

The **<c:out>** tag can automatically escape XML tags so they aren't evaluated as actual tags.

Attribute

The <c:out> tag has the following attributes −

Attribute	Description	Required	Default
Value	Information to output	Yes	None
Default	Fallback information to output	No	body
escapeXml	True if the tag should escape special XML	No	true

	characters		

- **<c:set >**

The **<c:set>** tag is JSTL-friendly version of the **setProperty** action. The tag is helpful because it evaluates an expression and uses the results to set a value of a JavaBean or a **java.util.Map object**.

Attribute

The **<c:set>** tag has the following attributes –

Attribute	Description	Required	Default
Value	Information to save	No	body
Target	Name of the variable whose property should be modified	No	None
property	Property to modify	No	None
Var	Name of the variable to store information	No	None
Scope	Scope of variable to store information	No	Page

If target is specified, property must also be specified.

Example of out & set tag

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> out & set Tag Example</title>
  </head>
```

```
  <body>
    <c:set var = "salary" scope = "session" value = "${2000*2}"/>
    <c:out value = "${salary}"/>

    <jsp:useBean id="b1" class="p1.JavaBean" ></jsp:useBean>
    <c:set target="${b1}" property="mob" value="123" />
    <jsp:getProperty name="b1" property="mob"></jsp:getProperty>
  </body>
</html>
```

POJO file (store this file into 'Sourse Packages' folder)

```
package p1;
public class JavaBean
{
   private int mob;
   public int getMob() {
      return mob;
   }
   public void setMob(int mob) {
      this.mob = mob;
   }
}
```

- **<c:remove>**

The **<c:remove>** tag removes a variable from either a specified scope or the first scope where the variable is found (if no scope is specified). This action is not particularly helpful, but it can aid in ensuring that a JSP cleans up any scoped resources it is responsible for.

Attribute

The **<c:remove>** tag has the following attributes –

Attribute	Description	Required	Default
var	Name of the variable to remove	Yes	None
scope	Scope of the variable to remove	No	All scopes

Example

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title>remove Tag Example</title>
  </head>

  <body>
    <c:set var = "salary" scope = "session" value = "${2*4000}"/>
    <p>Before Remove Value: <c:out value = "${salary}"/></p>
      <c:remove var = "salary"/>
    <p>After Remove Value: <c:out value = "${salary}"/></p>
  </body>
</html>
```

- **<c:forEach>**

These tags exist as a good alternative to embedding a Java **for, while,** or **do-while** loop via a scriptlet. The **<c:forEach>** tag is a commonly used tag because it iterates over a collection of objects. The **<c:forTokens>** tag is used to break a string into tokens and iterate through each of the tokens.

Attribute

The **<c:forEach>** tag has the following attributes –

Attribute	Description	Required	Default
items	Information to loop over	No	None
begin	Element to start with (0 = first item, 1 = second item, ...)	No	0
end	Element to end with (0 = first item, 1 = second item, ...)	No	Last element
step	Process every step items	No	1
var	Name of the variable to expose the current item	No	None

varStatus	Name of the variable to expose the loop status	No	None

Example for <c:forEach>

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> forEach Tag Example</title>
  </head>

  <body>
    <c:forEach var = "i" begin = "1" end = "5">
      Item <c:out value = "${i}"/><br>
    </c:forEach>
  </body>
</html>
```

The above code will generate the following result –

Item 1
Item 2
Item 3
Item 4
Item 5

Example for

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> forEach Tag Example</title>
  </head>

  <body>
    <c:forEach items = "Ramesh,Satish,Rahul" var = "name">
      <c:out value = "${name}"/><br>
    </ c:forEach>
  </body>
</html>
```

The above code will generate the following result –

Ramesh
Satish
Rahul

- **<c:forTokens>**

The **<c:forTokens>** tag has similar attributes as that of the **<c:forEach>** tag except one additional attribute **delims** which specifies sharacters to use as delimiters.

Attribute	Description	Required	Default
Delims	Characters to use as delimiters	Yes	None

Example for <c:forTokens>

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> forTokens Tag Example</title>
  </head>

  <body>
    <c:forTokens items = "Ramesh;Satish;Rahul" delims = ";" var = "name">
      <c:out value = "${name}"/><p>
    </c:forTokens>
  </body>
</html>
```

The above code will generate the following result –

Ramesh
Satish
Rahul

- **<c:catch>**

The **<c:catch>** tag catches any **Throwable** that occurs in its body and optionally exposes it. It is used for error handling and to deal more gracefully with the problem.

Attribute

The **<c:catch>** tag has the following attributes –

Attribute	Description	Required	Default

var	The name of the variable to hold the java.lang.Throwable if thrown by elements in the body.	No	None

Example

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> catch Tag Example</title>
  </head>

  <body>
    <c:catch var ="catchException">
      <% int x = 10/0;%>
    </c:catch>
      The exception is : ${catchException} <br>
  </body>
</html>
```

The above code will generate the following result –

The exception is : java.lang.ArithmaticException: / by zero

- **<c:if>**

The **<c:if>** tag evaluates an expression and displays its body content only if the expression evaluates to true.

Attribute

The **<c:if>** tag has the following attributes –

Attribute	Description	Required	Default
test	Condition to evaluate	Yes	None
var	Name of the variable to store the condition's result	No	None
scope	Scope of the variable to store the condition's result	No	page

Example

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title>  if Tag Example</title>
  </head>

  <body>
    <c:set var = "salary" scope = "session" value = "${2000*2}"/>
    <c:if test = "${salary > 2000}" >
      <p>My salary is:  <c:out value = "${salary}"/><p>
    </c:if>
  </body>
</html>
```

The above code will generate the following result –

My salary is: 4000

- **<c:choose>**
- **<c:when>**
- **<c:otherwise>**

The **<c:choose>** works like a Java **switch** statement in that it lets you choose between a number of alternatives. Where the **switch** statement has **case** statements, the **<c:choose>** tag has **<c:when>** tags. Just as a switch statement has the **default** clause to specify a default action, **<c:choose>** has **<c:otherwise>** as the default clause.

Attribute

- The **<c:choose>** tag does not have any attribute.
- The **<c:when>** tag has one attributes which is listed below.
- The **<c:otherwise>** tag does not have any attribute.

The **<c:when>** tag has the following attributes –

Attribute	Description	Required	Default
test	Condition to evaluate	Yes	None

Example

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title> choose, when & otherwise Tags Example</title>
  </head>

  <body>
    <c:set var = "salary" scope = "session" value = "${2000*2}"/>
    <p>Your salary is : <c:out value = "${salary}"/></p>
    <c:choose>

      <c:when test = "${salary <= 0}">
        Salary is very low to survive.
      </c:when>

      <c:when test = "${salary > 1000}">
        Salary is very good.
      </c:when>

      <c:otherwise>
        No comment sir...
      </c:otherwise>
    </c:choose>

  </body>
</html>
```

The above code will generate the following result –

Your salary is : 4000
Salary is very good.

- **<c:import>**

The **<c:import>** tag provides all functionalities of the **<include>** action but also allows for the inclusion of absolute URLs.

For example, using the import tag allows for the inclusion of content from a different Website or an FTP server.

Attribute

The **<c:import>** tag has the following attributes –

Attribute	Description	Required	Default
url	URL to retrieve and import into the page	Yes	None
context	/ followed by the name of a local web application	No	Current application
charEncoding	Character set to use for imported data	No	ISO-8859-1
var	Name of the variable to store imported text	No	Print to page
scope	Scope of the variable used to store imported text	No	Page
varReader	Name of an alternate variable to expose java.io.Reader	No	None

Example

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title>import Tag Example</title>
  </head>

  <body>
    <c:import var = "data" url = "http://www.google.co.in "/>
    <c:out value = "${data}"/>
  </body>
</html>
```

- **<c:redirect>**

The **<c:redirect>** tag redirects the browser to an alternate URL by facilitating automatic URL rewriting, it supports context-relative URLs, and it also supports the **<c:param>** tag.

Attribute

The **<c:redirect>** tag has the following attributes −

Attribute	Description	Required	Default
url	URL to redirect the user's browser to	Yes	None
context	/ followed by the name of a local web application	No	Current application

Example

If you need to pass parameters to a **<c:import>** tag, use the **<c:url>** tag to create the URL first as shown below −

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title>redirect Tag Example</title>
  </head>

  <body>

    <c:redirect url = "https://www.google.co.in "/>
  </body>
</html>
```

- **<c:url>**

The **<c:url>** tag formats a URL into a string and stores it into a variable. This tag automatically performs URL rewriting when necessary. The **var** attribute specifies the variable that will contain the formatted URL.

The **JSTL** url tag is just an alternative method of writing the call to the **response.encodeURL()** method. The only real advantage the url tag provides is proper URL encoding, including any parameters specified by children **param** tag.

Attribute

The **<c:url>** tag has the following attributes −

Attribute	Description	Required	Default

Value	Base URL	Yes	None
context	/ followed by the name of a local web application	No	Current application
var	Name of the variable to expose the processed URL	No	Print to page
scope	Scope of the variable to expose the processed URL	No	Page

- **<c:param>**

The **<c:param>** tag allows proper URL request parameter to be specified with URL and also does the necessary URL encoding required.

Within a **<c:param>** tag, the name attribute indicates the parameter name, and the value attribute indicates the parameter value −

Attribute

The **<c:param>** tag has the following attributes −

Attribute	Description	Required	Default
name	Name of the request parameter to set in the URL	Yes	None
value	Value of the request parameter to set in the URL	No	Body

Example

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>

<html>
  <head>
    <title>url & param Tag Example</title>
  </head>

  <body>
```

```
        <c:url value = "/index.jsp" var = "myURL">
                <c:param name = "trackingId" value = "1234"/>
                <c:param name = "reportType" value = "summary"/>
        </c:url>
        <c:out value = "${myURL}"/>
</body>
</html>
```

Sql Tag

- **<sql:setDataSource>**

The **<sql:setDataSource>** tag sets the data source configuration variable or saves the data-source information in a scoped variable that can be used as input to the other JSTL database actions.

Attribute

The **<sql:setDataSource>** tag has the following attributes −

Attribute	Description	Required	Default
driver	Name of the JDBC driver class to be registered	No	None
url	JDBC URL for the database connection	No	None
user	Database username	No	None
password	Database password	No	None
var	Name of the variable to represent the database	No	Set default
scope	Scope of the variable to represent the database	No	Page

Example

Consider the following information about your MySQL database setup −

- We are using **JDBC MySQL** driver.

- We are going to connect to TEST database on local machine.
- We would use **user_id** and **mypassword** to access TEST database.

All the above parameters will vary based on your MySQL or any other database setup. Considering the above parameters, following example uses the **setDataSource** tag −

```
<%@ 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>JSTL sql:setDataSource Tag</title>
  </head>

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

  </body>
</html>
```

- **<sql:update>**

The **<sql:update>** tag executes an SQL statement that does not return data; for example, **SQL INSERT, UPDATE**, or **DELETE** statements.

Attribute

The **<sql:update>** tag has the following attributes −

Attribute	Description	Required	Default
sql	SQL command to execute (should not return a ResultSet)	No	Body
dataSource	Database connection to use (overrides the default)	No	Default database
var	Name of the variable to store the count of affected rows	No	None
scope	Scope of the variable to store the count	No	Page

	of affected rows		

Example

Let us now write a JSP which will make use of the **<sql:update>** tag to execute an **SQL INSERT** statement to create one record in the table as follows −

```
<%@ 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>JSTL sql:update Tag</title>
  </head>

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

    <sql:update dataSource = "${snapshot}" var = "count">
      INSERT INTO student VALUES (101, 'Deepak', 'Panvel');
    </sql:update>
  </body>
</html>
```

- **<sql:query>**

The **<sql:query>** tag executes an SQL SELECT statement and saves the result in a scoped variable.

Attribute

The **<sql:query>** tag has the following attributes −

Attribute	Description	Required	Default
Sql	SQL command to execute (should return a ResultSet)	No	Body
dataSour	Database connection to use	No	Default

ce	(overrides the default)		database
maxRows	Maximum number of results to store in the variable	No	Unlimited
startRow	Number of the row in the result at which to start recording	No	0
Var	Name of the variable to represent the database	No	Set default
Scope	Scope of variable to expose the result from the database	No	Page

Example

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>

<html>
  <head>
    <title>JSTL sql:query Tag</title>
  </head>

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

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

    <table border = "1" align="center">
      <tr>
        <th>Roll No</th>
        <th>Student Name</th>
        <th>Address</th>

      </tr>

      <c:forEach var = "row" items = "${result.rows}">
```

```
        <tr>
          <td> <c:out value = "${row.roll_no}"/></td>
          <td> <c:out value = "${row.stud_name}"/></td>
          <td> <c:out value = "${row.stud_add}"/></td>
        </tr>
      </c:forEach>
    </table>
  </body>
</html>
```

- **<sql:param>**

The **<sql:param>** tag used as a nested action for the **<sql:query>** tag and the **<sql:update>** tag to supply a value for a value placeholder. If a null value is provided, the value is set to **SQL NULL** for the placeholder.

Attribute

The **<sql:param>** tag has the following attributes –

Attribute	Description	Required	Default
Value	Value of the parameter to set	No	Body

Example

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>

<html>
  <head>
    <title>JSTL sql:param Tag</title>
  </head>

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

    <c:set var = "empId" value = "103"/>

    <sql:update dataSource = "${snapshot}" var = "count">
```

```
      DELETE FROM student WHERE roll_no = ?
      <sql:param value = "${empId}" />
    </sql:update>
    <c:out value="Data Deleted." />

  </body>
</html>
```

- **<sql:dateParam>**

The **<sql:dateParam>** tag is used as a nested action for the **<sql:query>** and the **<sql:update>** tag to supply a date and time value for a value placeholder. If a null value is provided, the value is set to **SQL NULL** for the placeholder.

Attribute

The **<sql:dateParam>** tag has the following attributes –

Attribute	Description	Required	Default
Value	Value of the date parameter to set (java.util.Date)	No	Body
type	DATE (date only), TIME (time only), or TIMESTAMP (date and time)	No	TIMESTAMP

Example

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ page import = "java.util.Date,java.text.*" %>
<%@ 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>JSTL sql:dataParam Tag</title>
  </head>

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

```
    <%
      Date DoB = new Date("2020/07/15");
      int studentId = 100;
    %>

    <sql:update dataSource = "${snapshot}" var = "count">
      UPDATE Students SET dob = ? WHERE Id = ?
      <sql:dateParam value = "<%=DoB%>" type = "DATE" />
      <sql:param value = "<%=studentId%>" />
    </sql:update>


  </body>
</html>
```

- **<sql:transaction>**

The **<sql:transaction>** tag is used to group the **<sql:query>** and **<sql:update>** tags into transactions. You can put as many **<sql:query>** and **<sql:update>** tags as statements inside the **<sql:transaction>** tag to create a single transaction.

It ensures that the database modifications performed by the nested actions are either committed or rolled back if an exception is thrown by any nested action.

Attribute

The **<sql:transaction>** tag has the following attributes –

Attribute	Description	Required	Default
dataSource	Database connection to use (overrides the default)	No	Default database
isolation	Transaction isolation (READ_COMMITTED, READ_UNCOMMITTED, REPEATABLE_READ, or SERIALIZABLE)	No	Database's default

Example

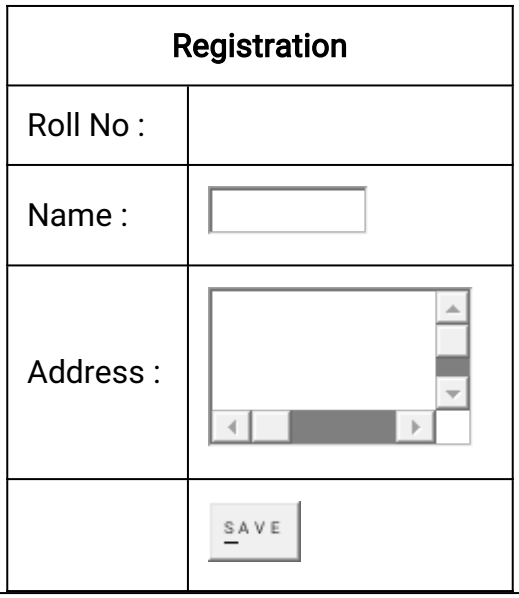
```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>
```

```
<html>
  <head>
    <title>JSTL sql:transaction Tag</title>
  </head>

  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
    url = "jdbc:mysql://localhost/TYIT" user = "root" password = ""/>
    <sql:transaction dataSource = "${snapshot}">
    <sql:update >
        insert into student values(105,'Rahul','Panvel');
    </sql:update>

    <sql:update  >
         insert into student values(105,'Pravin','New Panvel');
    </sql:update>
    </sql:transaction>
    <c:out value="Transaction Completed"/>
```

Simple Registration Example



```
<html>
   <head>
      <title>TODO supply a title</title>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
   </head>
   <body>
```

```
      <form action="data_save.jsp" method="post">
         <table border="1" align="center" cellspacing="5" cellpadding="10">
            <tr><th colspan="2">Registration</th></tr>
            <tr><td>Roll No :</td><td><input type="number" name="t1"></td></tr>
            <tr><td>Name :</td><td><input type="text" name="t2"></td></tr>
            <tr><td>Address :</td><td><textarea name="t3" rows="4"
cols="20"></textarea></td></tr>
            <tr><td></td><td><input type="submit" value="SAVE"></td></tr>
         </table>
      </form>
   </body>
</html>
```

data_save.jsp

```
<html>
  <head>
    <title>JSTL sql:setDataSource Tag</title>
  </head>

  <body>
    <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
      url = "jdbc:mysql://localhost/TYIT"
      user = "root"  password = ""/>
    <sql:update dataSource = "${snapshot}" >
       insert into student values(?,?,?);
       <sql:param value="${param.t1}"/>
       <sql:param value="${param.t2}"/>
       <sql:param value="${param.t3}"/>

    </sql:update>
    <c:out value="Data Saved."/>

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

    <table border = "1" align="center">
      <tr>
        <th>Roll No</th>
        <th>Student Name</th>
        <th>Address</th>

      </tr>
```

```
      <c:forEach var = "row" items = "${result.rows}">
        <tr>
          <td> <c:out value = "${row.roll_no}"/></td>
          <td> <c:out value = "${row.stud_name}"/></td>
          <td> <c:out value = "${row.stud_add}"/></td>

        </tr>
      </c:forEach>
    </table>
  </body>
</html>
```

XML tags

The JSTL XML tags provide a JSP-centric way of creating and manipulating the XML documents. Following is the syntax to include the JSTL XML library in your JSP.

The JSTL XML tag library has custom tags for interacting with the XML data. This includes parsing the XML, transforming the XML data, and the flow control based on the XPath expressions.

<%@ taglib prefix = "x" uri = "http://java.sun.com/jsp/jstl/xml" %>

Before you proceed with the examples, you will need to copy the following two XML and XPath related libraries into your **<Tomcat Installation Directory>\lib** −

- **XercesImpl.jar** − Download it from https://www.apache.org/dist/xerces/j/
- **xalan.jar** − Download it from https://xml.apache.org/xalan-j/index.html

Following is the list of XML JSTL Tags −

<table>
<tr><th>S.No.</th><th>Tag & Description</th><th>Attribute</th></tr>
<tr><td>1</td><td><x:out>

Like <%= ... >, but for XPath expressions.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>select</td><td>XPath expression to evaluate as a string, often using XPath variables</td><td>Yes</td><td>None</td></tr>
<tr><td>escapeXml</td><td>True if the tag should escape special XML characters</td><td>No</td><td>true</td></tr>
</table>
</td></tr>
<tr><td>2</td><td><x:parse></td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
</table>
</td></tr>
</table>

	Used to parse the XML data specified either via an attribute or in the tag body.	var	A variable that contains the parsed XML data	No	None
		xml	Text of the document to parse (String or Reader)	No	Body
		doc	XML document to be parsed	No	Page
		scope	Scope of the variable specified in the var attribute	No	Page
3	<x:set > Sets a variable to the value of an XPath expression.	**Attribute**	**Description**	**Required**	**Default**
		var	A variable that is set to the value of the XPath expression	Yes	Body
		select	The XPath expression to be evaluated	No	None
		scope	Scope of the variable specified in the var attribute	No	Page
4	<x:if > Evaluates a test XPath expression and if it is true, it processes its body. If the test condition is false, the body is ignored.	**Attribute**	**Description**	**Required**	**Default**
		select	The XPath expression to be evaluated	Yes	None
		var	Name of the variable to store the condition's result	No	None
		scope	Scope of the variable specified in the var attribute	No	Page
5	<x:forEach> To loop over nodes in an XML document.	**Attribute**	**Description**	**Required**	**Default**
		select	The XPath expression to be evaluated	Yes	None
		var	Name of the variable to store the condition's result	No	None

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>scope</td><td>Scope of the variable specified in the var attribute</td><td>No</td><td>Page</td></tr>
</table>
</td></tr>
<tr><td>6</td><td><x:choose>

Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise> tags.</td><td>NO</td></tr>
<tr><td>7</td><td><x:when >

Subtag of <choose> that includes its body if its expression evalutes to 'true'.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>select</td><td>Condition to evaluate</td><td>Yes</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>8</td><td><x:otherwise >

Subtag of <choose> that follows the <when> tags and runs only if all of the prior conditions evaluates to 'false'.</td><td>NO</td></tr>
<tr><td>9</td><td><x:transform >

Applies an XSL transformation on a XML document</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>doc</td><td>Source XML document for the XSLT transformation</td><td>No</td><td>Body</td></tr>
<tr><td>docSystemId</td><td>URI of the original XML document</td><td>No</td><td>None</td></tr>
<tr><td>xslt</td><td>XSLT stylesheet providing transformation instructions</td><td>Yes</td><td>None</td></tr>
<tr><td>xsltSystemId</td><td>URI of the original XSLT document</td><td>No</td><td>None</td></tr>
<tr><td>result</td><td>Result object to accept the transformation's result</td><td>No</td><td>Print to page</td></tr>
<tr><td>var</td><td>Variable that is set to the</td><td>No</td><td>Print to</td></tr>
</table>
</td></tr>
</table>

			transformed XML document		page
		scope	Scope of the variable to expose the transformation's result	No	None
10	<x:param > Used along with the **transform** tag to set a parameter in the XSLT stylesheet	**Attribute**	**Description**	**Required**	**Default**
		name	Name of the XSLT parameter to set	Yes	Body
		Value	Value of the XSLT parameter to set	No	None

EXAMPLE :-

```
<%@ taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix = "x" uri = "http://java.sun.com/jsp/jstl/xml" %>

<html>
  <head>
    <title>JSTL xml Tags example</title>
  </head>

  <body>
    <h3>Books Info:</h3>

    <c:set var = "xmltext">
      <books>
        <book>
          <name>Enterprise Java </name>
          <author>Omkar</author>
          <price>300</price>
        </book>

        <book>
          <name>Asp .Net with C#</name>
          <author>Nikhil</author>
          <price>200</price>
```

```
        </book>
      </books>
    </c:set>

    <x:parse xml = "${xmltext}" var = "output"/>

      <x:forEach select = "$output/books/book/name" var = "item">
        Book Name: <x:out select = "$item" /><br>
      </x:forEach>

          <br>
        <h3>Book details</h3>
      <x:forEach select = "$output/books/book" var = "item">
        Book Name: <x:out select = "$item/name" /><br>
        Book Author: <x:out select = "$item/author" /><br>
        Book Price: <x:out select = "$item/price" /><br><br>
      </x:forEach>
  </body>
</html>
```

- **JSTL Function Tags**

The JSTL function provides a number of standard functions, most of these functions are common string manipulation functions. The syntax used for including JSTL function library in your JSP is:

<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %**>**

JSTL Function Tags List

JSTL Functions	Syntax	Description
fn:contains()	boolean contains(String inputstring, String substring)	It is used to test if an input string containing the specified substring in a program.
fn:containsIgnoreCase()	boolean contains(String inputstring, String substring)	It is used to test if an input string contains the specified substring as a case insensitive way.

fn:endsWith()	boolean endsWith(String input, String substring)	It is used to test if an input string ends with the specified suffix.
fn:escapeXml()	String escapeXml(String input)	It escapes the characters that would be interpreted as XML markup.
fn:indexOf()	int indexOf(String input, String search)	It returns an index within a string of first occurrence of a specified substring.
fn:trim()	String trim(String input)	It removes the blank spaces from both the ends of a string.
fn:startsWith()	boolean startsWith(String input, String substring)	It is used for checking whether the given string is started with a particular string value.
fn:split()	String[] split(String input, String delimit)	It splits the string into an array of substrings.
fn:toLowerCase()	String toLowerCase(String input)	It converts all the characters of a string to lower case.
fn:toUpperCase()	String toUpperCase(String input)	It converts all the characters of a string to upper case.
fn:substring()	String substring(String input, int start, int end)	It returns the subset of a string according to the given start and end position.
fn:substringAfter()	String substringAfter(String	It returns the subset of string after a specific substring.

	input,String sub)	
fn:substringBefore()	String substringBefore(String input, String sub)	It returns the subset of string before a specific substring.
fn:length()	int length(java.lang.Object)	It returns the number of characters inside a string, or the number of items in a collection.
fn:replace()	boolean replace(String input, String find, String replacewith)	It replaces all the occurrence of a string with another string sequence.

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/functions" prefix = "fn" %>

<html>
  <head>
    <title>Using JSTL Functions</title>
  </head>

  <body>
    <c:set var = "string1" value = "This is first String."/>
    <c:set var = "string2" value = "${fn:substringBefore(string1, 'first')}" />
    <p>before : ${string2}</p>
    after : ${fn:substringAfter(string1, 'first')}<br>
    UpperCase : ${fn:toUpperCase(string1)}<br>
    LowerCase : ${fn:toLowerCase(string1)}<br>
    length    : ${fn:length(string1)}<br>
    escapeXml : ${fn:escapeXml("This is <abc>example</abc>")}<br>
    wihout escapeXml : ${"This is <abc>example</abc>"}<br>
    trim : ${fn:trim("    This is      JSTL     ")}<br>
    split : ${fn:split("This is JSTL"," ")}
  </body>
</html>
```

- **Formatting Tags**

The JSTL formatting tags are used to format and display text, the date, the time, and numbers for internationalized Websites. Following is the syntax to include Formatting library in your JSP −

<%@ taglib prefix = "fmt" uri = "http://java.sun.com/jsp/jstl/fmt" %>

<table>
<tr><th>Formatting Tags</th><th>Descriptions</th></tr>
<tr><td>fmt:parseNumber</td><td>It is used to Parses the string representation of a currency, percentage or number.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>Value</td><td>Numeric value to read (parse)</td><td>No</td><td>Body</td></tr>
<tr><td>type</td><td>NUMBER, CURRENCY, or PERCENT</td><td>No</td><td>number</td></tr>
<tr><td>parseLocale</td><td>Locale to use when parsing the number</td><td>No</td><td>Default locale</td></tr>
<tr><td>integerOnly</td><td>Whether to parse to an integer (true) or floating-point number (false)</td><td>No</td><td>false</td></tr>
<tr><td>pattern</td><td>Custom parsing pattern</td><td>No</td><td>None</td></tr>
<tr><td>timeZone</td><td>Time zone of the displayed date</td><td>No</td><td>Default time zone</td></tr>
<tr><td>var</td><td>Name of the variable to store the parsed number</td><td>No</td><td>Print to page</td></tr>
<tr><td>scope</td><td>Scope of the variable to store the formatted number</td><td>No</td><td>page</td></tr>
</table>
</td></tr>
<tr><td>fmt:formatNumber</td><td>It is used to format the numerical value with specific format or precision.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>Value</td><td>Numeric value to display</td><td>Yes</td><td>None</td></tr>
<tr><td>type</td><td>NUMBER, CURRENCY, or PERCENT</td><td>No</td><td>Number</td></tr>
</table>
</td></tr>
</table>

		pattern	Specify a custom formatting pattern for the output.	No	None
		currencyCode	Currency code (for type = "currency")	No	From the default locale
		currencySymbol	Currency symbol (for type = "currency")	No	From the default locale
		groupingUsed	Whether to group numbers (TRUE or FALSE)	No	true
		maxIntegerDigits	Maximum number of integer digits to print	No	None
		minIntegerDigits	Minimum number of integer digits to print	No	None
		maxFractionDigits	Maximum number of fractional digits to print	No	None
		minFractionDigits	Minimum number of fractional digits to print	No	None
		var	Name of the variable to store the formatted number	No	Print to page
		scope	Scope of the variable to store the formatted number	No	page
fmt:parseDate	It parses the string representation of a time and date.	**Attribute**	**Description**	**Required**	**Default**
		Value	Date value to read (parse)	No	Body

		type	DATE, TIME, or BOTH	No	date
		dateStyle	FULL, LONG, MEDIUM, SHORT, or DEFAULT	No	Default
		timeStyle	FULL, LONG, MEDIUM, SHORT, or DEFAULT	No	Default
		parseLocale	Locale to use when parsing the date	No	Default locale
		pattern	Custom parsing pattern	No	None
		timeZone	Time zone of the parsed date	No	Default time zone
		var	Name of the variable to store the parsed date	No	Print to page
		scope	Scope of the variable to store the formatted date	No	page
fmt:formatDate	It formats the time and/or date using the supplied pattern and styles.	**Attribute**	**Description**	**Required**	**Default**
		Value	Date value to display	Yes	None
		type	DATE, TIME, or BOTH	No	date
		dateStyle	FULL, LONG, MEDIUM, SHORT, or DEFAULT	No	default
		timeStyle	FULL, LONG, MEDIUM, SHORT, or DEFAULT	No	default
		pattern	Custom formatting pattern	No	None
		timeZone	Time zone of the displayed date	No	Default time zone
		var	Name of the variable to store the formatted date	No	Print to page

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>scope</td><td>Scope of the variable to store the formatted date</td><td>No</td><td>page</td></tr>
</table>
</td></tr>
<tr><td>fmt:bundle</td><td>It is used for creating the ResourceBundle objects which will be used by their tag body.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>basename</td><td>Specifies the base name of the resource bundle that is to be loaded.</td><td>Yes</td><td>None</td></tr>
<tr><td>Prefix</td><td>Value to prepend to each key name in <fmt:message> subtags</td><td>No</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>fmt:setBundle</td><td>It loads the resource bundle and stores it in a bundle configuration variable or the named scoped variable.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>basename</td><td>Base name of the resource bundle family to expose as a scoped or a configuration variable</td><td>Yes</td><td>None</td></tr>
<tr><td>var</td><td>Name of the variable to store the new bundle</td><td>No</td><td>Replace default</td></tr>
<tr><td>scope</td><td>Scope of the variable to store the new bundle</td><td>No</td><td>Page</td></tr>
</table>
</td></tr>
<tr><td>fmt:timeZone</td><td>It specifies a parsing action nested in its body or the time zone for any time formatting.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>Value</td><td>Time zone to apply to the body</td><td>Yes</td><td>None</td></tr>
</table>
</td></tr>
<tr><td>fmt:setTimeZone</td><td>It stores the time zone inside a time zone configuration variable.</td><td>
<table>
<tr><th>Attribute</th><th>Description</th><th>Required</th><th>Default</th></tr>
<tr><td>Value</td><td>Time zone to expose as a scoped or configuration variable</td><td>Yes</td><td>None</td></tr>
</table>
</td></tr>
</table>

		var	Name of the variable to store the new time zone	No	Replace default
		scope	Scope of the variable to store the new time zone	No	Page
fmt:message	It display an internationalized message.	**Attribute**	**Description**	**Required**	**Default**
		key	Message key to retrieve	No	Body
		bundle	Resource bundle to use	No	Default bundle
		var	Name of the variable to store the localized message	No	Print to page
		scope	Scope of the variable to store the localized message	No	Page

Example

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>

<html>
<head>
<title>fmt:formatNumber Tag</title>
</head>
<body>

<h3>Formatting of Number:</h3>
<c:set var="Amount" value="9850.14115" />
<p> Formatted Number-1:
<fmt:formatNumber value="${Amount}" type="currency" /></p>
<p>Formatted Number-2:
<fmt:formatNumber type="number" groupingUsed="true" value="${Amount}" /></p>
<p>Formatted Number-3:
```

```
<fmt:formatNumber type="number" maxIntegerDigits="3" value="${Amount}" /></p>
<p>Formatted Number-4:
<fmt:formatNumber type="number" maxFractionDigits="6" value="${Amount}" /></p>
<p>Formatted Number-5:
<fmt:formatNumber type="percent" maxIntegerDigits="4" value="${Amount}" /></p>
<p>Formatted Number-6:
<fmt:formatNumber type="number" pattern="###.###$" value="${Amount}" /></p>
</body>
</html>
```

Output:

```
Formatting of Number:
Formatted Number-1: $9,850.14
Formatted Number-2: 9,850.141
Formatted Number-3: 850.141
Formatted Number-4: 9,850.14115
Formatted Number-5: 5,014%
Formatted Number-6: 9850.141$
```

Example

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>

<html>
<head>
  <title>fmt:parseDate Tag</title>
</head>
<body>
<h3>Parsed Date:</h3>
<c:set var="date" value="12-08-2016" />

<fmt:parseDate value="${date}" var="parsedDate"  pattern="dd-MM-yyyy" />
<p><c:out value="${parsedDate}" /></p>

</body>
</html>
```

Output:

Parsed Date:

Fri Aug 16 00:00:00 IST 2019

Example of <fmt:formatDate> tag:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<html>
<head>
<title>fmt:formatDate</title>
</head>
<body>
<h2>Different Formats of the Date</h2>
<c:set var="Date" value="<%=new java.util.Date()%>" />
<p>
Formatted Time :
<fmt:formatDate type="time" value="${Date}" />
</p>
<p>
Formatted Date :
<fmt:formatDate type="date" value="${Date}" />
</p>
<p>
Formatted Date and Time :
<fmt:formatDate type="both" value="${Date}" />
</p>
<p>
Formatted Date and Time in short style :
<fmt:formatDate type="both" dateStyle="short" timeStyle="short"
value="${Date}" />
</p>
<p>
Formatted Date and Time in medium style :
<fmt:formatDate type="both" dateStyle="medium" timeStyle="medium"
value="${Date}" />
</p>
<p>
Formatted Date and Time in long style :
<fmt:formatDate type="both" dateStyle="long" timeStyle="long"
value="${Date}" />
</p>

</body>
</html>
```

Output:

```
Different Formats of the Date
Formatted Time : 4:20:50 PM
Formatted Date : Aug 16, 2019
Formatted Date and Time : Aug 16, 2019 4:20:50 PM
```

Formatted Date and Time in short style : 8/16/19 4:20 PM
Formatted Date and Time in medium style : Aug 16, 2019 4:20:50 PM
Formatted Date and Time in long style : August 16, 2019 4:20:50 PM IST