1. Introduction

Commencez par créer un dossier `tp4` pour ce TP et travaillez dedans.

1.1. Transformation d'un document XML par une feuille XSL

Téléchargez le document `albums.xml`, sa DTD `albums.dtd` et sa feuille `albums.xsl`.

Le but du TP est d'écrire d'autres feuilles XSLT pour transformer ce document de différentes manières. Il suffira de changer le nom de la feuille dans `albums.xml` dans la ligne suivante :

```
<?xml-stylesheet type="text/xsl" href="FICHIER.xsl"?>
```

Ensuite, pour exécuter la feuille et voir le résultat, vous avez plusieurs possibilités :

- Avec cette page : `xslt.html`. Il vous suffit de choisir le document XML à traiter tout en haut, puis la feuille de style. Vous pouvez aussi faire glisser un fichier XSL dans la grande zone en pointillés. En bas, cochez HTML si la feuille produit du HTML, puis cliquez sur `Transformer`. N'oubliez pas de recopier le source XSL vers un fichier quand ça marche. NB: cette page est encore expérimentale, donc il est recommandé de recopier très régulièrement la feuille de style en cours de mise au point dans un vrai fichier.
- Dans certains navigateurs : selon sa version et configuration, il vous suffit d'ouvrir le document XML avec Firefox. Dans certains cas, il faut modifier la configuration de Firefox pour autoriser la lecture de fichiers locaux (rejet des requêtes CORS : « La requête CORS n'est pas HTTP »). Donc :
  - Ouvrez un nouvel onglet dans Firefox, tapez `about:config` dans la barre d'adresse. Acceptez les risques,
  - Cherchez la clé `privacy.file_unique_origin` et mettez-la à `false` – il suffit de double-cliquer sur la valeur. Vous penserez à remettre `true` à la fin du TP.

  Ensuite, si c'est du HTML, il sera affiché comme prévu. Si c'est du XML, vous ne verrez que les textes.
- Dans XML Copy Editor : ouvrez le document XML puis utilisez le menu `XML`, `transformation XSL...` (raccourci `F8`) pour appliquer la feuille sur le document. Le résultat est affiché dans un nouvel onglet. Par contre, il semble qu'il y ait un bug majeur quand vous créez une feuille erronée puis que vous la corrigez, XML Copy Editor ne parvient pas à se remettre sur pied et continue à afficher le même message d'erreur. Il faut le fermer et le relancer.
- En ligne de commande : tapez seulement `xsltproc DOCUMENT.xml`. La commande va automatiquement chercher la feuille associée. Une autre possibilité consiste à lancer `xsltproc FEUILLE.xsl DOCUMENT.xml` pour choisir la feuille à appliquer sur le document.

1.2. Tutoriel

Le document XSLT minimal est le suivant (à enregistrer sous le nom `albums.xsl`) :

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/1999/XSL/Transform https://www.w3.org/2007/sche
```

```
  <!-- type du document de sortie : xml, html ou text -->
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <!-- METTEZ VOS TEMPLATES ICI -->

</xsl:stylesheet>
```

Référencez-le dans `albums.xml` ou utilisez `xslt.html`.

Un *template* (patron) est une règle de transformation de certains élément du document XML. Essayez par exemple, ces templates. À chaque fois, mettez le template précédent en commentaire.

- Exemple 1

```
<xsl:template match="/">
  <collection><xsl:copy-of select="//titre"/></collection>
</xsl:template>
```

Ce patron s'applique à la racine du document, donc à l'arbre tout entier. Il le remplace par un élément `<collection>` contenant tous les éléments `<titre>` du document.

- Exemple 2

```
<xsl:template match="//album">
  <BD>
    <xsl:attribute name="nom"><xsl:value-of select="titre"/></xsl:attribute>
    <xsl:copy-of select="date/annee"/>
  </BD>
</xsl:template>
```

Contrairement au précédent schéma, celui-ci ne génère pas un document XML bien formé en sortie car il n'y a pas de racine unique. Pour chaque élément `<album>` du document, le schéma génère un élément `<BD>` en lui rajoutant un attribut nom affecté avec le titre de l'album et un sous-élément `<annee>` provenant de la date de l'album.

Maintenant, au lieu de `<xsl:copy-of select="date/annee"/>`, mettez `<xsl:value-of select="date/annee"/>`. Cela met l'année directement au lieu de l'élément `<annee>`. En effet, `<xsl:copy-of>` recopie l'élément entier, avec son contenu, alors que `<xsl:value-of>` ne recopie que les textes qui sont dedans. De plus, il faut savoir que `<xsl:value-of>` ne recopie le contenu que du premier élément sélectionné (au lieu des contenus de tous les éléments).

- Exemple 3

```
<xsl:template match="/">
  <liste>
    <xsl:attribute name="nombre">
      <xsl:value-of select="count(albums/album)"/>
    </xsl:attribute>
    <xsl:for-each select="albums/album">
```

```
      <nom><xsl:value-of select="titre"/></nom>
    </xsl:for-each>
  </liste>
</xsl:template>
```

Ce patron traite la racine du document et il crée du XML bien formé avec une racine appelée `<liste>`. Il ajoute un attribut `nombre` à cette racine puis il fait une boucle permettant de générer un élément `<nom>` pour chaque album, dans l'ordre décroissant des années.

Pour classer les albums par année décroissante :

```
    <xsl:for-each select="albums/album">
      <xsl:sort select="date/annee" data-type="number" order="descending"/>
      <nom><xsl:value-of select="titre"/></nom>
    </xsl:for-each>
```

- Exemple 4

```
<xsl:template match="/">
  <collection serie="Tintin">
    <xsl:apply-templates
      select="/albums/album[@serie='Tintin' and @numero&gt;=10 and @numero&lt;=20]"/>
  </collection>
</xsl:template>

<xsl:template match="album">
  <nom><xsl:value-of select="titre"/></nom>
</xsl:template>
```

Au lieu de faire une boucle, on peut déléguer la transformation à un autre patron, lui demander de s'appliquer sur certains éléments du document. Remarquez comment les chaînes sont entourées dans un attribut, uniquement avec des `'...'`, et aussi comment les opérateurs de comparaison sont écrits.

- Exemple 5

```
<xsl:output method="html"/>

<xsl:template match="/">
  <html>
    <body>
      <ol>
        <xsl:apply-templates select="/albums/album[@serie='Tintin']"/>
      </ol>
    </body>
  </html>
</xsl:template>

<xsl:template match="album">
  <li>
```

```
    <xsl:value-of select="titre"/>
    <xsl:if test="@numero=16">, <b>mon album préféré</b></xsl:if>
  </li>
</xsl:template>
```

Au lieu de générer du XML, on génère du HTML. Le résultat est une énumération numérotée affichable dans un navigateur : ouvrez le document XML avec firefox pour voir le résultat. Notez que la source du document (`CTRL-U`) reste le document XML d'origine.

- Exemple 6 

```
<xsl:output method="text"/>

<xsl:template match="/">Voici ma collection de Tintin :
  <xsl:apply-templates select="/albums/album[@serie='Tintin']"/>
</xsl:template>

<xsl:template match="album">
  <xsl:value-of select="titre"/>
  <xsl:text>, </xsl:text>
</xsl:template>
```

La balise `<xsl:text>` permet de spécifier du texte à produire tel quel en sortie : espaces, sauts de ligne... Cette balise peut aussi être employée dans les autres types de sorties : html et xml.

Vous remarquerez que le dernier album est suivi d'une virgule. Ce serait mieux qu'il y ait un point. Voici comment faire : 

```
<xsl:output method="text"/>

<xsl:template match="/">Voici ma collection de Tintin :
  <xsl:apply-templates select="/albums/album[@serie='Tintin']"/>.
</xsl:template>

<xsl:template match="album">
  <xsl:value-of select="titre"/>
  <xsl:if test="position()!=last()">
    <xsl:text>, </xsl:text>
  </xsl:if>
</xsl:template>
```

La balise `<xsl:if>` fait un traitement conditionnel tout simple : afficher ou non une virgule et un espace. Pour des cas plus complexes, on utiliserait une balise `<xsl:choose>`. Notez que le point final est produit par l'autre template.

- Exemple 7 

```
<xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

<xsl:key name="index_mois"
```

```
    match="//album" use="date/mois/text()"/>

<xsl:template match="/">
  <parutions mois="janvier">
    <xsl:for-each select="key('index_mois', 'janvier')">
      <xsl:copy-of select="titre"/>
    </xsl:for-each>
  </parutions>
</xsl:template>
```

La directive `<xsl:key name="`*nom_index*`" match="`*élément*`" use="`*valeur*`">` demande à XLST de mémoriser un index, c'est à dire des couples (valeur, liste des éléments liés à cette valeur). Ici, on va mémoriser les albums en fonction de leur mois de parution. Donc on va avoir, entre autres, un couple (`'janvier'`, *liste des albums parus en janvier*).

Le patron consiste à parcourir cette liste. Il utilise la fonction `key(`*nom_index*`,`*valeur*`)` pour aller chercher les éléments ayant cette valeur. Donc ça va afficher les albums parus en janvier.

On peut faire ça plus simplement sans utiliser d'index. Un index permet d'isoler les valeurs distinctes d'une information. Par exemple, ici, on avait un index avec les mois en tant que clés et des collections d'albums en tant que valeurs en face des clés. Le malheur vient du fait que XSL ne permet pas d'itérer sur les clés d'un index. On peut seulement itérer sur les éléments et demander si chacun est le premier de chaque case d'index. C'est cette astuce qui est utilisée dans l'exemple suivant.

- Exemple 8 

```
<xsl:key name="annees" match="//album" use="date/annee/text()" />

<xsl:template match="/">
  <calendrier>
    <xsl:for-each
      select="//album[generate-id()=generate-id(key('annees', date/annee/text())[1])]">
      <xsl:sort select="date/annee" data-type="number" order="descending"/>
      <annee>
        <xsl:value-of select="date/annee"/>
        <xsl:text> : </xsl:text>
        <xsl:for-each select="key('annees', date/annee/text())">
          <xsl:value-of select="titre"/>
          <xsl:text>, </xsl:text>
        </xsl:for-each>
      </annee>
    </xsl:for-each>
  </calendrier>
</xsl:template>
```

C'est le patron le plus complexe. Il comprend deux boucles imbriquées. La boucle extérieure est appelée boucle avec regroupement. Le principe est de construire une liste des valeurs distinctes de quelque chose, ici ce sont les années. On place cette liste dans un index comme dans l'exemple

précédent. Chaque libellé d'année est associé à la liste des albums de cette année. NB: dans le cas des albums de Tintin, il n'y a qu'un seul album par année.

Ensuite, il reste à faire une boucle sur les valeurs de la clé, et c'est ça qui est très bizarre et qui ne sera pas expliqué ici en détails. Il faut juste savoir qu'il y a un mécanisme sous-jacent qui numérote les éléments d'un document de manière unique. On peut accéder à l'identifiant de l'élément courant par la fonction `generate-id()`. Et l'appel `generate-id(chose)` retourne l'identifiant de la chose. L'idée de la boucle est de parcourir tous les éléments `<annee>` mais seulement ceux qui sont les premiers dans l'index des années. Ainsi la même année n'est parcourue qu'une seule fois.

Dans la boucle extérieure, il y a une boucle intérieure qui parcourt les albums de l'année courante. Elle utilise la fonction `key()` pour obtenir la liste des albums concernés. Mais avec ce fichier XML, il n'y a qu'un seul album par année, alors c'est d'intérêt limité.

En résumé, pour faire une boucle sur les valeurs distinctes d'un élément, il suffira d'appliquer le schéma suivant :

```
<xsl:key name="INDEX" match="ELEMENT" use="VALEUR" />

<xsl:template match="...">
  <xsl:for-each
        select="ELEMENT[generate-id()=generate-id(key('INDEX', VALEUR)[1])]">
    ...
  </xsl:for-each>
</xsl:template>
```

2. Exercices sur `uwm_courses.xml`

On va travailler sur une liste de cours de l'Université du Wisconsin-Milwaukee (UWM) qui ont été rendus gracieusement disponibles (libres de droits) et adaptés pour ce TP. Téléchargez le fichier `uwm_courses.xml` et sa DTD `uwm_courses.dtd`, puis rajoutez-lui des feuilles de transformation selon les demandes suivantes.

Prenez un peu de temps pour étudier la structure du fichier XML et savoir quelles sont les informations disponibles. C'est très répétitif, donc il suffit de regarder le début.

Remarque importante : les exercices sont difficiles. Si vous n'arrivez pas exactement au résultat demandé, essayez de vous en rapprocher du mieux possible. Pour chaque exercice, il y a plusieurs niveaux de réalisation. Le plus simple est de produire la racine du document demandé. Ensuite, essayer de produire les éléments du premier niveau, puis leurs attributs, leurs contenus, etc.

2.1. Modules rapportant au moins 6 crédits : `uwm_courses_E1.xsl`

Produire un document HTML listant tous les modules de cours rapportant au moins 6 crédits sous forme d'une énumération, indiquant aussi le nombre de sous-modules. Voici le résultat attendu, il s'affichera proprement dans un navigateur :

```
<html>
  <body>
    <p>Il y a 26 module(s) rapportant au moins 6 crédits :</p>
    <ul style="list-style-type:square">
      <li>BASIC ELEMENTS OF FILMMAKING II : 2 sous-modules</li>
      <li>JUNIOR PROJECT : 2 sous-modules</li>
      <li>SENIOR PROJECT : 2 sous-modules</li>
      <li>NURSING PRACTICE I : 10 sous-modules</li>
      <li>WASHINGTON INTERNSHIP : 1 sous-modules</li>
      <li>OCCUPATIONAL THERAPY FIELD SERVICE I : 2 sous-modules</li>
      ...
    </ul>
  </body>
</html>
```

Vous aurez besoin de regarder l'imbrication entre `<submodule>`, `<module>`, `<credits>` et ses attributs. Le nombre de crédits à prendre en compte est le nombre `min` de l'élément `<credits>`. Attention à bien écrire une condition du style *nombre min de crédits >= 6* même si, dans ce fichier, le *min* ne dépasse pas 6.

2.2. Emploi du temps de la salle EMS/E495 : `uwm_courses_E2.xsl`

Produire un document HTML qui énumère les modules enseignés dans la salle « E495 » du bâtiment « EMS » et les jours de la semaine où ils sont donnés, ainsi que les enseignants, le tout dans un tableau. Voici le résultat attendu :

```
<html>
  <body>
    <h3>Room EMS/E495 courses</h3>
    <table>
      <thead>
        <tr>
          <th>Module (submodule)</th>
          <th>Days</th>
          <th>Instructor</th>
        </tr>
      </thead>
      <tbody>
        <tr>
          <td>CONTEMPORARY APPLICATIONS OF MATHEMATICS (Se 013)</td>
          <td>tuesday, thursday</td>
          <td>Arnold</td>
        </tr>
        <tr>
          <td>CALCULUS WITH PRECALCULUS I (Se 004)</td>
          <td>monday, wednesday, friday</td>
          <td>Brazauskas</td>
        </tr>
```

```
        ...
      </tbody>
    </table>
  </body>
</html>
```

Vous aurez besoin de regarder l'imbrication des éléments `<location>`, `<day>` et `<instructor>` ainsi que `<module>` et `<submodule>`.

C'est un peu compliqué de ne pas avoir une virgule à la fin de la liste des jours. Il faut tester si l'élément `<day>` n'est pas le dernier.

Voici quelques styles pour améliorer l'apparence du tableau : 

```
  <head>
    <style>
    table {
      box-shadow: 3px 4px 5px rgba(0, 0, 0, 0.2);
    }
    table, th, td {
      border: 1px solid gray;
      border-collapse: collapse;
    }
    th, td {
      padding: 6px;
    }
    </style>
  </head>
```

2.3. Modules ayant 8 sous-modules : `uwm_courses_E3.xsl`

Produire un document HTML qui énumère les modules ayant exactement 8 sous-modules, et pour chacun des sous-modules afficher son nom et l'enseignant. Voici le résultat attendu :

```
<html>
  <body>
    <ul>
      <li>
        FIRST-SEMESTER FRENCH
        <ol>
          <li>Se 001: Alkhas</li>
          <li>Se 002: Swenson</li>
          <li>Se 003: Lemke</li>
          ...
        </ol>
      </li>
      <li>
        OUR PHYSICAL ENVIRONMENT
        <ol>
```

```
          <li>Lc 401: Fredlund</li>
          <li>Lc 402: Ottone</li>
          <li>La 801: Mujica</li>
          ...
        </ol>
      </li>
        ...
    </ul>
  </body>
</html>
```

Vous aurez besoin de regarder l'imbrication des éléments `<module>`, `<submodule>` et `<instructor>`.

2.4. Nombre de cours par enseignant : `uwm_courses_E4.xsl`

Produire un document HTML qui affiche le nombre de sous-modules de chaque enseignant pour ceux qui en ont plus de 30, de préférence classée par nombre décroissant comme dans le résultat attendu suivant :

```
<html>
<body>
<ol>
<li>TA : 453 cours</li>
<li>Acad Staff : 299 cours</li>
<li>Faculty : 206 cours</li>
<li>Baer : 85 cours</li>
<li>Peterson : 43 cours</li>
<li>Thompson : 40 cours</li>
<li>Harris : 34 cours</li>
<li>McGinn : 32 cours</li>
<li>Hanrahan : 31 cours</li>
</ol>
</body>
</html>
```

Indications :

- Construire un index `instructors` contenant les noms des enseignants.
- Parcourir cet index dans une boucle. Elle sert à construire les éléments `<li>` du résultat.
- Compter les sous-modules de chaque enseignant. Pour cela, vous avez deux solutions, soit parcourir tout le document pour chaque enseignant, soit utiliser l'index. La seconde est de très loin la plus rapide.
- Cette méthode de comptage est reprise pour classer les enseignants.

3. Travail à rendre

Dans Firefox, onglet `about:config`, remettez `privacy.file_unique_origin` à `true` si vous l'aviez changé.

Vous avez travaillé dans le dossier `tp4`. Remontez au dessus avec le navigateur de fichiers. Cliquez droit sur le dossier `tp4`, choisissez `Compresser...`, cliquez sur `Créer`. Ça va créer une archive `tp4.tar.gz`. Déposez cette archive sur Moodle, dans la page de cours dédiée L4IN121T Formats et traitements de données internet.