Technique et chaîne de publication électronique avec XSLT

Jean-Damien Généro

2025, 1er dec. - 2026, 20 janv.

École nationale des chartes - PSL. M2 TNAH

Plan

Contact et objectifs

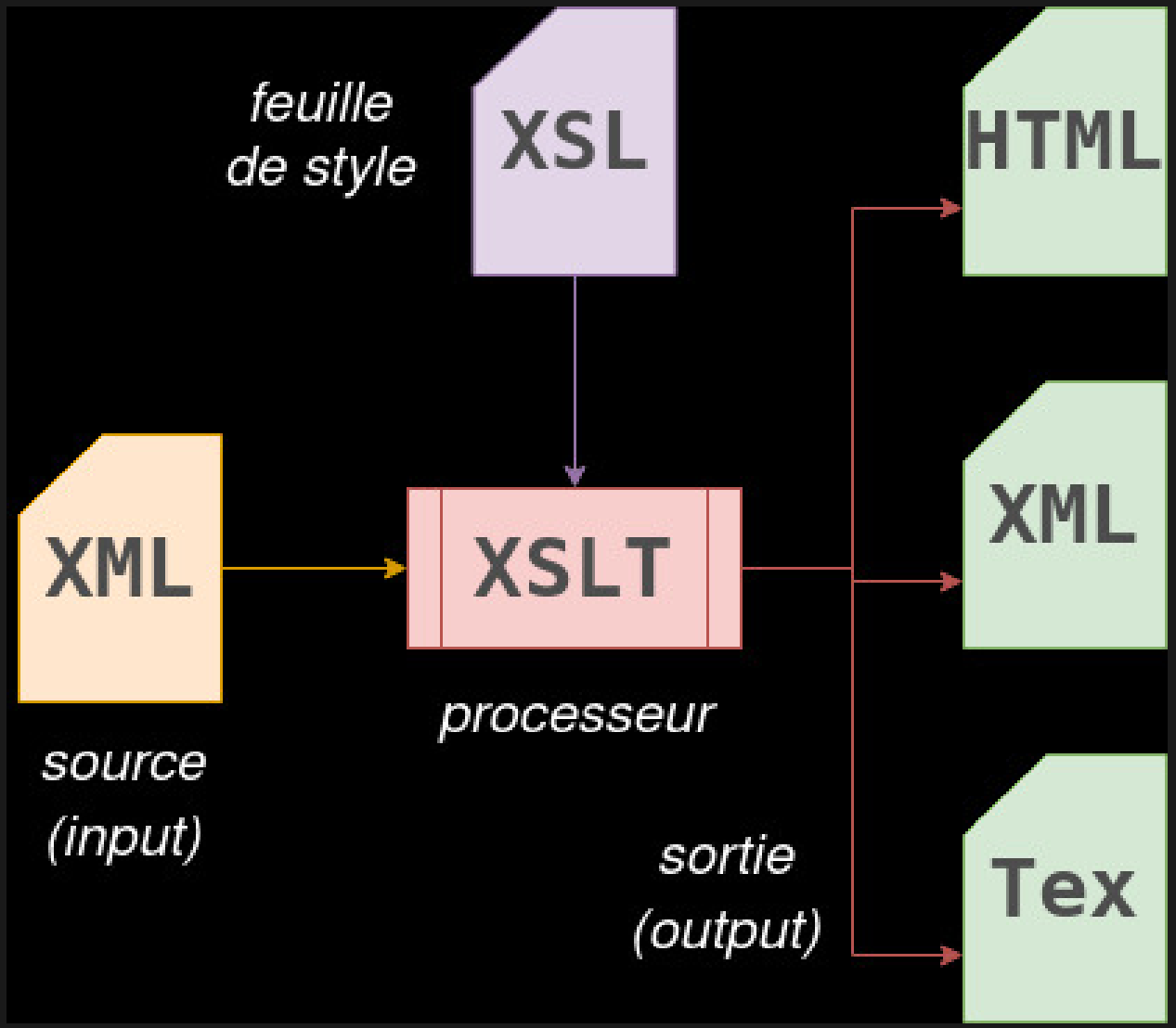
- jean-damien.genero@cnrs.fr
- XPath
  - Naviguer dans un arbre XML
  - Manipuler les principales fonctions XPath
- XSLT
  - Manipuler les règles (*templates*) basiques
  - Manipuler les conditions et les boucles
- Édition numérique
  - Transformer un document XML en un autre document XML
  - Transformer un document XML en un document HTML
  - Transformer un document XML en un document LaTeX
  - Coder une chaîne de publication électronique utilisant XSLT avec Python

1. Écosystème XML

- XML $\rightarrow$ *langage de balisage qui encode une description de la mise en page et de la structure logique d'un document*
  - XPath $\rightarrow$ *langage de requête pour naviguer dans la structure hiérarchique d'un document XML à l'aide de chemins*;
- XSL $\rightarrow$ *spécifications pour écrire des feuilles de style (stylesheet)*;
- XSLT $\rightarrow$ *un langage conçu pour transformer des documents XML en d'autres documents selon les spécifications XSL.*

W3C Recommendations : XML, XPath, XSL, XSLT.

Écosystème XML/ Schéma



- Navigation XPath :
  - Doc XML = structuré, on parle d'« **arbre** » (*XML tree*) ;
  - **Itinéraire** vers une balise (« **parent** » $\rightarrow$ « **enfant** ») ;
  - Rédaction dans une **syntaxe propre**.
- Règle XSL (*template*) :
  - **Agir** sur une ou plusieurs balises (*transform*) ;
  - Exemples : **copier** balise + contenu, copier uniquement le contenu et l'**insérer** dans une nouvelle balise, **ajouter** ou **supprimer** un attribut, etc. ;
  - Rédaction dans une **syntaxe XML** (langage à balises).
- Utilisation avec d'autres langages
  - *Python* : librairie `lxml`.

Écosystème XML/ Exemple d'un chemin XPath dans un *XML tree*

- Comment accéder aux balises *<p>* avec XPath ?

```
<TEI>
    <teiHeader/>
    <text>
        <body>
            <div>
                <p>title</p>
                <p>text</p>
            </div>
        </body>
    </text>
</TEI>
```

- *<TEI>* → *<text>* → *<body>* → * * → *<p>*
- Traduction XPath : */TEI/text/body/div/p*
- Simplification : *//body/div/p* (/~~TEI/text~~/body/div/p)

Écosystème XML/ Exemple d'une règle XSL

- Comment appliquer une règle XSL aux balises *<p>*?
  - Une règle XSL est toujours contenue dans une balise *<xsl:template/>* possédant un *@match*.
- Indiquer le chemin XPath vers les *<p>* dans l'*@match*.

```
<xsl:template match="/TEI/text/body/div/p">
    <xsl:copy-of select="."/>
</xsl:template>
```

ou

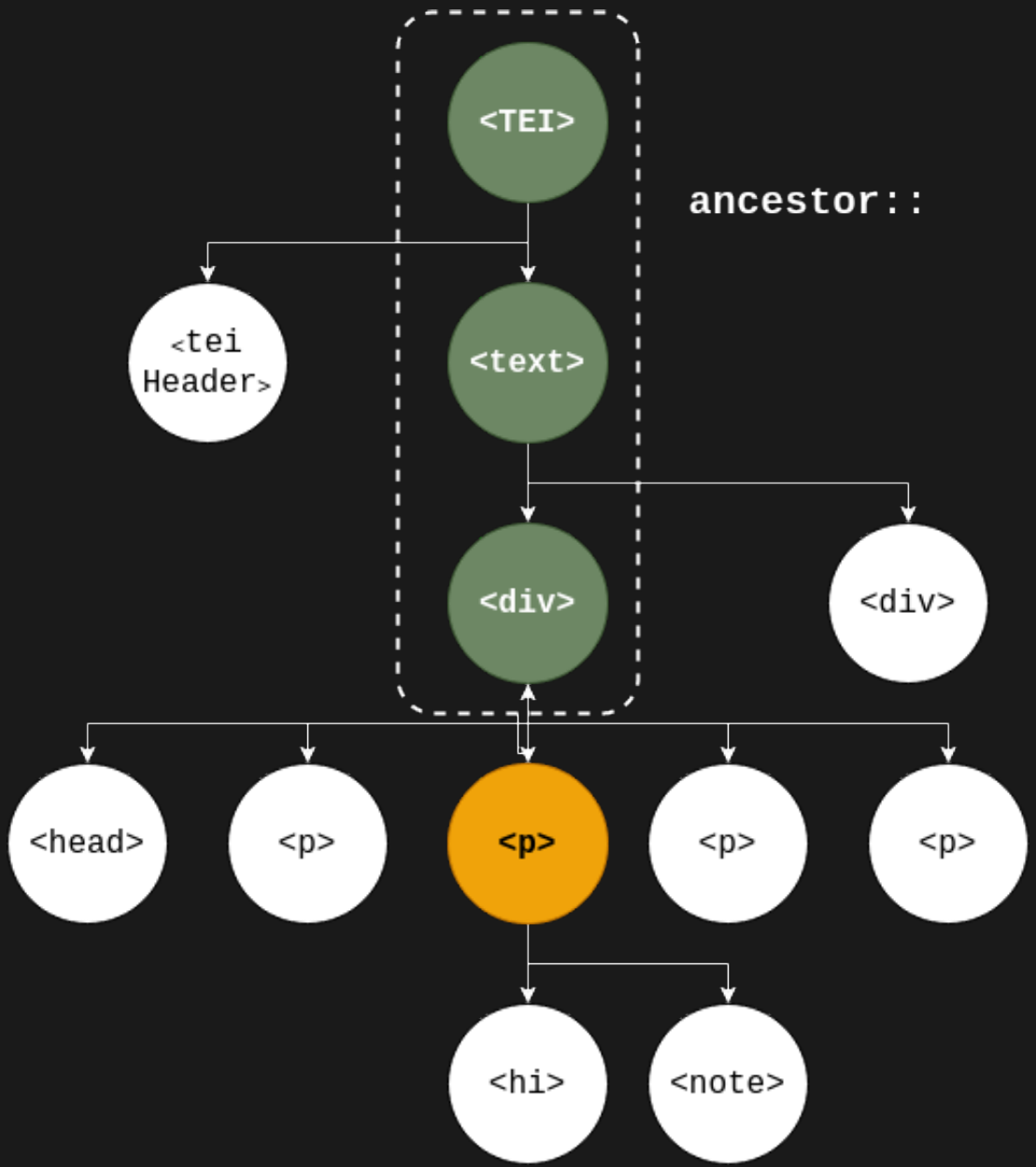
```
<xsl:template match="//body/p">
    <xsl:copy-of select="."/>
</xsl:template>
```

2. XPath

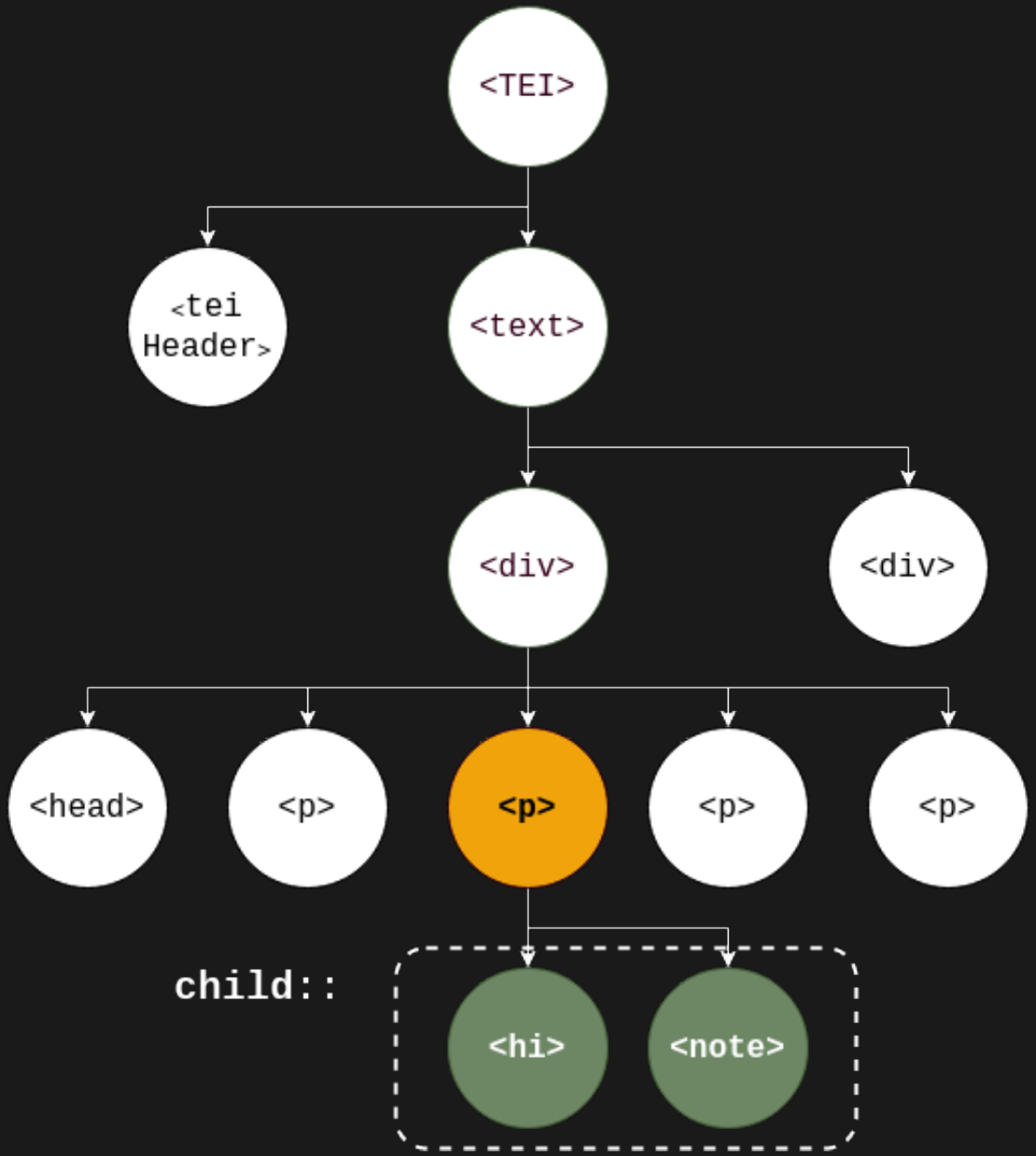
- **Nœud** (*node*) (7) = composant de l'arbre : *root*, *element*, *attribute*, *text*, *comment*, *namespace* et *processing instruction*.
- Deux rôles : ***current node*** (fixe, le premier du chemin) ou ***context node*** (variable, celui que XPath évalue à l'instant *t*);
- Écrire un chemin (*expression*) : succession de nœuds séparés par l'**opérateur /** ;
  - Tester dans Oxygen :
    `/TEI/text/body/div/p`

- Un axe permet de qualifier la **relation** entre le **nœud de contexte** et un ou des **autre(s) nœud(s)**.
- Direction basique : **parent** $\rightarrow$ **enfant**.
- 13 axes XPath, dont :
  - *`parent::`* $\rightarrow$ nœud immédiatement au-dessus ;
  - *`child::`* $\rightarrow$ nœud immédiatement en-dessous ;
  - *`following-sibling::`* $\rightarrow$ nœud(s) après celui de contexte et qui ne fait/font pas partie de ses descendants ;
  - *`ancestor`*, *`descendant`*, *`preceding`*, etc.

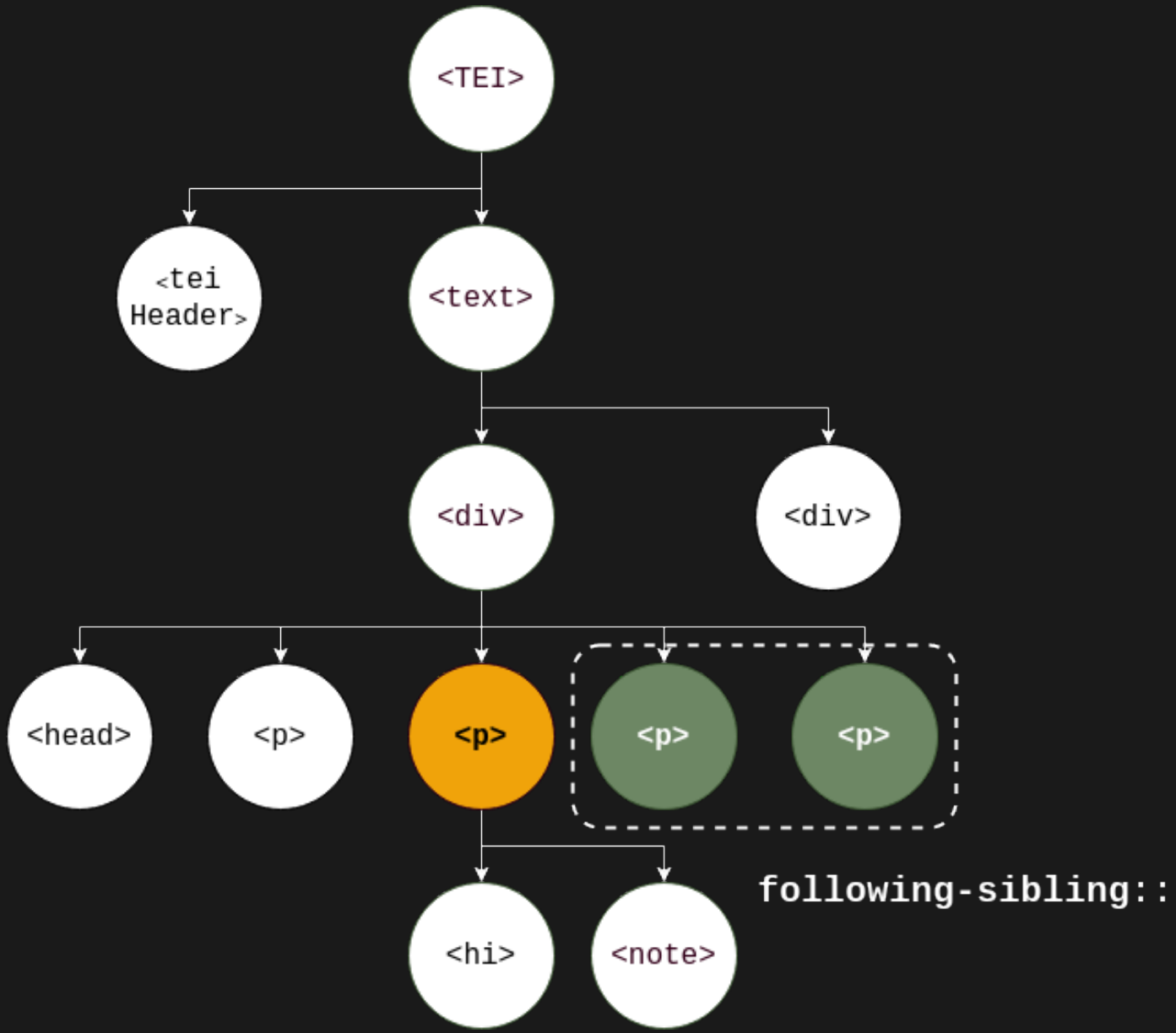
XPath/ Axe `ancestor::`



XPath/ Axe `child::`



XPath/ Axe following-sibling::



XPath/ Abbréviations des axes

- 1er élément de l'arbre (*root node*) → `/` (*attention : différent de la racine* `<TEI>`)
- `decendant-or-self::` → `//`
- `self::` → `.`
- `attribute::` → `@`

- Donc : vous pouvez écrire `//div/@n` au lieu de `//div/attribute::n`.

Prédicats : définition

- Prédicat (filtre) = **condition** qui doit être satisfaite par le nœud de contexte (existence d'un attribut, valeur d'un attribut, position d'une balise, etc.).
- Écrit entre crochets *[ ]*.

- *//div[@n='2']/head* = le *<head>* de la * * avec un *@n* de valeur *2*.

- *//body/div[@n='2']/p[2]* = *?*

- *//tag[position()=2]* ou *//tag[2]* → condition de position (*<tag>* n°2);
- *//tag[last()]* → dernier *<tag>*;
- *//tag[@foo='bar']* → expression logique (*<tag>* avec un *@foo* dont la valeur est *bar*);

3. XSLT. Élément racine et instructions de premier niveau

XSLT/ Définition

- XSLT $\rightarrow$ un **langage de programmation**;
- Permet de **transformer un doc XML** en un autre doc (*.xml*, *.html*, *.tex*, etc.);
- Transformation opérée par un **processeur XSLT**
  - **Construit** l'arbre output, **transforme** l'arbre input et **sérialise** le document output;
  - Le plus connu $\rightarrow$ **Saxon** (en ligne de commande);
  - Généralement **intégré** à des logiciels (**Oxygen**) ou des librairies (**python : lxml**).

- Feuille de style XSL == un doc XML `.xsl` ;
- Contient des instructions ou règles (*templates*) ;
- Élément racine : `<xsl:stylesheet>`

- Dans Oxygen → ouvrir un nouveau doc « XSLT Stylesheet ».
- Observer les attributs de l'élément racine.

Élément racine : `<xsl:stylesheet>` (1/4)

Espaces de nom (*namespace*) et attributs présents de base dans Oxygen :

- *`@xmlns:xsl`* : espace de nom XSL;
- *`@xmlns:xs`* : espace de nom XML;
- *`@xmlns:math`* : espace de nom math (XPath 3);
- *`@exclude-result-prefixes`* : liste des préfixes qui ne seront pas copiés dans l'output;
- *`@version`* : version de XSL (1, 2 ou 3).

Attributs à remplacer dans Oxygen *pour une transformation vers XML-TEI* :

- remplacer *`@xmlns:xs`* par *`@xmlns:tei`* → espace de nom de l'élément racine de l'output.
  - évite l'ajout de *`tei:`* à chaque élément "matché" par une règle (*`@match`*).

- remplacer *`xs`* par *`tei`* dans *`@exclude-result-prefixes`*.

Élément racine : `<xsl:stylesheet>` (3/4)

Attributs à ajouter dans Oxygen *pour une transformation vers XML-TEI* :

- *`@xpath-default-namespace`* : espace de nom des chemins XPath de la feuille de style (http ://www.tei-c.org/ns/1.0).
  - évite de devoir écrire le préfixe *`tei:`* devant chaque nœud d'une expression XPath.
- *`@xmlns`* avec l'adresse de l'espace de nom TEI.
  - détermine l'espace de nom de l'ensemble du document de sortie.

Élément racine : `<xsl:stylesheet>` (4/4)

- **attention** : TEI est un standard XML parmi d'autres ! Pour EAD, changer l'espace de nom TEI par celui d'EAD et le préfixe *`tei:`* par *`ead:`*.
- Atributs pour la transformation vers HTML ou LaTeX :
  - *`@xmlns:xsl`* (adresse de l'espace de nom XSLT)
  - *`@xpath-default-namespace`* (adresse de l'espace de nom TEI)
  - *`@version`* (version de XSL utilisée)

XSLT/ Instruction de 1er niveau : `<xsl:output>`

- Instruction de premier niveau ;
- « contrôle les caractéristiques du document de sortie ».

```
<xsl:output
    method="xml | html | text"
    indent="yes | no"
    omit-xml-declaration="yes | no"
    encoding="UTF-8"
/>
```

XSLT/ Écriture d'une règle : `<xsl:template>` (1/4)

- une règle peut être écrite de plusieurs manières;
- *`<xsl:template>`* → définir une règle;
- Possède un *`@match`* avec pour valeur un chemin XPath qui donne l'emplacement du nœud sur lequel sera appliquée la règle.
- Un doc peut être composé d'une seule règle ou d'une succession de règle. Important → **cohérence des XPath**.

 Écriture d'une règle : `<xsl:template>` (2/4)

- 1. Appliquer une règle vide sur la racine :

```
<xsl:template match="/">
</xsl:template>
```

- 2. Appliquer la règle *copy-of* sur la racine :

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

 Écriture d'une règle : `<xsl:template>` (3/4)

- Effet d'une règle vide $\rightarrow$ la partie de l'arbre d'entrée sélectionnée n'est pas copiée dans l'arbre de sortie.
- Effet de *<xsl:copy-of/>* $\rightarrow$ copie à l'identique la balise matchée par le @select et ses nœuds enfants ;
- **Comment copier le `<text>` dans l'output, mais sans le `<teiHeader>` ?**

 Écriture d'une règle : `<xsl:template>` (4/4)

- Comment obtenir cet arbre en output ?

```
<div>
    <head>Extrait du Journal de
          Jean Le Fèvre</head>
    <div n="1">
        <!-- copie de la div n°1 de l'input -->
    </div>
    <div n="2">
        <!-- copie de la div n°2 de l'input -->
    </div>
</div>
```

4. XSLT. Règles basiques

XSLT/ <xsl:template>

- Un *<xsl:template>* contient une ou plusieurs règles ;
- L'*@match* contient le chemin XPath vers la balise qui sera le point de départ de la ou des règles, qui peut s'appliquer à :
  - La balise désignée dans le *@match* ;
  - Les enfants ou descendants de cette balise ;
  - N'importe quelle balise de l'arbre *via* des axes (*ancestor*, *preceding*, etc.).
  - La partie de l'arbre sur laquelle la règle s'applique est en général indiquée dans le *@select* de l'instruction XSL (*cf. infra*).

XSLT/ Contenu d'un `<xsl:template>`

- *`<xsl:template>`* peut contenir :
  - Des balises XML ou HTML (avec leurs @tt);
  - Du texte (commandes LaTeX);
  - Des instructions ou variables/param XSL : *`<xsl:.../>`*;

Contenu d'un `<xsl:template>` : fichier d'exemple

- Ouvrir Oxygen.
- Désactiver le paramètre par défaut :
  Preferences > *XML* > *XML Parser* > *RELAX NG*,
  décocher *Add default attributes*

- Ouvrir *`class_2_journal_lefevre.xml`*
- Ouvrir *`class_2_ex_instructions.xsl`*

XSLT/ Contenu d'un `<xsl:template>` : XML, HTML, LaTeX (1/2)

- Des balises peuvent être écrites directement dans un *`<xsl:template>`* :

```
<xsl:template match="//text/body/div[1]">
    <body>
        <div>
            <head type="chap">
                <xsl:value-of select="./head"/>
            </head>
            <xsl:copy-of select="./p"/>
        </div>
    </body>
</xsl:template>
```

XSLT/ Contenu d'un `<xsl:template>` : XML, HTML, LaTeX (2/2)

- Les balises XML ou HTML peuvent être écrites dans des *<xsl:element name="tag">* et les attributs dans des *<xsl:attribute name="@tt">* :

```
<xsl:template match="//body">
    <xsl:element name="div">
        <xsl:element name="head">
            <xsl:attribute name="type">chap</xsl:at
            <xsl:value-of select="./p[1]"/>
        </xsl:element>
        <xsl:copy-of select="./p[2]"/>
    </xsl:element>
</xsl:template>
```

XSLT/ Contenu d'un `<xsl:template>` : du texte (1/2)

- Du texte peut être mis directement dans une règle : remplacer le contenu d'une balise, écrire le préambule d'un doc LaTeX.

```
<xsl:template match="/">
    \documentclass[a4paper]{book}
    \usepackage[utf8]{inputenc}
    \usepackage[french]{babel}
    \usepackage{fontspec}
    \begin{document}
        <xsl:apply-templates/>
    \end{document}
</xsl:template>
```

XSLT/ Contenu d'un `<xsl:template>` : du texte (2/2)

- Du texte peut aussi être mis dans un *`<xsl:text>text</xsl:text>`* :

```
<xsl:template match="/TEI/text/body/div[1]">
    <body>
        <div>
            <head type="chap">
                <xsl:text>Chapitre n°1. </xsl:text>
                <xsl:value-of select="./head"/>
            </head>
            <xsl:copy-of select="./p"/>
        </div>
    </body>
</xsl:template>
```

5. XSLT. Les quatre principales instructions

XSLT/ Les quatre principales instructions XSL

- *<xsl:copy/>*
- *<xsl:copy-of/>*
- *<xsl:value-of/>*
- *<xsl:apply-templates/>*

Principales instructions (1/4) : `<xsl:copy>`

- Copie de la balise matchée par le `@match`, sans les *namespaces*, attributs, texte, etc.
- *`<xsl:copy/>`* peut contenir d'autres règles, du texte, etc.
- Intérêt → copier un élément pour appliquer des règles à ses enfants.

```
<xsl:template match="/TEI/text/body">
    <xsl:copy>...</xsl:copy>
</xsl:template>
```

 Principales instructions (2/4) : `<xsl:copy-of>`

- Copie à l'identique de la balise matchée par le `@select` et de ses nœuds enfants (balises, attributs, textes).
- Élément vide (pas de règles internes).
- **Impossible de modifier les éléments copiés !**
- Intérêt → reproduire rapidement une partie de l'arbre que l'on ne veut pas modifier.

```
<xsl:template match="/TEI/text/body">
    <xsl:copy-of select="."/>
</xsl:template>
```

 Principales instructions (3/4) : `<xsl:value-of>`

- Renvoie uniquement la valeur textuelle de la balise matchée par le `@select`.
- Élément vide (pas de règles internes).
- Intérêt → copier du texte sans les balises (utile pour LaTeX).

```
<xsl:template match="/TEI/text/body">
    <xsl:value-of select="."/>
</xsl:template>
```

XSLT/ Principales instructions (4/4) : `<xsl:apply-templates/>`

- Ouvrir *att_apply_templates_class2.xsl*
- *<xsl:apply-templates/>* est une **instruction récursive** : le processeur va **examiner les nœuds enfants** de la balise matchée par le *@match* dans l'ordre et **appliquer les règles** qui leur sont associées.

XSLT/ L'élément `<xsl:apply-templates/>` (1/4)

- Sans elle, le processeur s'arrête à l'emplacement désigné par le *@match* du *<xsl:template/>* et ne traite pas les éléments enfants.
- Exemple : la balise *<TEI/>* (*cf.* fichier).

```
<xsl:template match="/">
    <TEI>
        <xsl:apply-templates/>
    </TEI>
</xsl:template>
```

XSLT/ L'élément `<xsl:apply-templates/>` : `@select` (2/4)

- *`@select`* permet d'appliquer les règles uniquement au nœud sélectionné.

- Ex. → inverser *`<teiHeader/>`* et *`<text/>`* (*cf.* fichier) :

```
<xsl:template match="/">
    <TEI>
        <xsl:apply-templates select="//text"/>
        <xsl:apply-templates select="//teiHeader"/>
    </TEI>
</xsl:template>
```

XSLT/ L'élément `<xsl:apply-templates/>` : `@mode` (3/4)

- *`@mode`* permet d'appliquer des règles différentes **à un même élément XML** en fonction de son emplacement ou de son contenu (= son « mode ») *sans avoir à mettre ces règles dans un seul `<xsl:template/>`*.
- Le *`@mode`* doit être présent et dans le *`<xsl:apply-templates/>`* et dans le *`<xsl:template/>`* avec la même valeur (= le nom du mode).

XSLT/ L'élément `<xsl:apply-templates/>` : `@mode` (4/4)

- Exemple d'utilisation de *`@mode`* : créer une table des matières (*cf.* fichier).

```
<xsl:template match="//body">
    <xsl:copy>
        <xsl:apply-templates />
        <div>
            <head>Table des matières</head>
            <xsl:apply-templates mode="toc"/>
         </div>
    </xsl:copy>
</xsl:template>
```

XSLT/ Une autre instruction : `<xsl:number/>` (1/2)

- *`<xsl:number/>`* « compte des éléments de façon séquentielle ».
- Attributs :
  - *`@count`* → définit les éléments de l'input qui seront numérotés dans l'output, avec une expression XPath
  - *`@level="single|multiple|any"`* → niveaux de l'arbre pris en compte pour le comptage ;
  - *`@format="1|01|A|a|I|i"`* → format des numéros.

XSLT/ Une autre instruction : <xsl:number/> (2/2)

```
<xsl:template match="//body//p">
        <xsl:copy>
            <xsl:attribute name="n">
                <xsl:number
                    count="//div[@n]/p"
                    level="any"
                    format="1"/>
            </xsl:attribute>
            <xsl:value-of select="."/>
        </xsl:copy>
</xsl:template>
```

- XSLT commence par chercher la règle à appliquer au nœud racine ;
- Cette règle fait appel à d'autres avec *`apply-templates`* (avec ou sans @tt), elles sont appliquées dans l'ordre ;
- S'il n'y a pas de règle, il passe au suivant.

- Donc : l'ordre d'écriture des règles n'a pas importance ;
- **Attention → restez lisible + commentez votre code !**

- Si plusieurs règles s'appliquent à un même nœud $\rightarrow$ **une seule règle est exécutée**.
- XSLT examine la **priorité** de chaque règle : plus un match est spécifique, plus il a de chances d'être choisi :
  - Priorité par défaut : *`node()`*, *`text()`* et `*` à -0.5 (générique); les balises à 0.5 (spécifique).
  - Priorité explicite : *`@priority`*.

6. XPath. Les fonctions

- Les fonctions peuvent être utilisées dans des expressions XPath ou dans des prédicats :
  - Expression $\rightarrow$ `count(./p)`
  - Prédicat (`[ ]`) $\rightarrow$ `./[count(p)]`
- Cf. fiche : fichier xpath-nodes-axis-functions.

7. XSLT. Variables et paramètres

XSLT/ Différence entre `variable` et `param`

- Rappel → variable : nom (unique) + valeur (statique ou dynamique).
- Dans XSLT, portée globale (premier niveau) ou locale (intégrée à un *`xsl:template`*);
- *`<xsl:variable/>`* : **valeur fixe** (*NB : la valeur courante peut changer au sein d'une itération)*;
- *`<xsl:param/>`* : **valeur fixe ou dynamique**, peut être pris en argument par le processeur (ex : saxon dans lxml).

XSLT/ `<xsl:variable/>` (1/3)

- Attributs : *`@name`* (*req.*) et *`@select`* (*opt.*);
- Deux manières de définir la valeur de *`<xsl:variable/>`* :
  - Par la valeur de l'*`@select`* (expression XPath ou texte);
  - Par le contenu de *`<xsl:variable/>`* (règle ou texte).

```
<xsl:variable name="w" select="Xpath"/>
<xsl:variable name="x">règle/texte</xsl:variable>
[<xsl:variable name="y" select="'texte'"/>]
[<xsl:variable name="z" select="1234"/>]
```

XSLT/ <xsl:variable/> (2/3)

- Pour appeler une variable : *$* + *@name* dans le *@select* d'un *<xsl:value-of/>* (uniquement le texte) ou d'un *<xsl:copy-of/>* (texte + balises).

```
<xsl:template math="/">
    <xsl:value-of select="$x"/>
    <xsl:copy-of select="$y"/>
</xsl:template>
```

- Possibilité de les ajouter directement comme valeur d'attribut avec des *{ }*.

```
<xsl:template math="/">
    <TEI xml:lang="{$language}">...</TEI>
</xsl:template>
```

XSLT/ <xsl:param/>

- Fonctionne de la même manière que *<xsl:variable/>*.
- En utilisation locale, doit être le premier enfant de *<xsl:tempate/>*.

8. XSLT. Conditions

- Deux moyens d'exprimer une condition en XSLT : *<xsl:if/>* et *<xsl:choose/>*;
- Attention : les conditions XSLT ne fonctionnent pas de la même manière que celles d'autres langages de programmation, comme Python où l'on trouve le triplet *if*, *elif* et *else*.

XSLT/ `<xsl:if/>` (1/2)

- *`<xsl:if/>`* possède un *`@test`* obligatoire ;
- Contenu appliqué si le *`@test`* est validé.

- Pas de *`elif`* : utiliser plusieurs *`<xsl:if/>`* ;
- Pas de *`else`* : utiliser *`<xsl:choose/>`*.

XSLT/ <xsl:if/> (2/2)

Exemple : dans le *Journal*, ajouter un *@n* aux *<p/>* si ces derniers sont dans la ** ** nº 2 :

```
<xsl:template match="//text//div/p">
    <p>
        <xsl:if test="parent::div/@n = 2">
            <xsl:attribute name="n">
                <!-- à vous de compléter -->
            </xsl:attribute>
        </xsl:if>
        <xsl:value-of select="."/>
    </p>
</xsl:template>
```

- *<xsl:choose/>* a pour enfant un ou des *<xsl:when/>* (req.) et un seul *<xsl:otherwise/>* (opt.).
- *<xsl:when/>* a un *@test* obligatoire (même fonctionnement que *<xsl:if/>*);
- si aucun *<xsl:when/>* n'est réalisé, alors *<xsl:otherwise/>* est activé

9. XSLT. Boucles et tri

XSLT/ Boucle simple : `<xsl:for-each/>` (1/2)

- *`<xsl:for-each/>`* a un *`@select`*;
- Itération sur chaque nœud désigné par le *`@select`*, la règle contenue dans le *`<xsl:for-each/>`* est appliquée;
- Le point de départ des chemins Xpath au sein d'un *`<xsl:for-each/>`* est la balise sélectionnée par le *`@select`*;
- Peut intégrer d'autres instructions XSL comme *`<xsl:if/>`* et *`<xsl:choose/>`*.

XSLT/ Boucle simple : `<xsl:for-each/>` (2/2)

- Exemple :

```
<xsl:for-each select="//persName">
  <persName>
    <xsl:value-of select="./forename"/>,
    <xsl:value-of select="./surname"/>.
  </persName>
</xs:for-each>
```

XSLT/ Tri : `<xsl:sort/>` (1/2)

- S'utilise comme premier enfant de *`<xsl:for-each/>`* ou *`<xsl:for-each-group/>`*;
- Change l'ordre d'origine des nœuds sélectionnés en un autre ordre (alphabétique, numérique, etc.);

XSLT/ Tri : `<xsl:sort/>` (2/2)

- Exemple :

```
<xsl:for-each select="//date[@when]">
  <xsl:sort select="./when"
            data-type="number"/>
  <xsl:copy-of select="."/>
</xs:for-each>
```

XSLT/ Itération sur des groupes : <xsl:for-each-group/> (1/4)

- Rassemble les nœuds (*@select*) en groupe selon un critère donné (*@group-by*) et leur applique les règles définies à l'intérieur de *<xsl:for-each-group/>*.

```
<xsl:for-each-group select="//tag"
        group-by="@att">
        <!-- règles -->
</xsl:for-each-group>
```

XSLT/ Itération sur des groupes : `<xsl:for-each-group/>` (2/4)

1. Itère sur les nœuds désignés dans le *`@select`*;
2. Regroupe ces nœuds selon le critère défini dans le *`@group-by`*;
3. Applique les règles indiquées dans le contenu de la balise.

XSLT/ Itération sur des groupes : `<xsl:for-each-group/>` (3/4)

- Plusieurs *`<xsl:for-each-group/>`* sont souvent utilisés ensemble (boucles dans une boucle);
- Ils peuvent contenir des *`<xsl:for-each/>`*, des conditions, etc...
- Attention : c'est une nouveauté de XSLT 2.0.

XSLT/ Itération sur des groupes : `<xsl:for-each-group/>` (4/4)

- *`current-group()`* retourne la liste des nœuds du groupe de l'itération en cours d'exécution ;
  - à utiliser dans le *`@select`* d'un sous-*`<xsl:for-each-group/>`* ou d'un *`<value-of/>`*.
- *`current-grouping-key()`* retourne la clef utilisée pour l'itération en cours d'exécution.
  - à utiliser dans le *`@select`* d'un *`<value-of/>`*.

10. XSLT. Enregistrement du document de sortie

XSLT/ Enregistrement : `<xsl:result-document/>` (1/2)

- *`<xsl:result-document/>`* permet de générer plusieurs documents de sortie ;
- Nouveauté de XSLT 2.0 ;
- S'utilise comme enfant d'un *`<xsl:template/>`* ou d'un *`<xsl:call-template/>`* (*cf. cours 6*).

Trois attributs :

- *`@href`* permet d'indiquer l'URI du document à générer : absolue (depuis la racine) ou relative (dans ce cas, le processeur part de l'emplacement du fichier d'entrée);
- *`@method`* permet de spécifier le format du code de sortie;
- *`@indent`* permet d'indiquer si le résultat sera indenté;

XSLT/ <xsl:result-document/>, exemple

```xml
<xsl:template match="/">
  <xsl:result-document href="chapitre1.xml"
        method="xml" indent="yes">
    <TEI><!-- règles --></TEI>
  </xsl:result-document>
  <xsl:result-document href="chapitre2.xml"
        method="xml" indent="yes">
    <TEI><!-- règles --></TEI>
  </xsl:result-document>
  <xsl:result-document href="index.xml"
        method="xml" indent="yes">
    <TEI><!-- règles --></TEI>
  </xsl:result-document>
</xsl:template>
```

XSLT/ `<xsl:result-document/>` et variables

- Inconvénient de créer X docs de sortie → répéter X fois des éléments récurrents (*`<teiHeader/>`*, header HTML, etc.);
- Stocker ces éléments dans des variables permet de ne les modifier qu'une seule fois.

XSLT/ `<xsl:result-document/>`. Notions essentielles

- *`<xsl:result-document/>`* permet de générer un ou plusieurs documents de sortie ;
- L'URI est indiquée dans un *`@href`* ;
- Les variables sont utiles pour :
  - Contenir les éléments récurrents des documents de sortie (header, footer, etc).

11. XSLT. Transformation vers HTML

XSLT/ Méthode HTML (1/2)

- Dans l'en-tête XSL : les *@xmlns:tei*, *@xmlns* et *@exclude-result-prefixes* ne sont plus nécessaires;
- Dans *<xsl:output/>* : ajouter *@method='html'*, *@omit-xml-declaration* n'est plus nécessaire.

XSLT/ Méthode HTML (2/2). Template HTML basique

```
<html lang="fr">
    <head>
        <meta http-equiv="Content-Type"
            content="text/html; charset=UTF-8">
        <!-- le 1er <meta> est ajouté par XSLT -->
        <title><!-- doc title --></title>
        <meta name="description" content="#"/>
        <meta name="author" content="#"/>
        <!-- <link rel="stylesheet" href="#"/> -->
    </head>
    <body>
        <!-- some text here -->
        <!-- <script src="#"></script> -->
    </body>
</html>
```

XSLT/ Instruction `call-template` (1/2)

- `<xsl:call-template name="#"/>`;
- Permet d'appeler autant de fois qu'on le veut une template dans un *`<xsl:template/>`*
- Cas pratiques :
  - Un *`call-template`* par *`result-document`*;
  - Peut permettre de stocker des parties du document de sortie (header, footer) au même titre que les variables.

XSLT/ Instruction `call-template` (2/2). Exemple.

```
<xsl:template match="/">
    <xsl:call-template name="name1"/>
</xsl:template>

<xsl:template name="name1">
    <xsl:result-document href="#">
        <html>
            ...
        </html>
    </xsl:result-document>
</xsl:template>
```

12. XSLT. Transformation vers LaTeX

- Même en-tête XSL que pour la transformation vers HTML.
- Transformation directe de XML à LaTeX non-prise en charge par XSLT → **utiliser la méthode `text` dans le `<xsl:output/>`**.
- Conséquence → les commandes LaTeX ne sont pas reconnues comme telles et traitées comme du texte.

XSLT/ Méthode texte (2/2). Template LaTeX basique

```
\documentclass[]{book}
\usepackage[T1]{fontenc}
\usepackage[french]{babel}
\title{}
\author{}
\date{}

\begin{document}
\maketitle

%% votre texte %%

\end{document}
```

13. Python et XSLT

Python/ Chaîne de publication électronique

- Une chaîne reproductible + documentée + automatisable
- Charger plusieurs données d'entrée (*XML*, *CSV*, *JSON*, *YAML*, *SQL*)
- Manipuler les données (nettoyage, fusion, extraction)
- Produire plusieurs formats de données de sortie adaptés aux intervenants du projet
- Pipeline : charger $\rightarrow$ pré-traiter $\rightarrow$ transformer $\rightarrow$ post-traiter $\rightarrow$ publier

- Gestion des fichiers (*`Pathlib`*, *`os`*)
- Chargement des données / data parsing (*`pandas`*, *`json`*, *`lxml`*, *`yaml`*)
- Manipulation des données (variables, *`dict`*, *`list`*, *`dataframe`*, *`xml trees`*)
- Écriture des formats de sortie (chaîne multi-sorties : HTML + LaTeX + PDF)
- Publication des données (API).

- Sortir d'Oxygen (logiciel propriétaire)
- Construire une chaîne de publication électronique cohérente
- XSLT : personnalisation plus fine que d'autres librairies clefs en main (*Pandoc*) : besoins d'un travail scientifique ou patrimonial
- Python ne remplace pas XSLT mais le complète avec des traitements pré-transformation et post-transformation.

Python/ lxml

- Manipulation XML/HTML : parsing, validation, transformation
- Fonctionne avec un fichier ou une string
- Représente le document comme un arbre de nœuds (*`ElementTree`*)
- Implémentation de XPath 1 et XSLT 1

```python
from lxml import etree
# charger un fichier: deux solutions
tree = etree.parse("file.xml")
# ou
with open(file, 'r') as xml_file:
    tree = etree.parse(xml_file)
# charger une string
xml_string = "<div><p>Text</p></div>"
tree = etree.fromstring(xml_string)
```

```
from lxml import etree

namespaces = {'tei':
    'http://www.tei-c.org/ns/1.0'}

header = tree.xpath(
    '/tei:TEI/tei:teiHeader',
    namespaces=namespaces)

header[0].attrib['type'] = 'metadata'
```

- NB : la fonction *.xpath()* retourne une liste.
- NB : *attrib* est un *dict*.

Python/ Transformation XSLT

```
from lxml import etree

def xsl_transformation(xsl_file, xml_file):

    source = etree.parse(xml_file)

    xsl_doc = etree.parse(xsl_file)

    xsl_transformer = etree.XSLT(xsl_doc)

    output_doc = xsl_transformer(source)

    return output_doc
```

```
from lxml import etree

str(output_doc).write('output.html',
    encoding='utf-8'
)
```

- *`etree.XSLT()`* n'accepte que XSLT 1
- Solution de contournement :
  - XSLT produit plusieurs arbres XML dans un même fichier
  - Utiliser *`lxml`* pour parser chaque arbre
  - Utiliser python pour les enregistrer un à un