1. Introduction

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

Téléchargez le document `albums.xml` et sa DTD `albums.dtd`. Le but du TP est d'écrire plusieurs feuilles XSLT pour transformer ce document.

Dans chacun des exercices suivants, vous allez créer une feuille de style XSLT différente. Vous les associerez successivement au même document XML en changeant seulement le nom du fichier 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 :

- Dans le navigateur : il vous suffit d'ouvrir le document XML avec Iceweasel/Firefox. 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 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.

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

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <!-- type du document de sortie : xml, html ou text -->
  <xsl:output method="xml" version="1.0" encoding="UTF-8" indent="yes"/>

  <!-- VOUS METTEZ DES TEMPLATES ICI -->

</xsl:stylesheet>
```

Un *template* (patron) est une règle de transformation de certains élément du document XML. Essayez par exemple, ces templates. Faites un fichier différent pour chacun en repartant de `modele.xsl` à chaque fois.

- 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">
      <xsl:sort select="date/annee" order="descending"/>
      <xsl:variable name="album" select="current()"/>
      <nom><xsl:value-of select="$album/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. Dans la boucle, on peut utiliser cet élément avec `.` ou avec la fonction `current()` mais ici, j'ai préféré définir une variable contenant l'élément courant de la boucle ; je l'utilise par `$album`.

- 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 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. Il faut seulement remplacer le second template par celui-ci :

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

```
    <xsl:when test="position()=last()">
      <xsl:text>.</xsl:text>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>, </xsl:text>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>
```

La balise `<xsl:choose>` permet de faire une conditionnelle multiple, avec des éléments `<xsl:when>` et un dernier `<xsl:otherwise>`. Une simple conditionnelle est possible avec `<xsl:if>`, mais il n'y a pas de *else*.

- Exemple 7 

```
<xsl:key name="index_debut_titre"
    match="//album[@serie='Tintin']" use="substring(titre,1,6)"/>

<xsl:template match="/">
  <Tintins>
    <xsl:for-each select="key('index_debut_titre', 'Tintin')">
      <xsl:copy-of select="titre"/>
    </xsl:for-each>
  </Tintins>
</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 ayant cette valeur). Ici, on va mémoriser les albums en fonction des 6 premiers caractères de leur titre. Donc on va avoir, entre autres, un couple (`'Tintin'`, *liste des albums dont le titre commence par 'Tintin'*).

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 titres des albums qui commencent par `Tintin`. NB: on peut faire ça plus simplement.

- Exemple 8 

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

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

```
          <xsl:value-of select="@numero"/>
          <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 élement, il suffira de prendre le schéma suivant :

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

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

2. Exercices sur `spatial.xml`

Téléchargez le fichier `spatial.xml` et sa DTD `spatial.dtd`, puis rajoutez-lui des feuilles de transformation selon les demandes suivantes.

Remarque importante : certains de ces 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. Noms des programmes : `spatial_E1.xsl`

Produire un document HTML listant toutes les nations ayant un programme spatial sous forme d'une énumération. Voici le résultat attendu :

```
<html>
<body>
<p>Il y a 2 nations :</p>
<ul style="list-style-type:square">
<li>USA : 3 programmes de 1958 à 1975</li>
<li>URSS : 3 programmes de 1961 à 1981</li>
</ul>
</body>
</html>
```

Les informations nombre et dates des programmes sont extraits du document XML.

2.2. Missions de Gene Cernan : `spatial_E2.xsl`

Produire un document HTML qui énumère les missions de Gene Cernan : rôle, nom et date. Voici le résultat attendu :

```
<html>
<body>
<p>Gene Cernan a effectué 3 missions :</p>
<ol>
<li>équipage dans la mission Gemini 9, 1966-06-03</li>
<li>équipage dans la mission Apollo 10, 1969-05-18</li>
<li>commandant dans la mission Apollo 17, 1972-12-07</li>
</ol>
</body>
</html>
```

Tous ces textes sont générés automatiquement avec xsltproc, en particulier, la feuille XSLT écrit « équipage » quand il n'y a pas de rôle. Mais ça ne marche pas avec Firefox et XML Copy Editor, le rôle sera vide, alors si vous pouvez, rajoutez un test.

2.3. Sorties dans l'espace : `spatial_E3.xsl`

Produire un document XML listant toutes les missions où l'un des buts était « Sortie dans l'espace ». Il doit y avoir un élément `<mission>` pour chacune. Cet élément doit avoir trois attributs : le programme de la mission, le nom de la mission et `nb_astronautes` valant le nombre d'astronautes de la mission. Et en dessous, il doit y avoir la liste des astronautes comme dans le fichier XML d'origine. Pour finir, la racine `<missions>` doit avoir un attribut `nombre` valant le nombre total de missions concernées.

Voici un extrait de ce qu'il faut obtenir :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<missions nombre="4">
  <mission programme="Gemini" nom="Gemini 4" nb_astronautes="2">
    <astronaute role="commandant">James McDivitt</astronaute>
    <astronaute role="équipage">Edward White</astronaute>
  </mission>
  ...
  <mission programme="Gemini" nom="Gemini 12" nb_astronautes="2">
    <astronaute role="commandant">Jim Lovell</astronaute>
    <astronaute role="équipage">Edwin Aldrin</astronaute>
  </mission>
</missions>
```

Il y a une petite difficulté car la chaîne à chercher contient un `'`. Il faut écrire des choses comme ceci, en jouant sur les `'` et `"` et les entités. Toutes les combinaisons ne sont pas bonnes et celle-ci fonctionne :

```
select='//mission[but="Sortie dans l&apos;espace"]'
```

2.4. Événements : `spatial_E4.xsl`

Produire un document texte listant les événements survenus. Il y a une ligne par événement et le nom de la mission devant.

Voici un extrait de ce qu'il faut obtenir :

```
Mercury 4 : Submersion de la capsule à l'amerrissage
Mercury 6 : Rupture de la fixation du bouclier thermique, sans conséquence
...
```

2.5. Missions des astronautes : `spatial_E5.xsl`

Produire un document HTML affichant les missions des astronautes.

Rappel : une table HTML se place dans un élément `<table>`. Chaque ligne se place dans un élément `<tr>`. Chaque case se place dans un élément `<td>`, sauf les titres en première ligne, dans des éléments `<th>`. Pour obtenir un résultat un tout petit peu sucré, vous pouvez rajouter des styles CSS dans l'entête HTML, voir l'extrait ci-dessous.

Voici un court extrait de ce qu'il faut obtenir :

```html
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<style>
              table, th, td {
                border: 1px solid black;
                border-collapse: collapse;
                padding: 15px;
```

```
            }
        </style>
</head>
<body>
    <table>
        <tr>
            <th>astronaute</th>
            <th>missions</th>
        </tr>
        <tr>
            <td>Alan Shepard</td>
            <td>
                <ul>
                    <li>équipage dans Mercury 3</li>
                    <li>équipage dans Apollo 14</li>
                </ul>
            </td>
        </tr>
        <tr>
            <td>Virgil Grissom</td>
            <td>
                <ul>
                    <li>commandant dans Mercury 4</li>
                    <li>commandant dans Gemini 3</li>
                    <li>commandant dans Apollo 1</li>
                </ul>
            </td>
        </tr>
        ...
    </table></body>
</html>
```

NB: la ligne `<meta...` est générée automatiquement par `xsltproc`.

Indications :

- Relisez attentivement l'exemple 8.
- Construire un index mémoriser tous les astronautes en fonction de leur nom. Il est dans `text()`.
- Écrire une boucle externe pour parcourir les astronautes distincts en utilisant l'index.
- Stocker l'astronaute courant dans une variable. C'est pour pouvoir y faire référence dans la boucle interne suivante.
- Écrire une boucle interne pour parcourir les différentes missions de l'astronaute et générer une énumération HTML.
- Si vous ne pouvez pas tout faire, essayez au moins d'afficher une liste des astronautes sans doublons. Rajoutez-y leurs missions sans aucune mise en page.

3. Travail à rendre

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.