## Conditions de réalisation

Ce travail peut être réalisé individuellement ou à deux (voire trois, en reprenant ou pas les groupes de XML). Il est à rendre pour le mercredi 23/02/2022 (et, avec tiers-temps pour les étudiants qui en bénéficient, le 09/03/2022).

## Documents à déposer

.zip Les différents fichiers (XML, XSLT, HTML et CSS, et PDF) devront respecter les consignes de nommage et être réunis dans un dossier compressé de format ZIP déposé sur Moodle (par l'étudiant du groupe dont le nom est le premier par ordre alphabétique) :

- 🗁 **Q1**_*nom(s).zip* ← *Sarrasine_TEI_nom(s).xml*
  - ← *XML-TEI_vers_HTML_nom(s).xml*
  - ← *Sarrasine_nom(s).html* et *Sarrasine_nom(s).css*
- 🗁 **Q2**_*nom(s).zip* ← *recette_nom(s).xml*
  - ← *XML_vers_HTML_nom(s).xml*
  - ← *recette_nom(s).html* et *recette_nom(s).css*
- 🗁 **Q3**_*nom(s).zip* ← *XSLT_avec_commentaires_nom(s).pdf* [réponse à la question 2 complétée à la fin de ce sujet]

## 1 Élaborer une version XHTML par transformation du fichier XML de *Sarrasine* produit pour l'évaluation XML (Q2)

Vous traiterez le texte de Sarrasine à partir de votre fichier XML TEI (de l'évaluation XML du S1).

Vous devrez donner à lire soit l'édition Furne soit l'édition Houssiaux soit les deux (avec matérialisation typographique de ce qui revient à chaque édition) et permettre de circuler d'un fichier à l'autre par un jeu de liens hypertextes.

Chaque document HTML contiendra un texte lisible (au sein duquel les mots des fins et débuts de nœuds *text()* ne seront pas collés) et mis en forme en employant des éléments HTML , <p> ou <q> (pour les tours de parole), et , tous porteurs d'@class dont les valeurs sont associées à des instructions de mise en forme dans une feuille de styles CSS.

Les instructions CSS donneront par exemple une coloration distinctive aux divers noms d'entités, des trames de fonds ou encadrements…, mais ce sera le XSLT qui insérera les titres additionnels et intégrera le contenu du document XML traité.

Chaque document HTML comportera, en marge gauche (position définie dans la CSS), un index de toutes les occurrences des divers noms d'entités <placeName>, <persName>, etc. de votre document avec tri principal par @key, si c'est pertinent, et par graphie de la dénomination. Si votre document XML ne comporte pas d'@key, la valeur de l'attribut sera considérée comme vide.

Les items de chaque liste d'index porteront des liens hypertextes vers le texte complet.

La création de l'index des <placeName> triés par graphie et des <persName> triés par @key puis, secondairement, par graphie est fournie dans le fichier de la transformation (vue aux derniers cours) : *XMLversHTML_index_persName@key+texte_complet+structure_HTML_2021-12-*

*[…].xsl*. Le placement de la division contenant les index en marge gauche du texte complet est régi par les instructions CSS vues dans les fichiers *Robert-Google.css* et *Sarrasine.css*.

Ce sera autant la qualité de votre code XSLT que de votre rendu HTML + CSS (élégant et digne d'une édition universitaire savante) qui entrera dans la notation.

## 2 Élaborer une version XHTML par transformation du fichier XML d'une recette de cuisine produit pour l'évaluation XML (Q1)

Vos recettes de cuisine structurées en XML pour l'évaluation du S1 ont des balisages XML non respectueux d'un standard comme la TEI. La transformation ne doit donc pas s'attendre à trouver des noms d'éléments décrits dans une documentation en ligne.

Votre transformation XML non TEI vers XHTML offrira à la lecture le contenu de votre recette, avec une mise en forme compatible avec ce type de production éditoriale et gérée par des instructions CSS.

Votre objectif est de permettre une lecture aisée de ce texte qui doit guider vos lecteurs dans la réalisation d'une activité culinaire de détente.

Ce sera autant la qualité de votre code XSLT que de votre rendu HTML + CSS (clair et attractif) qui entrera dans la notation.

## 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) : .....

Ficher *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]
```

[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.

```
    <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>

    <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>
            <!-- Restitution du texte complet -->
            <xsl:element name="div">
```

[2] Qu'indique ceci ?
[3] Pourrions-nous alternativement créer cet élément dans un autre <xsl:template> présent ou à ajouter ? Si oui, celui de quel élément ? Pourquoi ?
[4] Pouvons-nous coder cet axe d'une autre manière (non alphabétique) ? Quelle serait alors la valeur de l'@select ?
[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 ?

```
                <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 -->
        <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>
```

[6] Pourquoi créer un élément HTML <li> ? Où son père <ul> est-il créé ?
[7] Quelle est la fonction d'un élément HTML <a> doté d'un @href ?
[8] Pourquoi faut-il que la valeur de l'@href débute par « # » dans ce contexte ?
[9] Que testons-nous ? Pourquoi faire ceci ?

```
                    <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>
    </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">
```

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

```
            <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>
```

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

```
    </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>
                            <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 ?