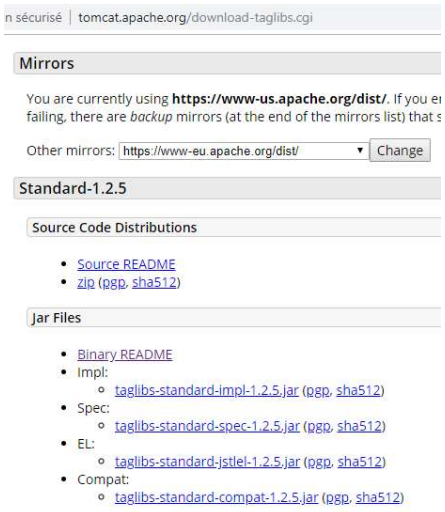
Chapitre 5 : JSTL

Introduction

- JSTL = Java Server Page Standard Tag Library
  - Ensemble de balises prédéfinies organisées en un ensemble de bibliothèques
  - Permet facilement d'accéder et manipuler les données de l'application sans scriptlets.
  - Plus facile à lire car JSTL est basé sur XML, qui est similaire à HTML.
- But : se passer du code Java au sein des pages JSP

Installation

- JSTL fait partie de la spécification Jakarta EE
- Il est par conséquent implémenté par tous les conteneurs de servlet
- Tomcat fournit une implémentation téléchargeable à l'adresse : http://tomcat.apache.org/download-taglibs.cgi
- Les bibliothèques téléchargées doivent être placés dans le dossier « WEB-INF/lib »



Types de JSTL

- JSTL fournit quatre types de bibliothèques de balises :
  - Core JSTL
  - XML Tag Library
  - Format Tag Library
  - SQL Tag Library
  - Functions Tag Library
- Il propose aussi un langage nommé EL (Expression Language) qui permet de manipuler des objets Java accessibles dans les différentes pages JSP.

Langage EL : Introduction

- EL= Expression Language
- Est un langage de script qui permet d'accéder d'une façon plus simple aux objets Java accessibles dans les différents contextes de la page JSP.
- La syntaxe de base est ${nomVariable}
- **Exemple** : accès à l'attribut nom d'un objet « etudiant » situé dans la session
  - Avec Java:<%= session.getAttribute("etudiant").getNom() %>
  - Avec EL: ${sessionScope.etudiant.nom}

Langage EL : Objets implicites

- PageScope : variables couvertes par la portée de la page (correpond à PageContext dans JSP)
- RequestScope : variables couvertes par la portée de la requête (HttpServletRequest)
- SessionScope : variables couvertes par la portée de la session (HttpSession)
- ApplicationScope : variables couvertes par la portée de l'application (ServletContext)
- Param : paramètres de la requête HTTP
- ParamValues : paramètres de la requête sous forme d'une collection

Langage EL : Objets implicites

- Header : en-tête de la requête
- HeaderValues : en-têtes de la requête sous forme d'une collection
- InitParam : Paramètres d'initialisation du contexte
- Cookie : valeurs du cookie
- PageContext : correspond à l'objet PageContext de la page en cours

Langage EL : Operateurs de base

- **.** : permet d'obtenir une propriété d'un objet,
  - Exemple ${param.nom}
- [ ] : permet d'obtenir une propriété par son nom ou son indice.
  - Exemple : ${param[" nom "]}, ${row[1]}
- **empty** : Teste si les valeurs de variables sont vides.
  - Exemple : ${empty param.nom}
- **==** ou **eq** : teste l'égalité de deux objets
- **!=** ou **ne** : teste l'inégalité de deux objets

Langage EL : Operateurs de base

- **<** ou **lt** : test strictement inférieur
- **>** ou **gt** : test strictement supérieur
- **<=** ou **le** : test inférieur ou égal
- **>=** ou **ge** : test supérieur ou égal
- **+** : Addition , **-** : Soustraction, ***** : Multiplication, **/** ou **div** : Division
- **%** ou **mod** : Modulo
- **&&** ou **and** : conjonction
- **||** ou **or** : disjonction
- **!** ou **not** : Négation d'une valeur

Langage EL : variables locales

- Le langage EL ne permet pas l'accès aux variables locales.
- Pour pouvoir les utiliser, il faut obligatoirement en créer une copie dans une des portées particulières : page, request, session ou application.
- Exemple :

```
<% int x= 2019;
     int y=2020;
    pageContext.setAttribute("y", new Integer(y));
%>
Valeur de x = <c:out value="${x}" /><BR/>
Valeur de y = <c:out value="${y}" /><BR/>
```

- Exécution :
  - Valeur de x = // x ne sera pas affiché car c'est une variable locale
  - Valeur de y = 2020

Core JSTL: introduction

- Propose un ensemble de tags pour l'itération, le traitement conditionnel et le langage d'expression.
- Au niveau des pages JSP, la déclaration se fait comme suit:
  - <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
- Ces tags sont répartis en 3 catégories :
  - Pour le langage EL : set, out, remove, catch
  - Pour les conditions et itérations : if, choose, forEach, forTokens
  - Pour la gestion des URL : import, url, redirect

Core JSTL : balise <c:set>

- Permet de stocker une variable dans une portée particulière (page, requête, session ou application).
- Dispose des attributs suivants:
  - **var** : nom de la variable qui va stocker la valeur
  - **value** : valeur à stocker
  - **scope** : portée de la variable qui va stocker la valeur
  - **target** : nom de la variable contenant un bean dont la propriété doit être modifiée
  - **property** : nom de la propriété à modifier

Core JSTL : balise <c:set>

- Exemples :
  - <c:set var="a" value="valeur1" scope="page" />
  - <c:set var="b" value="valeur2" scope="request" />
  - <c:set var="c" value="valeur3" scope="session" />
  - <c:set var="d" value="valeur4" scope="application" />
- Remarque 1 : la valeur peut être déterminée dynamiquement :
  - <c:set var="e" value="${param.id}" scope="page" />
- Remarque 2 : La valeur de la variable peut être précisée dans le corps de la balise
  - <c:set var="maVariable" scope="page">Valeur de ma variable</c:set>

Core JSTL : balise <c:out>

- Est utilisée pour afficher les valeurs contenues dans variables ou le résultat d'une expression implicite de la même façon que <%=…%>
- Attributs :
  - **Value** : valeur à afficher (obligatoire)
  - **Default** : définir une valeur par défaut si la valeur est null
  - **escapeXml** : booléen qui permet de convertir les caractères spéciaux à leurs codes correspondants
- Elle offre en plus la possibilité d'accès aux propriétés grâce au **"."**

Core JSTL : balise <c:out>

- Exemples :
  - <c:out value='${pageScope.maVariable1}' />
  - <c:out value='${requestScope.maVariable2}' />
  - <c:out value='${sessionScope.maVariable3}' />
  - <c:out value='${applicationScope.maVariable4}' />

Core JSTL : balise <c:out>

- Remarque :
  - Si la protée de la variable n'est pas précisée la variable est recherchée prioritairement dans la page, puis la requête, puis la session et enfin l'application.
  - L'attribut default définit une valeur par défaut si le résultat de l'évaluation de la valeur est null.
  - Si la valeur est null et que l'attribut default est absent alors c'est une chaîne vide qui est renvoyée
- Exemple :
  - <c:out value="${personne.nom}" default="Inconnu" />

Core JSTL : balise <c:out>

- **Remarque** : exemple de génération de code dans un formulaire sans passer par les scriplets.
  - <input type="text" name="nom" value="<c:out value="${param.nom}"/>" />

Core JSTL : balise <c:remove>

- Permet de supprimer une variable d'une portée particulière.
- attributs :
  - Var : nom de la variable à supprimer (obligatoire)
  - Scope : portée de la variable
- Exemples :
  - <c:remove var="maVariable1" scope="page" />
  - <c:remove var="maVariable2" scope="request" />
  - <c:remove var="maVariable3" scope="session" />
  - <c:remove var="maVariable4" scope="application" />

Core JSTL : balise <c:catch>

- Permet de capturer des exceptions qui sont levées lors de l'exécution du code inclus dans son corps.
- Attributs :
  - Var : nom d'une variable qui va contenir des informations sur l'anomalie

Core JSTL : balise <c:catch>

- Exemple :

```
<c:set var="valeur" value="abc" />
<c:catch var="erreur">
  <fmt:parseNumber var="valeurInt" value="${valeur}"/>
</c:catch>
<c:if test="${not empty erreur}">
  la valeur n'est pas numerique
</c:if>
```

- Resultat : la valeur n'est pas numerique

Core JSTL : balise <c:catch>

- Remarque :
  - L'objet désigné par l'attribut var du tag catch possède une propriété message qui contient le message d'erreur
- Exemple :

```
<c:set var="valeur" value="abc" />
<c:catch var="erreur">
  <fmt:parseNumber var="valeurInt" value="${valeur}"/>
</c:catch>
<c:if test="${not empty erreur}">
 <c:out value="${erreur.message}"/>
</c:if>
```

- Résultat : In <parseNumber>, value attribute can not be parsed: "abc"

Core JSTL : balise <c:catch>

- Remarque :
  - Le problème avec cette balise est qu'il n'est pas possible de savoir quelle exception a été levée.

Core JSTL : balise <c:if>

- Évalue une expression et affiche le contenu de son corps uniquement si l'expression est évaluée à Vrai.
- Attributs :
  - test : condition à évaluer
  - var : nom de la variable qui contiendra le résultat de l'évaluation
  - scope : portée de la variable qui contiendra le résultat
- Exemples:
  - <c:if test="${empty personne.nom}" >Inconnu</c:if>
  - <c:if test="${empty personne.nom}" var="resultat" />

```
<%-- le résultat est stocké dans une variable--%>
```


Core JSTL : balise <c:choose>

- Permet d'effectuer un choix parmi plusieurs mutuellement exclusifs
- Ne possède pas d'attributs mais deux balises filles <c:when> et <c:otherwise>
- Exemple :

```
<c:choose>
  <c:when test="${personne.civilite == 'Mr'}">
    Bonjour Monsieur
  </c:when>
  <c:when test="${personne.civilite == 'Mme'}">
    Bonjour Madame
  </c:when>
```

Core JSTL : balise <c:choose>

```
<c:when test="${personne.civilite == 'Mlle'}">
  Bonjour Mademoiselle
 </c:when>
 <c:otherwise>
  Bonjour
 </c:otherwise>
</c:choose>
```


Core JSTL : balises <c:forEach> et <c:forTokens>

- Constituent une alternative au boucles Java dans les scriptlets.
- La balise <c: forEach> itère sur une collection d'objets.
- La balise <c: forTokens> est utilisée pour diviser une chaîne en jetons et les parcourir.
- Attributs communs :
  - **var** : nom de la variable qui contient l'élément en cours de traitement
  - **items** : collection à traiter
  - **varStatus** : nom d'une variable qui va contenir des informations sur l'itération en cours de traitement

Core JSTL : balises <c:forEach> et <c:forTokens>

- **begin** : numéro du premier élément à traiter (le premier possède le numéro 0)
- **end** : numéro du dernier élément à traiter
- **step** : pas des éléments à traiter (par défaut 1)

- Attribut en plus pour <c: forTokens> :
  - **delims** : Caractères à utiliser comme délimiteurs ou séparateurs.
- Exemple pour <c:forEach>

```
<c:forEach var = "i" begin = "1" end = "5">
        élément <c:out value = "${i}"/><p>
 </c:forEach>
```


Core JSTL : balises <c:forEach> et <c:forTokens>

- Exemple pour <c: forTokens> :

```
<c:forTokens items = "Ali,Omar,Taha" delims = "," var = "name">
        <c:out value = "${name}"/><p>
</c:forTokens>
```

- **Remarque** : varStatus permet de définir une variable qui va contenir des informations sur l'itération en cours d'exécution. Cette variable possède plusieurs propriétés :
  - index indique le numéro de l'occurrence dans l'ensemble de la collection
  - count indique le numéro de l'itération en cours (en commençant par 1)
  - first booléen qui indique si c'est la première itération
  - last booléen qui indique si c'est la dernière itération

Core JSTL : balises <c:forEach> et <c:forTokens>

- Exemple :

```
<c:forEach begin="1" end="12" var="i" step="3" varStatus="vs">
  index = <c:out value="${vs.index}"/> :
  count = <c:out value="${vs.count}"/> :
  value = <c:out value="${i}"/>
  <c:if test="${vs.first}">  : Premier element  </c:if>
  <c:if test="${vs.last}">  : Dernier element  </c:if> <br>
</c:forEach>
```

- Résultat :
  - index = 1 : count = 1 : value = 1 : Premier element
  - index = 4 : count = 2 : value = 4
  - index = 7 : count = 3 : value = 7
  - index = 10 : count = 4 : value = 10 : Dernier element

Core JSTL : balise <c:import>

- Permet d'inclure une ressource identifiée par une URL tout comme l'action <jsp :include> mais a l'avanatage de ne pas être limité au contexte de l'application web.
- Attributs :
  - **url** : URL de la ressource (relative à l'application web ou absolue)
  - **var** : nom de la variable qui va stocker le contenu de la ressource sous la forme d'une chaîne de caractères
  - **scope** : portée de la variable qui va stocker le contenu de la ressource
  - **context** : contexte de l'application Web qui contient la ressource (si la ressource n'est pas l'application web courante)

Core JSTL : balise <c:import>

- **charEncoding** : jeu de caractères utilisé par la ressource
- **varReader** : nom de la variable qui va stocker le contenu de la ressource sous la forme d'un objet de type java.io.Reader

- Exemples :
  - Import dircet du contenu de la ressource :

```
<c:import url="/message.txt" /><br>
```

  - Import du contenu de la ressource dans une variable :

```
<c:import url="/message.txt" var="message" />
<c:out value="${message}" /><BR/>
```


Core JSTL : balise <c:redirect>

- Permet de faire une redirection vers une nouvelle URL.
- Les paramètres peuvent être fournis grâce à un ou plusieurs tags fils param.
- Exemple :

```
<c:redirect url="liste.jsp">
  <c:param name="id" value="123"/>
</c:redirect>
```

Core JSTL : balise <c:url>

- Formate une URL en chaîne et la stocke dans une variable.
- C'est une alternative à l'appel de la méthode ***response.encodeURL()***
- Attributs :
  - value : base de l'URL (obligatoire)
  - var : nom de la variable qui va stocker l'URL
  - scope : portée de la variable qui va stocker l'URL
  - context : contexte.
- Admet **param** comme balise fille, cette balise dispose des attributs :
  - **name** : nom du paramètre
  - **value** : valeur du paramètre

Core JSTL : balise <c:url>

- Exemple :
  - <a href = "<c:url value = "/jsp/index.htm"/>">TEST

Core JSTL : Exercices

1) Créer une page jsp qui affiche dans un tableau HTML la table de multiplication du nombre 3 .
2) Créer une page web jsp permettant d'afficher les nombres pairs *compris* entre 1 et 40 en utilisant la JSTL.
3) Développer une page jsp qui prend en paramètre un nombre et calcule s'il est premier ou non

Table de multiplication du nombre 3

3 x 1	=	3
3 x 2	=	6
3 x 3	=	9
3 x 4	=	12
3 x 5	=	15
3 x 6	=	18
3 x 7	=	21
3 x 8	=	24
3 x 9	=	27
3 x 10	=	30

Solution exercice 1

```
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>Table de multiplication du nombre 3</title>
   </head>
   <body>
      <TABLE border="1">
         <CAPTION>  Table de multiplication du nombre 3 </CAPTION>
         <TBODY>
```

Solution exercice 1 (suite)

```
<c:forEach var = "i" begin = "1" end = "10">
            <TR><TD> 3 x <c:out value = "${i}"/> </TD> <TD> = </TD>
                  <TD> <c:out value = "${3*i}"/> </TD>
             </TR>
          </c:forEach>
       </TBODY>
    </TABLE>
</body>
</html>
```


Solution exercice 2

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
    <head>
         <title>Les nombres pairs compris entre 1 et 40</title>
    </head>
    <body>
        <c:forEach var="i" begin="1" end="40" step="1">
            <c:if test="${i%2 == 0}">
                  <c:out value="${i}" /><br />
            </c:if>
        </c:forEach>
    </body>
</html>
```

Solution exercice 3

```
<!DOCTYPE html>
<html>
   <head>
      <title>Fichier index.html</title>
   </head>
   <body>
      <form method="post" action="traitement.jsp">
         <label>Entrer un nombre : </label>
         <input type="text" name="nombre" />
         <input type="submit" value="valider" />
      </form>
   </body>
</html>
```


Solution exercice 3 (suite)

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
   <head>
      <title>Fichier traitement.jsp</title>
   </head>
   <body>
      <c:set var="nbr" value="${param.nombre}" />
      <c:set var="estPremier" value ="${true}" />
      <c:forEach var="i" begin="${2}" end="${nbr/2}">
         <c:if test="${nbr % i == 0 && nbr != i}">
            <c:set var="estPremier" value ="${false}" />
         </c:if>
      </c:forEach>
```

Solution exercice 3 (suite)

```
<!-- Affichage du résultat-->
      <c:choose>
        <c:when test="${estPremier == true}">
          <c:out value="${nbr}"/> est premier <br/>
        </c:when>
        <c:when test="${estPremier == false}">
          <c:out value="${nbr}"/> n'est pas premier <br/>
        </c:when>
      </c:choose>
   </body>
</html>
```


XML JSTL: introduction

- Elle est utilisée pour le traitement des données XML comme l'analyse, la sélection et la transformation dans une page JSP.
- Syntaxe de la balise principale:
  - `<%@ taglib uri = "http://java.sun.com/jsp/jstl/xml" préfixe = "x"%>`

XML JSTL

- Exemple de fichier XML :

```
<etudiants>
  <etudiant id="1">
    <nom>Alaoui</nom>
    <prenom>Ali</prenom>
  </etudiant>
  <etudiant id="2">
    <nom>Tahiri</nom>
    <prenom>Hassan</prenom>
  </etudiant>
  <etudiant id="3">
    <nom>Omari</nom>
    <prenom>Omar</prenom>
  </etudiant>
</etudiants>
```


XML JSTL

- S'appuie sur le langage de requêtes XPath pour naviguer dans un document XML et localiser un nœud particulier.
- Exemples d'expressions Xpath appliquées sur l'exemple précédent:
  - **/** : sélectionne tout le document, y compris le doctype <?xml version="1.0"?>
  - **//etudiant** : sélectionne tous les éléments "etudiant" du document
  - **//etudiant[@id='2']** : sélectionne l'étudiant qui a l'attribut id égal à 2

XML JSTL : Catégories de balises XML

- Catégories de balises XML JSTL :
  - Fondamentale : parse, set, out
  - Gestion du flux (condition et itération) : if, choose, forEach
  - Transformation XSLT : transform

XML JSTL : balise <x:parse>

- Permet d'analyser un document et de stocker le résultat dans une variable qui pourra être exploitée par la JSP.
- Attributs :
  - xml : contenu du document à analyser
  - var : nom de la variable qui va contenir l'arbre DOM généré par l'analyse
  - scope : portée de la variable qui va contenir l'arbre DOM
  - varDom : variable de type Document pour le document XML analysé
  - scopeDom : portée de la variable varDom
  - filter : filtre à appliquer sur le document source
  - system : URI du document XML en cours d'analyse

XML JSTL : balise <x:parse>

- Exemple :
  - <c:import url="/etudiants.xml" var="etudiants" />
  - <x:parse xml="${etudiants}" var="listetudiants" />
- Remarque :
  - Dans cet exemple, le fichier etudiants.xml se trouve dans le dossier racine de l'application Web.

XML JSTL : balise <x:set>

- Equivalent au tag set de la bibliothèque Core.
- Il permet d'évaluer l'expression Xpath fournie dans l'attribut select et de placer le résultat de cette évaluation dans une variable.
- L'attribut var permet de préciser la variable qui va recevoir le résultat de l'évaluation sous la forme d'un noeud de l'arbre du document XML.
- Attributs :
  - select : expression XPath à évaluer
  - var : nom de la variable qui va stocker le résultat de l'évaluation
  - scope : portée de la variable qui va stocker le résultat

XML JSTL : balise <x:set>

- Exemple :
  - <c:import url="/etudiants.xml" var="etudiants" />
  - <x:parse xml="${etudiants}" var="listetudiants" />
  - <x:set var="unetudiant" select="$listetudiants/etudiants/etudiant[@id=2]" />
  - <h1>nom = <x:out select="$unetudiant/nom"/></h1>

XML JSTL : balise <x:out>

- Equivalent à celui de la bibliothèque Core
- Affiche le résultat d'une expression Xpath
- Attributs :
  - select : expression XPath à évaluer
  - escapeXML : true par défaut, si le contenu comprend des tags HTML, XML ou autres ils seront affichés tels quels. A false c'est l'évaluation des tags qui sera affichée
- Exemple :
  - Afficher le nom de l'étudiant dont l'id est 2

XML JSTL : balise <x:out>

```
<c:import url="/etudiants.xml" var="etudiants" />
<x:parse xml="${etudiants}" var="listetudiants"  />
<x:set var="unetudiant" select="$listetudiants/etudiants/etudiant[@id=2]" />
<h1><x:out select="$unetudiant/nom"/></h1>
```

- Remarque : Pour stocker le résultat de l'évaluation d'une expression dans une variable, il faut utiliser une combinaison du tag x:out et c:set
- Exemple :

```
<c:set var="etudiantId">
    <x:out select="$listtudiants /etudiants/etudiant[@id=2]" />
</c:set>
```


XML JSTL : balise <x:if>

- évalue une expression XPath, si elle est vraie, traite son corps sinon l'ignore.
- Attributs :
  - select : expression XPath à évaluer sous la forme d'un booléen
  - var : nom de la variable qui va stocker le résultat de l'évaluation
  - scope : portée de la variable qui va stocker le résultat de l'évaluation

XML JSTL : balise <x:if>

- Exemple :

```
<%@ 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 x:if Tags</title> </head>
  <body>
    <h3> Info livres:</h3>
    <c:set var = "xmltext">
      <livres>
        <livre>
          <nom>PHP</nom>
          <auteur>Ali</auteur>
          <prix>100</prix>
        </livre>
```


XML JSTL : balise <x:if>

```
 <livre>
   <nom>Java</nom>
   <auteur>Omar</auteur>
   <prix>2000</prix>
 </livre>
 </livres>
</c:set>
 <x:parse xml = "${xmltext}" var = "output"/>
 <x:if select = "$output//livre"> Le Document a au moins un element<livre>. </x:if> <br />
 <x:if select = "$output/livres[1]/livre/prix > 100"> les prix des livres sont très élévés </x:if>
 </body>
</html>
```

XML JSTL : balise <x:choose>

- Permet d'effectuer un choix parmi plusieurs mutuellement exclusifs
- Ne possède pas d'attributs mais deux balises filles <x:when> et <x:otherwise>
- La balise <x:when> a un seul attribut qui est :
  - select : condition à évaluer
- Exemple :

XML JSTL : balise <x:choose>

```
<c:set var = "xmltext">
  <livres>
    <livre>
       <nom>Programmer en Java</nom>
       <auteur>Ali</auteur>
       <prix>100</prix>
    </livre>
    <livre>
        <nom>Maitriser Java EE</nom>
        <auteur>Omar</auteur>
        <prix>2000</prix>
     </livre>
  </livres>
</c:set>
```

XML JSTL : balise <x:choose>

```
<x:parse xml = "${xmltext}" var = "f"/>
<x:choose>
  <x:when select = "$f//livre/auteur = 'Ali'">
     Livre écrit par Ali
  </x:when>
  <x:when select = "$f//livre/auteur = 'Omar'">
     Livre écrit par Omar
  </x:when>
  <x:otherwise>
    Auteur inconnu
  </x:otherwise>
</x:choose>
```


XML JSTL : balise <x:forEach>

- Est utilisée pour boucler sur les nœuds d'un document XML.
- Attributs :
  - select : expression XPath à évaluer
  - var : nom de la variable pour stocker l'élément en cours pour chaque boucle
  - begin : index de début pour l'itération
  - end : index de fin pour l'itération
  - step : taille de l'incrément d'index lors d'une itération sur la collection
  - varStatus : nom de la variable dans laquelle le statut de l'itération est stocké

XML JSTL : balise <x:forEach>

- Exemple

```
<c:set var = "xmltext">
 <livres>
  <livre>
   <nom>PHP</nom>
    <auteur>Ali</auteur>
    <prix>100</prix>
  </livre>
  <livre>
    <nom>Java</nom>
    <auteur>Omar</auteur>
    <prix>2000</prix>
  </livre>
 </livres>
</c:set>
```


XML JSTL : balise <x:forEach>

```
<x:parse xml = "${xmltext}" var = "output"/>
<ul class = "list">
  <x:forEach select = "$output/livres/livre/nom" var = "item">
     <li> nom livre: <x:out select = "$item" /></li>
  </x:forEach>
</ul>
```

XML JSTL : balise <x:transform>

- Permet d'appliquer une transformation XSLT à un document XML.
- Attributs :
  - xslt : feuille se style XSLT (obligatoire)
  - xml : nom de la variable qui contient le document XML à traiter
  - var : nom de la variable qui va recevoir le résultat de la transformation
  - scope : portée de la variable qui va recevoir le résultat de la transformation
  - docSystemId : system identifier (URI) pour parser le document XML
  - xsltSystemId : system identifier (URI) parser le document XSLT
  - result : chaîne de caractères qui contient le résultat de la transformation

XML JSTL : balise <x:transform>

- Exemple :
  - <x:transform xml='${docXml}' xslt='${feuilleXslt}'/>
- Remarque : Le document xml à traiter peut être fourni dans le corps de la balise:
- Exemple :

```
<x:transform xslt='${feuilleXslt}'>
<etudiants>
  <etudiant id="1">
    <nom>Alaoui</nom>
....
```

- Remarque : la balise transform peut avoir un ou plusieurs nœuds fils param pour fournir des paramètres à la feuille de style XSLT.

XML JSTL : Exercices

1) Ecrire une page JSP qui prend le fichier XML des étudiants en entrée et le présente dans un tableau HTML comme sur la figure
2) Reprendre l'exercice précédent mais cette fois en confiant la transformation à un fichier XSL

Liste des étudiants :

ID	Nom	Prénom
1	Alaoui	Ali
2	Tahiri	Hassan
3	Omari	Omar

Solution exercice 1

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>Liste des étudiants</title>
   </head>
   <body>
      <c:import var="fichierXML" url="etudiants.xml"/>
      <x:parse var="fichier" doc="${fichierXML}"/>
      <TABLE border="1">
         <CAPTION>  Liste des étudiants : </CAPTION>
```

Solution exercice 1 (suite)

```
<TBODY>
          <TR> <TH> ID </TH> <TH> Nom </TH> <TH> Prénom </TH> </TR>
        <x:forEach select="$fichier/etudiants/etudiant" var="et">
            <TR>
                <TD> <x:out select = "$et/@id"/></TD>
                <TD> <x:out select = "$et/nom"/> </TD>
                <TD> <x:out select = "$et/prenom"/> </TD>
                </TR>
        </x:forEach>
      </TBODY>
   </TABLE>
   </body>
</html>
```


Solution exercice 2

Fichier etudiants-xsl.xsl

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="html"/>
   <xsl:template match="/">
      <html>
         <head>
            <meta http-equiv="Content-Type" content="text/html" charset="ISO-8859-1" />
            <title>etudiants-xsl.xsl</title>
         </head>
         <body>
            <xsl:apply-templates />
         </body>
      </html>
   </xsl:template>
```

Solution exercice 2 (suite)

```
<xsl:template match="etudiants">
  <table border="1" width="200">
    <caption>  Liste des etudiants : </caption>
    <tbody>
     <tr><th> ID </th> <th> Nom </th> <th> Prenom </th> </tr>
    <xsl:for-each select="etudiant">
      <tr>
        <td>  <xsl:value-of select="@id" />  </td>
        <td>   <xsl:value-of select="nom" /> </td>
        <td> <xsl:value-of select="prenom" /> </td>
      </tr>
    </xsl:for-each>
    </tbody>
  </table>
</xsl:template>
</xsl:stylesheet>
```


Solution exercice 2 (suite)

Fichier JSP

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html" charset="ISO-8859-1"/>
    <title>Fichier JSP</title>
  </head>
  <body>
    <c:import url="etudiants-xsl.xsl" var="fichierXSL"/>
    <c:import url="etudiants.xml" var="fichierXML"/>
    <x:transform doc="${fichierXML}" xslt="${fichierXSL}"/>
  </body>
</html>
```

Format Tag Library : introduction

- La bibliothèque Format Tag est utilisée pour formater et analyser les données pouvant être un simple message, une date, une heure ou un nombre.
- Elle Supporte l'internationalisation dans les pages JSP
- Syntaxe de la bibliothèque de balises de format:
  - `<%@ taglib uri = "http://java.sun.com/jsp/jstl/fmt" prefix = "fmt"%>`

Format Tag Library : internationalisation

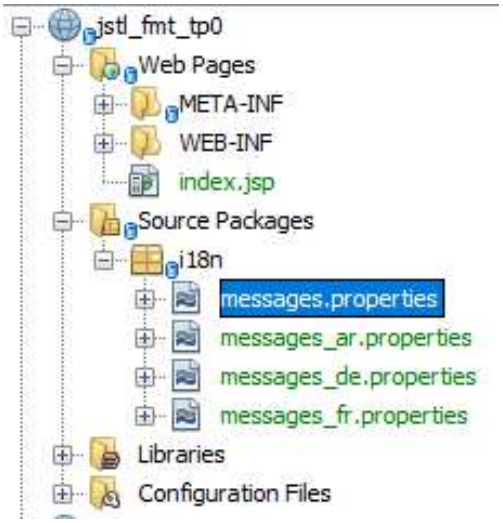
- L'internationalisation (notée i18n) consiste à disposer un site Web donnée en plusieurs langues.
- Chaque langue (appelée **locale**) dispose de sa propre façon de noter les nombres, les dates, etc.
- La localisation des messages, est réalisée à travers un ensemble de fichiers appelés **bundle** en anglais.
- Nous pouvons avoir un bundle pour les menus, un autre pour les messages, etc…
- Pour chaque bundle il faut définir un ensemble de fichiers où chaque fichier représente une langue particulière en plus d'un fichier pour la langue par défaut.

Format Tag Library : internationalisation

- Ces fichiers ont un préfixe commun appelé **basename** et doivent avoir comme extension ***.properties***
- Les fichiers pour les langues particulières utilisent le préfixe commun suivi du caractère "_" puis du code langue et éventuellement d'un "_" suivi du code pays.
- Exemple :
  - messages.properties // pour la langue par défaut
  - messages_en.properties // pour l'anglais
  - messages_ar.properties // pour l'arabe

Format Tag Library : internationalisation

- Remarque:
  - Les fichiers de l'internationalisation doivent être placés dans un sous-dossier du dossier src.
    - Par exemple : i18n
  - Après compilation ce dossier est copié dans le dossier web-inf/classes



Format Tag Library : internationalisation

- Dans chaque fichier, les clés sont identiques, seule la valeur associée change.
- Exemple :
  - le fichier messages.properties pour le français (langue par défaut)
  - msg=bonjour
- Exemple :
  - le fichier messages_en.properties pour l'anglais
  - msg=Hello
- Exemple :
  - le fichier messages_ar.properties pour l'arabe
  - msg=مرحبا

Format Tag Library : catégories de balises

- Catégories de balises :
  - **Définition de la langue** : setLocale
  - **Formatage de messages** : bundle, message, setBundle
  - **Formatage de dates et nombres** : formatNumber, parseNumber , formatDate, parseDate, setTimeZone, timeZone

Format Tag Library : balise <fmt:bundle>

- Permet de préciser un bundle à utiliser dans les traitements contenus dans son corps.
- Attributs :
  - **basename** : nom de base de la ressource à utiliser (obligatoire)
  - **prefix** : valeur préfixant chaque nom de clé pour les sous-attributs <fmt:message>
- Exemple :
  - <fmt:bundle basename="message" >
  - <fmt:message key="msg"/>
  - </fmt:bundle>

Format Tag Library : balise <fmt:setbundle>

- Permet de déclarer un bundle par défaut.
- Attributs :
  - **basename** : nom de base de la ressource à utiliser (obligatoire)
  - **var** : nom de la variable qui va stocker le nouveau bundle
  - **scope** : portée de la variable qui va recevoir le nouveau bundle
- Exemple :

```
monMessage =
<fmt:setBundle basename="message" />
 <fmt:message key="msg"/>
```

Format Tag Library : balise <fmt:message>

- Permet de localiser un message.
- Attributs :
  - key : clé du message à utiliser
  - bundle : bundle à utiliser
  - var : nom de la variable qui va recevoir le résultat du formatage
  - scope : portée de la variable qui va recevoir le résultat du formatage
- Exemple :

```
mon message =
<fmt:setBundle basename="message" />
  <fmt:message key="msg"/>
```

- Résultat : mon message = bonjour

Format Tag Library : balise <fmt:message>

- Remarque :
  - Si aucune valeur n'est trouvée pour la clé fournie alors le tag renvoie ???XXX ??? où XXX représente le nom de la clé.
- Exemple :
  - mon message =
  - <fmt:setBundle basename="message" />
  - <fmt:message key="test"/>
- Résultat :
  - mon message = ???test???

Format Tag Library : balise <fmt:setLocale>

- Permet de sélectionner une nouvelle Locale.
- Exemple :
  - <fmt:setLocale value="en"/>
  - <fmt:setBundle basename="message" />
  - <fmt:message key="msg"/>
- Résultat :
  - mon message = Hello

Format Tag Library : balise <fmt:formatNumber>

- Permet de formater des nombres selon la locale.
- Attributs :
  - value : valeur à formater
  - type : CURRENCY ou NUMBER ou PERCENT
  - pattern : format personnalisé
  - currencyCode : code de la monnaie à utiliser pour le type CURRENCY
  - currencySymbol : symbole de la monnaie à utiliser pour le type CURRENCY
  - groupingUsed : booléen pour préciser si les nombres doivent être groupés
  - maxIntegerDigits : nombre maximum de chiffres dans la partie entière
  - minIntegerDigits : nombre minimum de chiffres dans la partie entière

Format Tag Library : balise <fmt:formatNumber>

- maxFractionDigits : nombre maximum de chiffres dans la partie décimale
- minFractionDigits : nombre minimum de chiffres dans la partie décimale
- var : nom de la variable qui va stocker le résultat
- scope : portée de la variable qui va stocker le résultat

- Exemple :
  - <c:set var="montant" value="12345.67" />
  - montant = <fmt:formatNumber value="${montant}" type="currency"/>

Format Tag Library : balise <fmt:formatNumber>

- Exemple :

```
<h3>Formatage du nombre :</h3>
<c:set var="Amount" value="9850.14115" />
<p> Nombre formaté 1:
<fmt:formatNumber value="${Amount}" type="currency" /></p>
<p> Nombre formaté 2:
<fmt:formatNumber type="number" groupingUsed="true" value="${Amount}" /></p>
<p> Nombre formaté 3:
<fmt:formatNumber type="number" maxIntegerDigits="3" value="${Amount}" /></p>
<p> Nombre formaté 4:
<fmt:formatNumber type="number" maxFractionDigits="6" value="${Amount}" /></p>
<p> Nombre formaté 5:
<fmt:formatNumber type="percent" maxIntegerDigits="4" value="${Amount}" /></p>
<p> Nombre formaté 6:
<fmt:formatNumber type="number" pattern="####.###$" value="${Amount}" /></p>
```

Format Tag Library : balise <fmt:formatNumber>

- **Résultat** (nombre 9850.14115) :

  Formatage du nombre :

  Nombre formaté 1: $9,850.14

  Nombre formaté 2: 9,850.141

  Nombre formaté 3: 850.141

  Nombre formaté 4: 9,850.14115

  Nombre formaté 5: 5,014%

  Nombre formaté 6: 9850.141$

Format Tag Library : balise <fmt:parseNumber>

- Permet de convertir une chaîne de caractères qui contient un nombre en une variable décimale.
- Attributs :
  - value : valeur à traiter
  - type : CURRENCY ou NUMBER ou PERCENT
  - parseLocale : Locale à utiliser lors du traitement
  - integerOnly : booléen qui indique si le résultat doit être un entier (true) ou un flottant (false)
  - pattern : format personnalisé
  - var : nom de la variable qui va stocker le résultat
  - scope : portée de la variable qui va stocker le résultat

Format Tag Library : balise <fmt:parseNumber>

- Exemple :
  - Convertir en entier un identifiant passé en paramètre de la requête
  - <fmt:parseNumber value="${param.id}" var="id"/>

Format Tag Library : balise <fmt:formatDate>

- Permet de formater des dates selon la Locale.
- Attributs :
  - value : valeur à formater
  - type : DATE ou TIME ou BOTH
  - dateStyle : FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
  - timeStyle : FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
  - pattern : format personnalisé
  - timeZone : timeZone utilisée pour le formatage
  - var : nom de la variable qui va stocker le résultat
  - scope : portée de la variable qui va stocker le résultat

Format Tag Library : balise <fmt:formatDate>

- Exemple :
  - <jsp:useBean id="now" class="java.util.Date" />
  - Nous sommes le <fmt:formatDate value="${now}" type="date" dateStyle="full"/>.

Format Tag Library : balise <fmt:parseDate>

- Permet d'analyser une chaîne de caractères contenant une date pour créer un objet de type java.util.Date
- Attributs :
  - value : valeur à traiter
  - type : DATE ou TIME ou BOTH
  - dateStyle : FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
  - timeStyle : FULL ou LONG ou MEDIUM ou SHORT ou DEFAULT
  - pattern : format personnalisé
  - parseLocale : Locale utilisée pour le formatage
  - timeZone : timeZone utilisée pour le formatage
  - var : nom de la variable de type java.util.date qui va stocker le résultat
  - scope : portée de la variable qui va stocker le résultat

Format Tag Library : balise <fmt:setTimeZone>

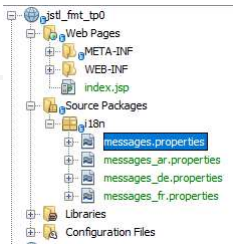
- permet de stocker un fuseau horaire dans une variable.
- Attributs :
  - value : fuseau horaire à stocker (obligatoire)
  - var : nom de la variable de stockage
  - scope : portée de la variable de stockage

Format Tag Library : balise <fmt:timeZone>

- Permet de préciser un fuseau horaire particulier à utiliser dans son corps.
- Attributs :
  - value : chaîne de caractères ou objet java.util.TimeZone qui précise le fuseau horaire à utiliser

Format Tag Library : Exercices

1) Ecrire une page JSP qui affiche un message de bienvenu par défaut en anglais
2) Modifier le fichier précédent pour imposer l'affichage du message en français
3) Modifier le fichier précèdent pour modifier le message dynamiquement en plusieurs langues (en, ar, fr, de) comme sur la figure
4) Ecrire une page JSP internationalisant un formulaire contenant un "user name" et un password.



Solution exercice 1

```
<%@ page contentType="text/html;charset=ISO-8859-1" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page isELIgnored="false" %>
<fmt:setBundle basename="i18n.messages"/>
<html>
<head>
   <title>i18n</title>
</head>
<body>
   <h2>
      <fmt:message key="label.welcome" />
   </h2>
</body>
</html>
```

- Contenu du fichier messages.properties : label.welcome = Welcome
- Contenu du fichier messages_fr.properties : label.welcome = Bienvenu
- …

Solution exercice 2

```
<%@ page contentType="text/html;charset=ISO-8859-1" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<%@ page isELIgnored="false" %>
<fmt:setLocale value="fr"/>
<fmt:setBundle basename="i18n.messages" />
<html>
<head>
   <title>i18n</title>
</head>
<body>
   <h2>
      <fmt:message key="label.welcome" />
   </h2>
</body>
</html>
```


Solution exercice 3

```
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt"%>
<%@ page isELIgnored="false"%>
<fmt:setLocale value="${param.lang}" />
<fmt:setBundle basename="i18n.messages" />
<html lang="${param.lang}">
   <head> <title>i18n</title> </head>
   <body>  <h2> <fmt:message key="label.welcome" /> </h2>
            <ul>
                  <li><a href="?lang=en">English</a></li>
                  <li><a href="?lang=ar">عربي </a></li>
                  <li><a href="?lang=fr">Français</a></li>
                  <li><a href="?lang=de">Deutsch</a></li>
            </ul>
   </body>
</html>
```

Solution exercice 4

```
…
 <body>
  <fmt:bundle basename="i18n.labels_fr">
    <form> <table>
        <tr> <td><fmt:message key="uname"></fmt:message></td>
                <td><input type="text"></td>
              </tr>
        <tr> <td><fmt:message key="pwd"></fmt:message></td>
             <td><input type="password"></td>
              </tr>
              <tr> <td> <input type="submit" value="Login"/> </td> </tr>
             </table>
          </form>
     </fmt:bundle>
   </body>
</html>
```


SQL JSTL

- Fournit un support SQL pour accéder à des bases de données relationnelles telles que Oracle, MySql, etc...
- Syntaxe de la bibliothèque de balises SQL:
  - `<%@ taglib uri = "http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>`
- Catégories de balises :
  - Définition de la source de données : setDataSource
  - Exécution de requêtes SQL : query, transaction, update

SQL JSTL : balise <sql:setDataSource>

- Permet de créer une connexion vers la base de données
- Attributs :
  - **driver** : nom de la classe du pilote JDBC à utiliser
  - **source** : URL de la base de données à utiliser
  - **user** : nom de l'utilisateur à utiliser lors de la connexion
  - **password** : mot de passe de l'utilisateur à utiliser lors de la connexion
  - **var** : nom de la variable qui va stocker l'objet créé lors de la connexion
  - **scope** : portée de la variable qui va stocker l'objet créé
  - **dataSource** : nom JNDI de la datasource

SQL JSTL : balise <sql:setDataSource>

- Exemples :
  - Cas de base de données Derby :

```
<sql:setDataSource driver="org.apache.derby.jdbc.ClientDriver"
url="jdbc:derby://localhost:1527/EtudiantsDB3" user="root" password="toto"/>
```

  - Cas de base de données MySQL :

```
<sql:setDataSource driver="com.mysql.cj.jdbc.Driver" url="jdbc:mysql://localhost/mydb"
user="root" password="toto"/>
```

SQL JSTL : balise <sql:query>

- Permet d'éxecuter des requêtes de sélection sur une source de données.
- Attributs :
  - sql : requête SQL à exécuter
  - var : nom de la variable qui stocke les résultats de l'exécution de la requête
  - scope : portée de la variable qui stocke les résultats
  - startRow : numéro de l'occurrence de départ à traiter
  - maxRow : nombre maximum d'occurrences à stocker
  - dataSource : connexion particulière à la base de données à utiliser

SQL JSTL : balise <sql:query>

- Exemple :
  - <sql:query var="reqEtudiants" sql="SELECT * FROM etudiants" />
- Remarque :
  - La requête SQL peut être précisée avec l'attribut sql ou dans le corps du tag :
- Exemple :
  - <sql:query var="reqEtudiants" >
  - SELECT * FROM etudiants
  - </sql:query>

SQL JSTL : balise <sql:query>

- Le résultat de la requête est stocké dans la variable spécifiée par l'attribut var.
- Cette variable est un objet de type jakarta.servlet.jsp.jstl.sql.Result qui dispose des méthodes suivantes :
  - String[] getColumnNames() : renvoie les noms des colonnes
  - int getRowCount() : renvoie le nombre d'enregistrements trouvés
  - Map[] getRows() : renvoie un tableau de Map où chaque élément représente une rangée.
  - Object[][] getRowsByIndex() : renvoie un tableau contenant les colonnes et leurs valeurs
  - boolean isLimitedByMaxRows() : renvoie un booléen qui indique si le résultat de la requête a été limité
- Exemple :
  - Nombre d'enregistrements trouvés : <c:out value="${reqEtudiants.rowCount}" />

SQL JSTL : balise <sql:query>

- Exemple 2: parcours des résultat avec foreach

```
<TABLE border="1" cellpadding="4" cellspacing="0">
 <TR> <td>id</td>  <td>nom</td>  <td>prenom</td> </TR>
 <c:forEach var="row" items="${reqEtudiants.rows}" >
 <TR> <td><c:out value="${row.id}" /></td>
  <td><c:out value="${row.nom}" /></td>
  <td><c:out value="${row.prenom}" /></td>
 </TR>
 </c:forEach>
</TABLE>
```

SQL JSTL : balise <sql:query> : Requête avec paramètres

- La requête SQL peut avoir des paramètres grâce au caractère "?".
- La valeur de ce paramètre est définie grâce aux balises <sql:param> ou <sql:dateParam> s'il s'agit d'une date.
- Attribut de la balise <sql:param> :
  - value : valeur de l'occurrence correspondante dans la requête SQL
- Attribut de la balise <sql:dateParam> :
  - value : objet de type *java.util.Date* qui contient la valeur de la date (obligatoire)
  - type : format de la date : TIMESTAMP ou DATE ou TIME

SQL JSTL : balise <sql:query> : Requête avec paramètre

- Exemple :
  - <c:set var="id" value="2" />
  - <sql:query var="reqEtudiants" >
  - SELECT * FROM etudiants where id = ?
  - <sql:param value="${id}" />
  - </sql:query>

SQL JSTL : balise <sql:transaction>

- Permet d'encapsuler plusieurs requêtes SQL dans une transaction.
- Attributs :
  - dataSource : connexion particulière à la base de données à utiliser
  - isolation : READCOMMITTED ou READUNCOMMITTED ou REPEATABLEREAD ou SERIALIZABLE
- Remarque :
  - l'**isolation** est la capacité d'un système d'isoler les modifications dans une transaction en cours de celles faites dans les autres transactions conduites simultanément

SQL JSTL : balise <sql:update>

- permet de réaliser une mise à jour grâce à une requête SQL sur la source de données.
- attributs :
  - sql : requête SQL à exécuter
  - var : nom de la variable qui stocke le nombre d'occurrences impactées par l'exécution de la requête
  - scope : portée de la variable qui stocke le nombre d'occurrences impactées
  - dataSource : connexion particulière à la base de données à utiliser

SQL JSTL : balise <sql:update>

- Exemple :
  - <c:set var="id" value="2" />
  - <c:set var="nouveauNom" value="nom 2 modifié" />
  - <sql:update var="nbRec">
  - UPDATE etudiants SET nom = ? WHERE id=?
  - <sql:param value="${nouveauNom}"/>
  - <sql:param value="${id}"/>
  - </sql:update>
  - nb enregistrements modifiés = <c:out value="${nbRec}"/>

SQL JSTL : exercice

- Reprendre le dernier exercice du chapitre JDBC et transformer le code JSP réservé pour l'affichage en utilisant SQL JSTL.

SQL JSTL : Solution

```
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>JSP Page</title>
   </head>
   <body>
      <sql:setDataSource var="dbSource"
                         driver="org.apache.derby.jdbc.ClientDriver"
                         url="jdbc:derby://localhost:1527/EtudiantsDB3"
                         user="root" password="root"/>
```


SQL JSTL : Solution

```
<sql:query dataSource="${dbSource}" var="resultats">
    SELECT * FROM ETUDIANT
 </sql:query>
 <h2> Liste des Etudiants Inscrits : </h2>
 <table border="1">
    <thead>
       <tr>
          <th>CNE</th>
          <th>NOM</th>
          <th>PRENOM</th>
       </tr>
    </thead>
```

SQL JSTL : Solution

```
<tbody>
    <c:forEach var="row" items="${resultats.rows}">
      <tr>
        <td><c:out value="${row.CNE}"/></td>
        <td><c:out value="${row.NOM}"/></td>
        <td><c:out value="${row.PRENOM}"/></td>
      </tr>
    </c:forEach>
  </tbody>
</table>

<a href="index.html"> retour à la page d'accueil </a>
</body>
</html>
```


Functions Tag Library

- Contient un ensemble de fonctions de manipulation de chaînes de caractères pouvant être utilisées avec le langage d'expression JSP.
- Syntaxe de la bibliothèque de balises de fonction:
  - `<%@ taglib uri = "http://java.sun.com/jsp/jstl/functions" prefix = "fn"%>`
- Liste des fonctions :
  - fn:contains() : Vérifie si une chaîne contient une sous-chaîne spécifiée.
  - fn:containsIgnoreCase() : Vérifie si une chaîne contient une sous-chaîne spécifiée sans tenir compte de la casse
  - fn:endsWith() : Vérifie si une chaîne se termine par le suffixe spécifié.

Functions Tag Library

- fn:escapeXml() : Remplace les caractères XML spéciaux par leurs codes
- fn:indexOf() : Renvoie l'index de la première occurrence de la sous-chaîne spécifiée.
- fn:join() : Joint tous les éléments d'un tableau avec la chaîne spécifiée.
- fn:length() : Renvoie le nombre d'éléments d'un tableau, d'une collection ou d'une chaîne de caractères.
- fn:replace() : Remplace une chaîne par une autre chaîne.
- fn:split() : Retourne un tableau de chaînes délimitées par le caractère en paramètre.
- fn:startsWith() : Retourne True si la chaîne commence par le préfixe spécifié.
- fn:endsWith() : Retourne True si la chaîne se termine par le suffixe spécifié.

Functions Tag Library

- fn:substring() : Retourne une sous-chaîne d'une chaîne de caractères.
- fn:substringAfter() : Retourne une sous-chaîne d'une chaîne de caractères après la sous-chaîne.
- fn:toLowerCase() : Renvoie tous les caractères de la chaîne en minuscule.
- fn:toUpperCase() : Renvoie tous les caractères de la chaîne en majuscule.
- fn:trim() : Supprime les espaces blancs au début et à la fin de la chaîne.

Functions Tag Library : Exemples

- Exemple 1 (fn:contains()):
  - <c:set var = "chaine" value = "bonjour tout le monde"/>
  - <c:if test = "${fn:contains(chaine, 'monde')}">
  - <p>monde trouvée<p>
  - </c:if>

Functions Tag Library : Exemples

- Exemple 2 (containsIgnoreCase ()):
  - <c:set var = "chaine" value = "bonjour tout le monde"/>
  - <c:if test = "${fn:containsIgnoreCase (chaine, 'MONDE')}">
  - <p>monde trouvée<p>
  - </c:if>

Functions Tag Library : Exemples

- Exemple 3 (fonction endsWith) :
  - <c:set var = "chaine" value = "bonjour tout le monde"/>
  - <c:if test = "${fn:endsWith (chaine, 'de')}">
  - <p>monde trouvée<p>
  - </c:if>

Functions Tag Library : Exercices

1) Ecrire une page JSP qui prend en paramètre une chaine de caractères et vérifie si c'est un palindrome (se lit des deux côtés) en supprimant d'éventuels espaces et en ignorant la casse.
2) Ecrire une page JSP qui cherche si un mot existe dans un fichier texte (qu'il faudra importer) et si oui affiche le nombre de ses occurrences.

Functions Tag Library : Solution Exercice 1

Index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>palindrome</title>
   </head>
   <body>
      <form method="get" action="palindrome.jsp">
         <label>Mot </label><input type="text" name="mot"/>
         <input type="submit" value="vérifier"/>
      </form>
   </body>
</html>
```


Functions Tag Library : Solution Exercice 1 (2)

palindrome.jsp

```
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/functions" prefix = "fn"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title>palidrome</title>
   </head>
   <body>
      <c:set var="ch" value="${param.mot}"/>
      <c:set var="lg" value="${fn:length(ch)}"/>
```

Functions Tag Library : Solution Exercice 1 (3)

```
<%-- calcul du milieu --%>

    <c:if test="${lg%2==0}">
        <c:set var="milieu" value="${lg/2}"/>
    </c:if>
    <c:if test="${lg%2!=0}">
        <c:set var="milieu" value="${(lg-1)/2}"/>
    </c:if>
    <%-- verification --%>
    <c:set var="estPalindrome" value ="${true}" />
    <c:forEach var="i" begin="${0}" end="${milieu}">
       <c:if test="${fn:substring(ch,i,i+1)!=fn:substring(ch,lg-1-i,lg-i)}">
          <c:set var="estPalindrome" value ="${false}" />
       </c:if>
    </c:forEach>
```


Functions Tag Library : Solution Exercice 1 (4)

```
<c:if test="${estPalindrome==true}">
        <c:out value="${ch}"/> est palindrome
    </c:if>
    <c:if test="${estPalindrome==false}">
        <c:out value="${ch}"/> n'est pas palindrome
    </c:if>
  </body>
</html>
```

Functions Tag Library : Solution Exercice 2

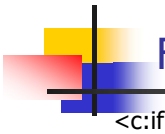
index.html

```
<!DOCTYPE html>
<html>
   <head>
      <title>Nombre d'occurences : index.html</title>
  </head>
   <body>
       <form method="get" action="nbrOccurences.jsp">
          <label>Mot : </label><input type="text" name="mot"/>
          <LABEL>Fichier : </LABEL>
          <INPUT type="file" name="fichier" accept=".txt"/>
          <input type="submit" value="chercher"/>
       </form>
   </body>
</html>
```


Functions Tag Library : Solution Exercice 2 (2)

nbrOccurences.jsp

```
<%@page contentType="text/html" pageEncoding="ISO-8859-1"%>
<%@ taglib uri = "http://java.sun.com/jsp/jstl/functions" prefix = "fn"%>
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<!DOCTYPE html>
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
      <title> nbrOccurences.jsp </title>
   </head>
   <body>
      <c:set var="motAchercher" value="${param.mot}"/>
      <c:set var="nomfichier" value="${param.fichier}"/>
      <c:import url="${nomfichier}" var="contenu" />
```



Functions Tag Library : Solution Exercice 2 (3)

```
<c:if test="${fn:contains(contenu,motAchercher)==true}" var="found"/>
      <c:if test="${found==true}">
        <c:set var="nbrOcc" value="${0}"/>
        <c:forTokens items = "${contenu}" delims = " ,." var = "mot">
          <c:if test="${motAchercher==mot}">
             <c:set var="nbrOcc" value="${nbrOcc+1}"/>
          </c:if>
        </c:forTokens>
      Nombre d'occurences du mot <c:out value="${motAchercher}"/> = <c:out value="${nbrOcc}"/>
      </c:if>
      <c:if test="${found==false}">
        <c:out value="${motAchercher}"/> n'existe pas dans le fichier
      </c:if>
   </body>
</html>
```