JSTL

1. Give the relation between Jakarta Apache taglib project and JSTL?
    a. The Jakarta Apache taglib project attempted to define a common standard and implementation for a universally useful set of tag libraries. These contain functionality that is common to practically every JSP development. Such was the popularity of these libraries that their tag definitions have been officially adopted as part of the JavaServer Page specification by Sun as the JavaServer Page Standard Tag Library—or JSTL.
2. List down the tag libraries that JSTL contains?
    a. The JSTL comprises five tag libraries:
        i. core: custom actions that do the programming “grunt work”—such as conditions and loops—and also fundamental JSP tasks such as setting attributes, writing output, and redirecting to other pages and resources
        ii. xml: custom actions that alleviate much of the work in reading and writing XML files
        iii. sql: custom actions dedicated to database manipulation
        iv. fmt: custom actions for formatting dates and numbers, and for internationalization of text
        v. function: a set of standardized EL functions.
3. Give the JSTL versions that go with JSP 1.2 and JSP 2.0 respectively?
    a. JSTL, which is 1.0 goes with JSP level 1.2 and 1.1 version of JSTL goes with JSP level 2.0.
4. Will JSP containers (such as Tomcat) come with an implementation of JSTL?
    a. JSP containers (such as Tomcat) don’t necessarily come with one already supplied. Fortunately, it’s easy and free to acquire an implementation, which is also easy to install into most containers. To get hold of the standard reference implementation, visit: http://jakarta.apache.org/taglibs/binarydist.html.
5. Give the taglib directive to include the core JSTL library?
    a. <%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
    b. To make use of the core tag library in your own JSP pages, you must include a taglib directive (or namespace reference) containing the right URI:

        i. http://java.sun.com/jsp/jstl/core

    You can choose whichever prefix you like to use with these tags, although popular convention suggests the use of “c.”

6. How the JSTL specification does splits the core JSTL actions into groups?
    a. General Purpose
        i. <c:out>
        ii. <c:set>
        iii. <c:remove>
        iv. <c:catch>
    b. Conditional
        i. <c:if>
        ii. <c:chose>
        iii. <c:when>
        iv. <c:otherwise>
    c. Iteration
        i. <c:forEach>
        ii. <c:forTokens>
    d. URL-Related
        i. <c:import>
        ii. <c:url>
        iii. <c:redirect>
        iv. <c:param>
7. Give the use of <c:out> core JSTL action?
    a. This action is used for writing expressions and template text to page output. <c:out> has a couple of good features:
        i. The provision of a default value if the main value expression evaluates to null. There are two ways to do this:
            1. <c:out value=${user.name} default="User name nt recognized" />
            2. <c:out value=${user.name}>User name not recognized</c:out>
        ii. The ability to escape XML-unfriendly characters, such as < and >. Entities are substituted instead, such as &lt; and &gt;. Whatever characters go to

output—whether in the value or the default value—they are escaped in this way. This feature is enabled by default but can be invoked explicitly:

1. <c:out value=${xmlUnfriendlyAttribute} escapeXml="true" />

Or switched off:

2. <c:out value=${xmlUnfriendlyAttribute} escapeXml="false" />

In the body we can even have a <jsp:include> standard action to include a JSP file:

1. <c:out value="${null}" escapeXml="true">
2. <jsp:include page="xmlUnfriendly.jsp" />
3. </c:out>

8. Give an advantage of using <c:out value="${person.name} instead of ${person.name}?
   a. Using cout will avoid cross-site scripting. By setting escapeXml attribute of <c:out> to true, it will escape XML-unfriendly characters, such as < and > with &lt; and &gt;. If person.name = <script>alert("Yo")</script> the script will be executed in the second case, but not when using c:out. The default value of escapeXml attribute of <c:out> is true.
9. Are there any mandatory attributes for <c:out>?
   a. value is mandatory.
10. Give the basic use and usage of <c:set> core JSTL action?
    a. This is used for setting attributes in any scope. The var attribute is used to name a variable, and the value attribute sets a value for that variable. You can even use an EL expression to set the value instead of a literal. As an alternative to using the value attribute, you can place the value in the body of the tag.
       i. <c:set value="9.21" var="numerator" />
       ii. <c:set value="3" var="denominator" />
       iii. <c:set value="${numerator/denominator}" var="calculationResult" />
       iv. <c:set var="calculationResult">${numerator/denominator}</c:set>
    b. Any available object can be the object of a <c:set> action, provided that the object can be constructed as a Java Bean. The target attribute specifies the bean under consideration. The property attribute specifies the name of the property.

        i. <c:set value="28" target="${pageContext.session}" property="maxInactiveInterval" />
        Here, session is available in EL through the implicit variable pageContext. The property attribute specifies the name of the property, maxInactiveInterval.
        ii. We can modify the value of a property on a target object.
            1. **<c:set target**=”obj_target” **property**=”prop_name” **value**=”value”/>
        We can also provide the value inside the body omitting the value attribute. In general the value of the target attribute will be a request time expression (either scriptlet or EL) which resolves to a valid JavaBeans or a Map object.
    c. If you want to set up attributes in different scopes, <c:set> has a scope attribute for the purpose:
        i. <c:set var="calculationResult" scope="session">${num/den}</c:set>

11. Give the use and usage of <c:remove> core JSTL action?
    a. 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). The action has only two attributes—var to name the variable, and the optional scope to specify the scope. As always, page is the default when scope is not specified:
        i. <c:remove var="calculationResult" />
        ii. <c:remove var="calculationResult" scope="session" />

12. Give the use and usage of <c:catch> core JSTL action?
    a. This action allows you to catch an error in your page, without it propagating to (for example) an error page defined in web.xml. It’s more like a try/catch rolled into one, where the catch does nothing: All errors are suppressed. The exception caught is a java.lang. Throwable. The <c:catch> action relates only to anything that goes on in its body.
        i. <c:catch>
        ii. <jsp:scriptlet>int zero = 0; out.write(3/zero);</jsp:scriptlet>
        iii. </c:catch>

   b. You can specify a named, page-scope attribute to represent the exception, and access this after the <c:catch> block. You use the var attribute to do this (there is no scope attribute; the exception dies with the page).
      i. <c:catch var="numException">
      ii. <jsp:scriptlet>int zero = 0; out.write(3/zero);</jsp:scriptlet>
      iii. </c:catch>
      iv. <p>If there was an exception, the message is: ${numException.message}</p>

13. List down the attributes of <c:out> core JSTL action? Also give the type and default value (if any)?
    a. <u>Mandatory</u>
       i. **value** (Type: Object, Default Value: None)
    b. <u>Optional</u>
       i. **escapeXML** (Type: boolean, Default Value: ***True***)
       ii. **default** (Type: Object, Default Value: ***Empty String***)
14. List down the attributes of <c:set> core JSTL action? Also give the type and default value (if any)?
    a. There are no mandatory parameters.
    b. Optional parameters are:
       i. **value** (Type: Object, Default Value: None)
       ii. **var** (Type: String, Default Value: None)
       iii. **scope** (Type: String, Default Value: ***Page***)
       iv. **target** (Type: Object, Default Value: None)
       v. **property** (Type: String, Default Value: None)
15. List down the attributes of <c:remove> core JSTL action? Also give the type and default value (if any)?
    a. Mandatory
       i. **var** (Type: String, Default Value: None)
    b. Optional
       i. **scope** (Type: String, Default Value: The first scope where the variable is found.)

16. List down the attributes of <c:catch> core JSTL action? Also give the type and default value (if any)?
    a. There are no mandatory parameters.
    b. Optional parameter is:
        i. **var** (Type: String, Default Value: None)
17. Give the use and usage of <c:if> core JSTL action?
    a. This action evaluates its body content if a condition is true. The condition is expressed in EL as the value for the test attribute.
        i. <c:if test="${user.loyaltyPoints gt 1000}">
        ii. <p>Welcome to one of our best customers!</p>
        iii. </c:if>
    b. You can store the result of the test in a scoped variable using the var and scope attributes.
        i. <c:if test="${mytags:checkRole(user, 'Manager')}"
        ii. var="userInManagerRole" scope="session" />
18. Give the use and usage of <c:chose> core JSTL action?
    a. In combination with <c:when> and <c:otherwise>, this action works something like a Java switch statement, but slightly differently. The structure is this: The <c:choose> action is just a container for two possible other actions—<c:when> and <c:otherwise>. Between the opening and closing tags for <c:choose>, you can include only white space (any amount) or these two actions. The rules on including <c:when> and <c:otherwise> are as follows:
        i. <c:when>: there must be at least one occurrence.
        ii. <c:otherwise>: optional—but if included, there cannot be more than one occurrence.
    b. Within the body of the <c:choose> each <c:when> test is performed in strict order of appearance. If a test is false, the body of <c:when> is ignored. If a test is true, the body of that <c:when> action is executed. Anything that follows within the <c:choose> action will be ignored once a true test has triggered. If none of the <c:when> tests evaluate to true, then and only then will the body of <c:otherwise> be executed. The <c:otherwise> must be the last action in the <c:choose> group.

   c. <c:when> has only a single, mandatory attribute: test. The test expression must evaluate to a boolean true for the body to be executed. If the result evaluates to non-boolean, the expression will be treated as equivalent to a boolean false.
   d. <c:otherwise> has no attributes.

19. Two attributes (if present) that does not allow Run-time Expression?
   a. var
   b. scope

20. Give the uses and usages of <c:forEach> ?
   a. This has two main uses:
      i. <u>To iterate over a collection of objects.</u> EL helpfully provides some ready-made implicit variables that are collections of objects—for example, ${headerValues}, which represents the collection of values for request headers. Here is a <c:forEach> loop that displays these values in an HTML table:
         1. <table border="1">
         2. <c:forEach var="hdr" items="${headerValues}">
         3. <tr><td>${hdr.key}</td><td>${hdr.value[0]}</td></tr>
         4. </c:forEach>
         5. </table>

         Each object in a Map is a Map.Entry object, with getKey() and getValue() methods, exposing the two bean-like properties key and value. So the EL syntax ${hdr.key} has the effect of getting the request header's key value and writing this to the current JspWriter. We know (from the EL specification) that each Map.Entry value object in the headerValues Map is a String array. EL allows the use of Java-language such as array syntax (${hdr.value[0]}) to obtain—in this case—the first available value for the named request header.
      ii. <u>To iterate a fixed number of times.</u> It can be used to perform a set number of iterations—much like a Java for loop:
         1. <table border="1">
         2. <c:set var="num" value="1" />

3. <c:forEach begin="1" end="128" step="2">
4. <c:set var="num" value="${num + num}" />
5. <tr><td>${num}</td></tr>
6. </c:forEach>
7. </table>

21. List down the different types allowable for the items attribute of <c:forEach> action?
    a. java.lang.Array (of some type of Object)
        i. Corresponding Type for var Attribute is the declared type for the Array.
    b. java.lang.Array (of primitives)
        i. Corresponding Type for var Attribute is the wrapper type (Integer, Double, etc.) corresponding with the declared primitive type (int, double, etc.) for the array.
    c. java.util.Collection, java.util.Iterator, java.util.Enumeration
        i. Corresponding Type for var Attribute is Whatever type of Object is returned from the underlying Collection, Iterator or Enumeration
    d. java.lang.String
        i. The String in items is split up into a separate Strings for each comma encountered—like the tokens in StringTokenizer.

22. Give the use of <c:forTokens>?
    a. <c:forTokens> is a specialized version of the <c:forEach> action, designed to perform String tokenization. The items attribute of <c:forTokens> will accept only a String as a value. An additional attribute, delims, is used to specify the delimiter to recognize when breaking up the String into tokens.
    b. Example:
        i. <table border="1">
        ii. <c:forEach var="hdr" items="${headerValues}">
        iii. <tr><td>${hdr.key}</td><td>${hdr.value[0]}</td></tr>
        iv. <c:if test="${hdr.key eq 'Accept'}">
        v. <c:set value="${hdr.value[0]}" var="acceptValues" />
        vi. </c:if>
        vii. </c:forEach>

viii. </table>

ix. <table border="1"><tr><th><b>Accept values</b></th></tr>

x. <c:forTokens items="${acceptValues}" delims="," var="value">

xi. <tr><td>${value}</td></tr>

xii. </c:forTokens>

xiii. </table>

23. List down the attributes for <c:foreach> ?
    a. There are no mandatory parameters. Also there are no default values for any attributes.
    b. Optional parameters are:
        i. Var (Type: String)
        ii. Items (Type: Object)
        iii. varStatus (Type: String)
        iv. begin (Type: int)
        v. end (Type: int)
        vi. step (Type: int)
24. List down the attributes for <c:forTokens>?
    a. There are no default values for any attributes.
    b. Mandatory parameters are:
        i. items (Type: String)
        ii. delims (Type: String)
    c. Optional parameters are:
        i. var (Type: String)
        ii. varStatus (Type: String)
        iii. begin (Type: int)
        iv. end (Type: int)
        v. step (Type: int)
25. List down the two attributes of <c:foreach> and <c:forTokens> that cannot accept runtime expressions?
    a. var
    b. varStatus.

26. Give the use and usage of <c:import>?

   a. It performs requesttime inclusion of other resources. <c:import> can be used for resources in other contexts on the same server, or even to other servers entirely. The result of the import doesn't have to be sent directly to page output. Instead, you can store the result for later use.
   b. Example 1: importing a resource from the same application. The only attribute you need to set up is url, which takes a relative or absolute URL value.:
      i. <c:import url="trailer.jsp" />
   c. Example 2: If you want to go outside of the current context, the syntax is this:
      i. <c:import url="/rounding.jspx" context="/examp0801" />

      The context attribute specifies the context root for the resource named in the url attribute. The context must reside (or at least be known to) the same container that houses the context for the file doing the importing. Both the url and context values must begin with a forward slash.
   d. Example 3: Importing from an external resource using an absolute URL and storing the result in a variable:
      i. <c:import url="http://c2.com/index.html" var="importedPage" scope="page" />
      ii. <jsp:scriptlet>String fiftyChars = ((String)
      iii. pageContext.getAttribute("importedPage")).substring(0,50);
      iv. pageContext.setAttribute("fiftyChars", fiftyChars);
      v. </jsp:scriptlet>
      vi. <pre>${fiftyChars}</pre>
   e. An alternative to var is the varReader attribute. With this, you have the opportunity to import a page into a Reader object. You can only make use of the reader within the body of the <c:import> tag—beyond that, it's unavailable.
      i. <c:import url="http://c2.com/index.html" varReader="myReader">
      ii. <!—Make use of myReader in the body of the import action—>
      iii. ...scriptlet goes here...
      iv. </c:import>

27. Give the use and usage of <c:url>?

   a. This works out a URL string for inclusion in page output. By default, the URL generated through this action is sent directly to page output. More usually, you make use of the var attribute to store the URL for later use.
      i. <c:url value="/rounding.jspx" context="/examp0801" var="myLink" />
      ii. <a href="${myLink}">Link to another page in another context</a>
   b. This attribute defaults to the page context—but by using the scope attribute you can place the attribute holding the URL in any scope you like.
   c. <c:url> is an improvement because it seamlessly rewrites URLs whenever necessary to include the session ID.

28. Give the use and usage of <c:redirect>?
   a. This causes the client to redirect to a specified URL. This action uses the HttpServletResponse.sendRedirect() method under the covers. There's one compulsory attribute (url) and one optional one (context). The URL can be a complete URL, including the protocol and host name and port number:
      i. <c:redirect url="http://localhost:8080/examp0801/rounding.jspx" />
   b. url can have a value that begin with a forward slash and also that doesn't begin with a forward slash.
      i. <c:redirect url="iteration.jspx" />
      ii. <c:redirect url="/iteration.jspx" />
   c. There's also an option to supply another context on the same server, by using the context attribute. Both the url and context values must begin with a forward slash.
      i. <c:redirect url="/rounding.jspx" context="/examp0801" />
   d. Like the <c:url> action, <c:redirect> has the capacity to rewrite URLs to include the session ID—when it needs to.

29. Give the use and usage of <c:param>?
   a. Used to attach parameters to any of the three URL actions <c:import>, <c:url>, or <c:redirect>.
      i. <c:url value="/rounding.jspx" context="/examp0801" var="myLink">
      ii. <c:param name="firstName" value="David" />
      iii. <c:param name="secondName">Bridgewater</c:param>
      iv. </c:url>

    v. <a href="${myLink}">Link to another page in another context</a>

30. List down the attributes of <c:import>?
    a. Mandatory parameter is url.
    b. Optional parameters are:
        i. context
        ii. var
        iii. scope (Default is page)
        iv. charEncoding (Default is ISO-8859-I)
        v. varReader
    c. url, context and charEncoding supports runtime expressions while var, scope and varReader doesn't support runtime expressions.
31. List down the attributes for <c:url>?
    a. There are no mandatory attributes.
    b. Optional Parameters are;
        i. value
        ii. context
        iii. var
        iv. scope (Default is page)
    c. value, context supports runtime expressions while var, scope doesn't support runtime expressions.