3. Lire la transformation *XMLversXHTML_articles+regroupements_sections_alphabetiques_LarDeb05-2021-12-17.xsl* et répondre aux questions de compréhension relatives à son code

En vous aidant des explications reçues pour les différentes transformations écrites ou simplement lues en cours et de toute documentation prtinente, pouvez-vous répondre aux questions portant sur cette nouvelle transformation ? Le texte de cette transformation *XMLversXHTML_articles+regroupements_sections_alphabetiques_LarDeb05-2021-12-17.xsl* comporte ci-après des zones surlignées en jaune associées à des notes de bas de page vous questionnant sur les instructions ou expressions Xpath surlignés. Merci de répondre (en français et de manière argumentée) dans les notes, sous chaque question, à raison de 10 lignes maximum par réponse.

Le fichier à rendre sous le nom *XSLT_avec_commentaires_nom(s).pdf* est constitué des pages de ce sujet contenant la transformation annotée et vos réponses (sous mes questions) dans les notes.

Nom(s) : DEME, MALAGNOUX, MODOLO

Fichier *XMLversXHTML_articles+regroupements_sections_alphabetiques_LarDeb05-2021-12-17.xsl*

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:xd="http://www.oxygenxml.com/ns/doc/xsl"
    exclude-result-prefixes="xs xd"
    version="2.0">[1]
    <xd:doc scope="stylesheet">
        <xd:desc>
            <xd:p><xd:b>Created on:</xd:b> Dec 17, 2021</xd:p>
            <xd:p><xd:b>Author:</xd:b> Gasiglia</xd:p>
            <xd:p>Trasformation XML vers XHTML de fichiers d'extraits du Larousse des débutants (2005), comme LarDeb05_37art.xml </xd:p>
        </xd:desc>
    </xd:doc>
```

[1] Pourquoi n'avons-nous pas besoin d'ajouter « xpath-default-namespace="http://www.tei-c.org/ns/1.0" » avant l'indication de la version de XSLT de cette transformation ? Nous en avons eu besoin en cours pour la transformation du XML de *Sarrasine* en XHTML.

Dans la transformation de Sarrasine, nous avons utilisé « xpath-default-namespace="http://www.tei-c.org/ns/1.0" » parce que dans les attributs de la transformation, nous avons indiqué que nous utilisions des noms d'éléments TEI qui sont les éléments déclarés dans la TEI. De plus, si nous utilisons des éléments avec les <namespace>, il faut impérativement que dans notre fichier, le système retrouve la référence à la TEI et la déclaration <namespace>. En revanche, dans cette transformation, nous ne voulons pas que le fichier se conforme à la TEI.

```
    <xsl:output method="xhtml" indent="no"/>[2]
    <xsl:strip-space elements="*"/>

<xsl:template match="/">
    <xsl:element name="html">[3]
        <xsl:element name="head">
            <xsl:element name="title"><xsl:text>Larousse des débutants (2005) [extrait]</
xsl:text></xsl:element>
            <xsl:element name="link">
                <xsl:attribute name="href">LarDeb2005_avec_index.css</xsl:attribute>
                <xsl:attribute name="rel">stylesheet</xsl:attribute>
                <xsl:attribute name="type">text/css</xsl:attribute>
            </xsl:element>
        </xsl:element>
        <xsl:element name="body">
            <!-- Création d'une liste "index" des items en adresse accompagnés de leur catégorie
-->
            <xsl:element name="div">
                <xsl:attribute name="class">index</xsl:attribute>
                <xsl:element name="ul">
                    <xsl:element name="h1"><xsl:text>Index des articles</xsl:text></
xsl:element>
                    <xsl:apply-templates select="descendant::composant-d-identification"
mode="index">[4]
                        <xsl:sort select="item-en-adresse" order="ascending" />[5]
                    </xsl:apply-templates>
                </xsl:element>
            </xsl:element>
```

---

[2] Qu'indique ceci ?

La balise <xsl:output> permet de formater le fichier-résultat, ici, sorti en html à partir d'un document xml comme l'indique <xsltoutput method=" xhtml " >.

[3] Pourrions-nous alternativement créer cet élément dans un autre <xsl:template> présent ou à ajouter ? Si oui, celui de quel élément ? Pourquoi ?

Oui, cela est possible, il peut être créé dans un autre <xsl:template> matchant <Larousse-des-debutants> qui est l'élément à la racine tandis qu'ici c'est la racine elle-même du document qui est matchée.

[4] Pouvons-nous coder cet axe d'une autre manière (non alphabétique) ? Quelle serait alors la valeur de l'@select ?

Oui, cela est possible, le descendant contenu dans la balise <xsl:apply-templates select=" descendant:: "> peut être remplacé par une double barre oblique. La valeur de l'@select pouvant être formulée de façon non alphabétique de la manière suivante : <xsl:apply-templates select=" //item-en-adresse " > .

[5] Quel est l'objet sélectionné ? La mention de cet objet par son nom nous dit que c'est le fils de quelque chose. De quoi est-il le fils ? Quand commençons-nous à manipuler le père ?

L'objet sélectionné est <item-en-adresse> trié par une balise <xsl:sort>. Il est le fils de <composant-d-identification> qui a été manipulé par le <xsl:apply-templates select="descendant::composant-d-identification" mode=« index"> qui le précède.

```
            <!-- Restitution du texte complet -->
            <xsl:element name="div">
                <xsl:attribute name="class">texte_complet</xsl:attribute>
                <xsl:element name="h1">Texte complet</xsl:element>
                <xsl:apply-templates select="descendant::article | descendant::article-de-
readressage" mode="regroupement"/>
            </xsl:element>
        </xsl:element>
    </xsl:element>
</xsl:template>

    <xsl:template match="composant-d-identification" mode="index">
        <xsl:variable name="ID_élément_courant" select="generate-id()"/>
        <xsl:element name="li">[6]
            <xsl:attribute name="class">
                <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:element name="a">
                <xsl:attribute name="id">
                    <xsl:text>index_</xsl:text>
                    <xsl:value-of select="$ID_élément_courant"/>
                </xsl:attribute>
                <xsl:attribute name="href">[7]
                    <xsl:text>#</xsl:text>[8]
                    <xsl:value-of select="$ID_élément_courant"/>
                </xsl:attribute>
                <xsl:apply-templates/>
            </xsl:element>
        </xsl:element>
    </xsl:template>

    <xsl:template match="article | article-de-readressage" mode="regroupement">
        <xsl:variable name="initiale" select="substring(descendant::item-en-adresse, 1, 1)"/
><!-- Mémorisation de la 1ère lettre de l'item en adresse de l'article ou article de réadressage
manipulé -->
```

---

[6] Pourquoi créer un élément HTML <li> ? Où son père <ul> est-il créé ?

 se trouve dans la balise <xsl:template match="composant-d-identification" mode=" index"> ce qui permet de représenter les différents éléments d'une liste, liste non ordonnée précédemment générée avec la balise <ul> dans l'élément qui a permis la création de l'index.

[7] Quelle est la fonction d'un élément HTML <a> doté d'un @href ?

Le @href lié à une ancre matérialisée par <a> permet la création d'un lien hypertexte vers une page web ou un fichier. Dans le document, ci-présent, cette ancre sert à créer un lien de l'index vers le texte.

[8] Pourquoi faut-il que la valeur de l'@href débute par « # » dans ce contexte ?

Dans ce contexte, le « # » permet de préciser là où nous nous trouvons.

```
        <xsl:if test="not(preceding::item-en-adresse[substring(., 1, 1)=$initiale])">9
                <xsl:text>
</xsl:text>
                <xsl:element name="div">
                        <xsl:attribute name="class"><xsl:text>section_alphabétique</
xsl:text></xsl:attribute>
                        <xsl:element name="h1"><xsl:value-of select="$initiale"/></
xsl:element>
                        <xsl:apply-templates select=". | following-
sibling::*[substring(descendant::item-en-adresse, 1, 1)=$initiale][substring(local-name(), 1,
7)='article']"/>10
                </xsl:element>
            </xsl:if>
    </xsl:template>

    <xsl:template match="composant-d-identification">
        <xsl:variable name="ID_élément_courant" select="generate-id()"/>
        <xsl:element name="div">
            <xsl:attribute name="class">
                <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:element name="a">
                <xsl:attribute name="id">
                    <xsl:value-of select="$ID_élément_courant"/>
                </xsl:attribute>
                <xsl:attribute name="href">
                    <xsl:text>#</xsl:text>
                    <xsl:text>index_</xsl:text>
                    <xsl:value-of select="$ID_élément_courant"/>
                </xsl:attribute>
                <xsl:apply-templates/>
            </xsl:element>
        </xsl:element>
```

---

9 Que testons-nous ? Pourquoi faire ceci ?

Ici, nous faisons un test de position permettant de voir s'il n'y a pas d'élément précédent qui se nomme <item-en-adresse > par rapport aux éléments matchés: "article | article-de-redressage". À condition que le premier caractère du contenu textuel récupéré par le substring soit égal à la variable initiale définie juste au dessus.

10 Que fait cet <xsl:apply-templates> ?

Cette expression dit de traiter les éléments actuellement manipulés (.) ou (|) tous les éléments (*) frères postposés (following-sibling::*) à condition que la chaîne de caractère récupérée contienne au moins un élément descendant « item-en-adresse » et que son contenu soit égal à la variable initiale stockée. Une autre condition s'ajoute, celle que le nom de ces éléments corresponde à une chaîne de caractères qui commence à la septième position et soit égal au string 'article'. Ce template a pour finalité de créer différentes sections alphabétiques des noms commençant par A et de ceux commençant par B.

```
    </xsl:template>

    <xsl:template match="article | article-de-readressage">
        <xsl:text>
    </xsl:text>
        <xsl:element name="article">
            <xsl:attribute name="class">
                    <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="composant-de-traitement">
        <xsl:text>
    </xsl:text>
        <xsl:element name="div">
            <xsl:attribute name="class">
                    <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

<xsl:template match="unité-de-traitement | information-derivationnelle | remarque | renvoi-
vers-articles | legende-d-image">
        <xsl:text>
        </xsl:text>
        <xsl:element name="p">
            <xsl:attribute name="class">
                    <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="image">
        <xsl:element name="img">
            <xsl:attribute name="src">
                    <xsl:value-of select="@fichier-source"/>
                    <xsl:text>.jpg</xsl:text>
            </xsl:attribute>
            <xsl:attribute name="alt">
                    <xsl:value-of select="@fichier-source"/>
```

```
                <xsl:text> : </xsl:text>
                <xsl:value-of select="following-sibling::legende-d-image[1]"/>
            </xsl:attribute>
        </xsl:element>
    </xsl:template>

    <xsl:template match="item-en-adresse[following-sibling::*[1][local-name()='separateur-de-formes']] | item-defini[following-sibling::node()[1][.='.']]">[11]
        <xsl:element name="span">
            <xsl:attribute name="class">
                <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
    </xsl:template>

    <xsl:template match="*">
        <xsl:element name="span">
            <xsl:attribute name="class">
                <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:apply-templates/>
        </xsl:element>
        <xsl:text> </xsl:text>
    </xsl:template>

    <!-- Pour créer des liens de chaque antonyme mentionné dans les articles vers leur article -->
    <xsl:template match="synonyme | antonyme | derive">
        <xsl:variable name="item_cherché"><xsl:value-of select="."/></xsl:variable>
        <xsl:element name="span">
            <xsl:attribute name="class">
                <xsl:value-of select="local-name()"/>
            </xsl:attribute>
            <xsl:element name="a">
                <xsl:attribute name="href">
                    <xsl:text>#</xsl:text>
```

[11] Que peut traiter cet <xsl:template> ? Pourquoi ces nœuds ont-ils besoin d'un traitement différent de celui du <xsl:template> qui matche "*" ?

Cet <xsl:template> traite les « item-en-adresse », à condition que ces derniers soient suivis d'un frère postposé, c'est-à-dire n'importe quel noeud à partir du premier et à condition aussi que le local-name soit égal à « séparateur-de-formes ». Quant à l' « item-défini », il faut qu'il soit suivi aussi d'un frère postposé (noeud ou chaîne de caractères) figuré par le « node », ce dernier doit être le premier noeud. De plus, l'élément courant ou en cours de traitement doit être égal à lui-même.

```
                        <xsl:value-of select="generate-id(/descendant::composant-d-
identification[item-en-adresse[.=$item_cherché]])"/>[12]
                    </xsl:attribute>
                    <xsl:apply-templates/>
                </xsl:element>
            </xsl:element>
        </xsl:template>
</xsl:stylesheet>
```

[12] En fonction de ce qui est cherché, pourquoi le chemin exprimé part-il de la racine du document «(/)» et pas de l'élément courant ?

Le chemin exprimé part de la racine du document « (/) » et non pas de l'élément courant car cela nous permet de parcourir l'ensemble des articles, tout en créant un lien hypertexte entre les items en cours de traitement (traités par le <xsl:template match="synonyme | antonyme | derive ">) et leur article.