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.

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 tous les programmes spatiaux sous forme d'une énumération. Voici le résultat attendu :

```
<html>
<body>
<p>Il y a 6 programmes spatiaux :</p>
<ul style="list-style-type:square">
<li>Mercury (1958 à 1963, USA) : Découvrir le vol spatial autour de la Terre</li>
<li>Gemini (1963 à 1966, USA) : Maîtriser le vol spatial en orbite
autour de la Terre et mettre au point les manœuvres de rendez-vous</li>
<li>Apollo (1961 à 1975, USA) : Poser des astronautes sur la Lune et l'explorer</li>
<li>Vostok (1961 à 1963, URSS) : Découvrir le vol spatial autour de la Terre</li>
<li>Voskhod (1964 à 1965, URSS) : Maîtriser le vol spatial en orbite
autour de la Terre et mettre au point les manœuvres de rendez-vous</li>
<li>Soyouz (1962 à 1981, URSS) : Poser des astronautes sur la Lune et l'explorer</li>
</ul>
</body>
</html>
```

Les informations nombre de programmes, noms des programmes, des nations et objectifs sont extraits du document XML.

2.2. Missions de Dave Scott : `spatial_E2.xsl`

Produire un document HTML qui énumère les missions de Dave Scott : nom, (date) et rôle. Voici le résultat attendu :

```
<html>
<body>
<p>Dave Scott a effectué 3 missions :</p>
<ol>
<li>Gemini 8 (1966-03-16) en tant que équipage</li>
<li>Apollo 9 (1969-03-03) en tant que pilote</li>
<li>Apollo 15 (1971-07-26) en tant que commandant</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.

2.3. Missions habitées : `spatial_E3.xsl`

Produire un document XML listant toutes les missions habitées quelque soit la nation. Il doit y avoir un élément `<mission>` pour chacune. Cet élément doit avoir trois attributs : `nation` valant

le nom de la nation concernée par cette mission, le nom de la mission et `nb_astronautes` valant le nombre d'astronautes de la mission. Pour finir, la racine `<missions>` doit avoir un attribut `nombre` valant le nombre total de missions habitées et un attribut `nb_astronautes` valant le nombre total d'astronautes de ces missions (peu importe si le même astronaute a fait plusieurs missions).

Voici un extrait de ce qu'il faut obtenir :

```
<?xml version="1.0" encoding="UTF-8"?>
<missions nombre="41" nb_astronautes="83">
  <mission nation="USA" nom="Mercury 3" nb_astronautes="1"/>
  <mission nation="USA" nom="Mercury 4" nb_astronautes="1"/>
  ...
  <mission nation="USA" nom="Apollo 17" nb_astronautes="3"/>
  <mission nation="URSS" nom="Vostok 1" nb_astronautes="1"/>
  <mission nation="URSS" nom="Vostok 2" nb_astronautes="1"/>
  ...
  <mission nation="URSS" nom="Soyouz 6" nb_astronautes="2"/>
</missions>
```

2.4. Missions par année : `spatial_E4.xsl`

Produire un document HTML affichant toutes les missions, regroupées par année.

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 extrait de ce qu'il faut obtenir : 

```
<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>année</th><th>missions</th></tr>
<tr><td>1961</td><td>Mercury 3 Mercury 4 Vostok 1 Vostok 2 </td></tr>
<tr><td>1962</td><td>Mercury 6 Mercury 7 Mercury 8 Vostok 3 Vostok 4 </td></tr>
...
<tr><td>1972</td><td>Apollo 16 Apollo 17 </td></tr>
</table></body>
</html>
```

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

Indications :

- Relisez attentivement l'exemple 8.
- Construire un index pour trouver les missions en fonction de l'année. L'année est dans les 4 premiers caractères de la date.
- Écrire une boucle externe pour parcourir les années distinctes en utilisant l'index.
- Écrire une boucle interne pour parcourir les différentes missions de l'année courante.

2.5. Buts : `spatial_E5.xsl`

Produire un document texte listant les buts des missions. Il y a une ligne par mission avec son but ensuite. Si une mission a plusieurs buts, cela fait plusieurs lignes affichées. Il ne faut pas afficher les missions qui n'ont pas de but (comme Soyouz 6).

Voici un extrait de ce qu'il faut obtenir :

```
Mercury 3 : Premier vol spatial américain
Mercury 4 : Test sub-orbital
Mercury 6 : Premier vol orbital
Mercury 7 : Expériences sur l'impesanteur
Mercury 9 : Vol de longue durée
Gemini 1 : Tester l'intégrité de la capsule
...
```

2.6. Astronautes : `spatial_E6.xsl`

Produire un document XML résumant les activités des astronautes. Il doit y avoir une racine `<astronautes>` contenant des éléments `<astronaute>` pour chaque astronaute distinct du document. Ces éléments `<astronaute>` ont un attribut `nom` valant le nom de l'astronaute. Ils ont des sous-éléments `<commandant>`, `<pilote>` et `<membre>` en fonction des rôles et missions qu'on eu les astronautes concernés. Ces éléments peuvent être vides le cas échéant.

Voici un extrait de ce qu'il faut obtenir :

```
<?xml version="1.0" encoding="UTF-8"?>
<astronautes>
  <astronaute nom="Alan Shepard">
    <commandant>
      <mission>Apollo 14</mission>
    </commandant>
    <pilote/>
    <membre>
      <mission>Mercury 3</mission>
    </membre>
  </astronaute>
  ...
  <astronaute nom="Valery Kubasov">
    <commandant/>
```

```
    <pilote/>
    <membre>
      <mission>Soyouz 6</mission>
    </membre>
  </astronaute>
</astronautes>
```

Indications :

- Construire un index contenant les noms des astronautes. Parcourir cet index dans une boucle externe. Elle sert à construire les éléments `<astronaute>` du résultat.
- Stocker l'astronaute courant dans une variable.
- Prévoir trois traitements similaires : un pour lister les missions dans lesquelles il est commandant, un pour celles où il est pilote, et une dernière pour les missions où il n'a pas de rôle.
- Écrire une boucle interne permettant de parcourir les missions en sélectionnant celles de cet astronaute.
- Ensuite, il faut arriver à écrire une expression XPath permettant d'isoler cet astronaute parmi ceux de cette mission. Utiliser la variable affectée dans la boucle externe.

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.