JSTL

JSTL stands for Java server pages standard tag library. The JSTL contains several tags that can remove scriplet code from a JSP page by providing some ready to use, already implemented common functionalities. It -

- encapsulates the core functionality common to many JSP applications.
- provides collection of custom JSP tag libraries that provide common web development functionality.
- provides tags to control the JSP page behavior.
- has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags.
- also provides a framework for integrating the existing custom tags with the JSTL tags.

Initially, Web designers used scriptlets in JSP pages to generate dynamic content. This resulted in readability issues and also made it difficult to maintain the JSP page. Custom tags were introduced to overcome the problems faced in using scriptlets. Although custom tags proved to be a better choice than scriptlets they had certain limitations too Web designers had to spend a lot of time in coding packaging and testing these tags before using them. This meant that Web designers were often left with little time to concentrate on the designing of Web pages. The introduction of JSTL has helped Web designers overcome the shortcomings of custom tags, by encapsulating the common functionalities that the Web designer may need to develop Web pages.

Features of JSTL

JSTL aims to provide an easy way to maintain SP pages. The use of tags defined in JSTL has Simplified the task of the designers to create Web pages. They can now simply use a tag related to the task that they need to implement in a JSP page. The main features of JSTL are as follows:

- Provides support for conditional processing and Uniform Resource Locator (URL)-related actions to process URL resources in a JSP page. You can also use the JSTL core tag library that provides iterator tags used to easily iterate through a collection of objects.
- Provides the XML tag library, which helps you to manipulate XML documents and perform actions related to conditional and iteration processing on parsed XML documents.
- Enables Web applications to be accessed globally by providing the internationalization tag library Internationalization means that an application can be created to adapt to various locales so that people of different regions can access the application in their native languages The internationalization tag library makes the implementation of localization in an application easy, fast and effective.

- Enables interaction with relational databases by using various SQL commands Web applications require databases to store information required for the application, which can be manipulated by using the SQL tag library provided by JSTL.
- Provides a series of functions to perform manipulations, such as checking whether an input String contains the substring specified as a parameter to a function or returning the number of items in a collection, or the number of characters in a 5tring These functions can be used in an El expression and are provided by the functions tag library.

Tag Libraries in JSTL

A tag library provides a number of predefined actions that functionalities to a specific JSP page. JSTL provides tag libraries that include a wide range of actions to perform common tasks. For example, if you want to access data from database, you can use SQL tag library in your applications. JSTL is a standard tag library that is composed of five tag libraries. Each of these tag libraries represents separate functional area and is used with a prefix. Below table describes the tag libraries available in JSTL.

NAME OF THE TAG LIBRARY	FUNCTION	URI	PREFIX
Core tag library	variable support Flow Control Iterator URL management Miscellaneous Core	http://java.sun.com/jsp/jstl/core	c
xml tag library	Flow Control Transformation Locale	http://java.sun.com/jsp/jstl/xml	x
Internationalization tag library	Message Formatting Number and date formatting	http://java.sun.com/jsp/jstl/fmt	fmt
SQL tag library	Database manipulation	http://java.sun.com/jsp/jstl/sql	sql
Functions tag library	Collection length String manipulation	http://java.sun.com/jsp/jstl/functions	fn

JSTL Core

**

The <c:out> tag is similar to JSP expression tag, but it can only be used with expression. It will display the result of an expression, similar to the way < %=...% > work.

Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:out value="${'Welcome to javaTpoint'}"/>
</body>
</html>
```

**

The **<c:import>** tag provides all functionalities of the **<include>** action but also allows for the inclusion of absolute URLs. It has an additional feature of including the content of any resource either within server or outside the server.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Tag Example</title>
</head>
<body>
<c:import var="data" url="http://www.javatpoint.com"/>
<c:out value="${data}"/>
</body>
</html>
```

**

The **<c:set>** tag is JSTL-friendly version of the **setProperty** action. It is used to set the result of an expression evaluated in a 'scope'. The tag is helpful because it evaluates an expression and uses the results to set a value of a JavaBean. This tag is similar to jsp:setProperty action tag.

Let's see the simple example of <c:set> tag:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="Income" scope="session" value="${4000*4}"/>
<c:out value="${Income}"/>
</body>
</html>
```

Output:

16000

**

The < c:if > tag is used for testing the condition and it display the body content, if the expression evaluated is true. It is a simple conditional tag which is used for evaluating the body content, if the supplied condition is true.

example:

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<c:if test="${income > 8000}">
   <p>My income is: <c:out value="${income}"/><p>
```

```
</c:if>
</body>
</html>
```

Output:

My income is: 16000

**

The < c:choose > tag is a conditional tag that establish a context for mutually exclusive conditional operations. It works like a Java **switch** statement in which we choose between a numbers of alternatives.

The <c:when > is subtag of <choose > that will include its body if the condition evaluated be 'true'.

The < c:otherwise > is also subtag of < choose > it follows &l;twhen > tags and runs only if all the prior condition evaluated is 'false'.

The c:when and c:otherwise works like if-else statement. But it must be placed inside c:choose tag.

Example

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="income" scope="session" value="${4000*4}"/>
<p>Your income is : <c:out value="${income}"/></p>
<c:choose>
   <c:when test="${income <= 1000}">
      Income is not good.
   </c:when>
   <c:when test="${income > 10000}">
       Income is very good.
   </c:when>
   <c:otherwise>
      Income is undetermined...
   </c:otherwise>
```

</c:choose>
</body>
</html>

This will produce the following result:

1. Your income is : 16000
2. Income is very good.

Even/Odd Example using c:when and c:otherwise

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<h1>JSTL c:when, c:otherwise, c:choose</h1>

<c:set value="10" var="num"></c:set>
<c:choose>
<c:when test="${num%2==0}">
<c:out value="${num} is even number"></c:out>
</c:when>
<c:otherwise>
<c:out value="${num} is odd number"></c:out>
</c:otherwise>
</c:choose>

</body>
</html>
```

Output:

10 is even number

**

The <c:for each > is an iteration tag used for repeating the nested body content for fixed number of times or over the collection.

These tag used as a good alternative for embedding a Java **while, do-while, or for** loop via a scriptlet. The < c:for each > tag is most commonly used tag because it iterates over a collection of object.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:forEach var="j" begin="1" end="3">
  Item <c:out value="${j}"/><p>
</c:forEach>
</body>
</html>
```

Output:

Item 1
Item 2
Item 3

**

S.No.	Function & Description
1	fn:contains() Tests if an input string contains the specified substring.
2	fn:containsIgnoreCase() Tests if an input string contains the specified substring in a case insensitive way.
3	fn:endsWith()

	Tests if an input string ends with the specified suffix.
4	fn:escapeXml() Escapes characters that can be interpreted as XML markup.
5	fn:indexOf() Returns the index withing a string of the first occurrence of a specified substring.
6	fn:join() Joins all elements of an array into a string.
7	fn:length() Returns the number of items in a collection, or the number of characters in a string.
8	fn:replace() Returns a string resulting from replacing in an input string all occurrences with a given string.
9	fn:split() Splits a string into an array of substrings.
10	fn:startsWith() Tests if an input string starts with the specified prefix.
11	fn:substring() Returns a subset of a string.
12	fn:substringAfter() Returns a subset of a string following a specific substring.
13	fn:substringBefore() Returns a subset of a string before a specific substring.
14	fn:toLowerCase() Converts all of the characters of a string to lower case.
15	fn:toUpperCase()

	Converts all of the characters of a string to upper case.
16	fn:trim() Removes white spaces from both ends of a string.

**

The **fn:contains()** function determines whether an input string contains a specified substring.

Syntax

The fn:contains() function has the following syntax −

boolean contains(java.lang.String, java.lang.String)

Example

Following example explains the functionality of **fn:contains()** function −

```
<%@ 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 = "theString" value = "I am a test String"/>

    <c:if test = "${fn:contains(theString, 'test')}">
      <p>Found test string<p>
    </c:if>

    <c:if test = "${fn:contains(theString, 'TEST')}">
      <p>Found TEST string<p>
    </c:if>

  </body>
</html>
```

You will receive the following result −

Found test string

**

The **fn:containsIgnoreCase()** function determines whether an input string contains a specified substring. While doing search it ignores the case.

Syntax

The **fn:containsIgnoreCase()** function has the following syntax −

boolean containsIgnoreCase(java.lang.String, java.lang.String)

Example

Following is the example to explain the functionality of the **fn:containsIgnoreCase()** function −

```
<%@ 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 = "theString" value = "I am a test String"/>

    <c:if test = "${fn:containsIgnoreCase(theString, 'test')}">
      <p>Found test string<p>
    </c:if>

    <c:if test = "${fn:containsIgnoreCase(theString, 'TEST')}">
      <p>Found TEST string<p>
    </c:if>

  </body>
</html>
```

You will receive the following result −

Found test string
Found TEST string

**

The **fn:endsWith()** function determines if an input string ends with a specified suffix.

Syntax

The **fn:endsWith()** function has the following syntax −

boolean endsWith(java.lang.String, java.lang.String)

Example

Following example explains the functionality of the **fn:substring** function −

```
<%@ 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 = "theString" value = "I am a test String 123"/>

    <c:if test = "${fn:endsWith(theString, '123')}">
      <p>String ends with 123<p>
    </c:if>

    <c:if test = "${fn:endsWith(theString, 'TEST')}">
      <p>String ends with TEST<p>
    </c:if>

  </body>
</html>
```

You will receive the following result −

String ends with 123

The **fn:indexOf()** function returns the index within a string of a specified substring.

Syntax

The **fn:indexOf()** function has the following syntax −

int indexOf(java.lang.String, java.lang.String)

Example

Following is the example to explain the functionality of the **fn:indexOf()** function −

```
<%@ 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 = "This <abc>is second String.</abc>"/>
    <p>Index (1) : ${fn:indexOf(string1, "first")}</p>
    <p>Index (2) : ${fn:indexOf(string2, "second")}</p>

  </body>
</html>
```

You will receive the following result −

Index (1) : 8
Index (2) : 13

**

The **fn:join()** function concatenates all the elements of an array into a string with a specified separator.

Syntax

The **fn:join()** function has the following syntax −

String join (java.lang.String[], java.lang.String)

Example

Following is the example to explain the functionality of the **fn:join()** function −

```
<%@ 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 = "${fn:split(string1, ' ')}" />
    <c:set var = "string3" value = "${fn:join(string2, '-')}" />
    <p>Final String : ${string3}</p>

  </body>
</html>
```

Note − The **fn:split()** function returns an array splitting into different elements.

You will receive the following result −

Final String : This-is-first-String.

**

The **fn:length()** function returns the string length or the number of items in a collection.

Syntax

The **fn:length()** function has the following syntax −

int length(java.lang.Object)

Example

Following example explains the functionality of the **fn:length()** function −

```
<%@ 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 = "This is second String." />
    <p>Length of String (1) : ${fn:length(string1)}</p>
    <p>Length of String (2) : ${fn:length(string2)}</p>
  </body>
</html>
```

You will receive the following result −

Length of String (1) : 21
Length of String (2) : 22

**

The **fn:replace()** function replaces all occurrences of a string with another string.

Syntax

The **fn:replace ()** function has the following syntax −

boolean replace(java.lang.String, java.lang.String, java.lang.String)

Example

Following is the example to explain the functionality of the **fn:replace()** function −

```
<%@ 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:replace(string1, 'first', 'second')}" />
    <p>Final String : ${string2}</p>
  </body>
</html>
```

You will receive the following result −

Final String : This is second String.