

# Cours COSME<sup>2</sup> - Partie 3

22 avril 2019

# Fusionner des chaînes de caractères : concat()

Syntaxe :

```
fn:concat(string,string,...)
```

# Le noeud contextuel

« [l'] endroit dans le document XML qui est considéré comme nœud courant pour appliquer l'expression XPath »

svgground.fr

## Noeud contextuel – exemple

`concat(@type, ' ', @n)` avec pour noeud contextuel la division « premier acte », donnera « act 1 ».

# Extraire des chaînes de caractères : substring()

Syntaxe :

```
fn:substring(string,start,len)  
fn:substring(string,start)
```

La fonction indique :

- la chaîne à transformer (*string*)
- le caractère de départ (*start*) **déterminé par sa position.**
- (option), la longueur de la sous-chaîne à conserver (*len* pour *length*).

## substring() - exemple

`substring('abcdefg', 3)` donne « cdefg », et `substring('abcdefg', 3, 3)` donne « cde »

# Changer la casse. upper|lower-case()

Syntaxe :

```
fn:lower-case(string)  
fn:upper-case(string)
```

# Exercice

Pour tous les speaker, transformer les noms des personnages de sorte à ce que seule la première lettre du nom soit en majuscule.

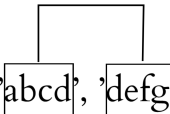


# Remplacer des caractères I : replace()

La fonction `replace()` permet de remplacer des chaînes de caractères qui respectent un motif (*pattern*) défini.

Syntaxe :

```
fn:replace(string,pattern,replace)
```



The diagram illustrates the replacement process. A horizontal line connects the top of the 'abcd' box to the top of the 'defg' box. Vertical lines then descend from each end of this horizontal line to the top of their respective boxes, visually representing the replacement of the first pattern with the second.


```
replace('abcdefg', 'abcd', 'defg')
```

## Remplacer des caractères II : translate()

La fonction `translate()` permet de remplacer des caractères **un à un**.  
Syntaxe :

```
fn:translate(string1,string2,string3)
```

`translate('abcdefg', 'abcd', 'defg')`



The diagram illustrates the mapping of characters from the first string 'abcdefg' to the second string 'abcd' and the third string 'defg'. The first four characters 'abcd' of the first string are mapped to the first four characters 'abcd' of the second string. The last three characters 'defg' of the first string are mapped to the last three characters 'defg' of the third string.

## translate() vs replace() - exemple

- `replace('abcdefg', 'dcba', 'defg')` donne `'abcdefg'` : le motif n'a pas été trouvé.
- `translate('abcdefg', 'dcba', 'defg')` donne `'gfedefg'`.

# Application

Pour un noeud donné, comment transformer un attribut de type identifiant (`@xml:id`) en attribut de type pointeur (`@target`), et vice-versa?

- quelle est la forme canonique de la valeur des attributs de type pointeur pour une référence interne au document?
- quelles fonctions pourraient permettre de passer d'un type d'attribut à un autre?

# Les variables

*If we use logic to control the flow of our stylesheets, we'll probably want to store temporary results along the way. In other words, we'll need to use variables. XSLT provides the <xsl:variable> element, which allows you to store a value and associate it with a name.*

TIDWELL, D. (2008). *XSLT : Mastering XML Transformations*. 2nd ed. O'Reilly Media, p.167.

On parle ici de **fonctions xsl** et non plus de fonctions XPath.

# Utilisation des variables

- Première méthode

```
1 <xsl:variable name="nom" select="valeur_a_capturer"/>
```

- Seconde méthode

```
1 <xsl:variable name="nom">  
2     regles a appliquer...  
3 </xsl:variable>
```

- On appelle les variables à l'aide du caractère dollar \$

# Illustration

- Imaginons une édition TEI avec un `teiHeader` qui comporte des informations précises sur les entités nommées du texte. Ces entités nommées sont donc présentées, définies, décrites dans le `teiHeader`, chacune a un identifiant unique `@xml:id`. Chaque entité du corps du texte renvoie vers sa définition à l'aide d'un pointeur `@target`.
- Pour chaque entité nommée du texte, nous voulons récupérer les informations correspondantes qui sont dans le `teiHeader`, et les mettre dans une `div` avec une classe donnée. Comment faire?

## Illustration – solution

- On peut, pour résoudre ce problème, créer une règle disant d'aller chercher, quand on rencontre une entité nommée, l'information correspondant dans les métadonnées du `teiHeader` (dans le noeud avec un `@xml:id` correspondant), et de l'imprimer. Nous allons voir comment faire dans l'exercice suivant, en utilisant les variables.



# Les variables – exercice 1

- Créer une notice pour chaque personnage listé dans la `<listPerson>`. Chaque notice devra contenir la description et les liens renvoyant vers la fichier wikipedia et wikidata. Cette notice doit apparaître au début de l'édition html.

# Solution possible I

```
1 <xsl:template match="person">
2     <div id="{@xml:id}">
3         <h3>
4             <xsl:value-of select="persName"/>
5         </h3>
6         <p>
7             <xsl:apply-templates select="note"/>
8             <br/>
9             <xsl:text>Reference:</xsl:text>
10            <a href="{descendant::ref[@type = 'wiki']/@target}"
11                >
12                <xsl:value-of select="bibl/ref[@type = 'wiki
13                    ']/@target"/>
14            </a>
15            <br/>
16        </p>
17    </div>
18 </xsl:template>
```

## Solution possible – suite

La règle plus haut ne fonctionne pas. Pourquoi? Pour faire en sorte que la notice apparaisse au début du document html; à l'endroit voulu, placer :

```
1 <div id="notices_personnages">
2     <h3>Notices: personnages</h3>
3     <xsl:apply-templates select="//person"/>
4 </div>
```

## Exercice II

Pour chaque nom de personnage qui apparaît dans le corps du texte **et qui est listé dans cette <listPerson>**, créer un lien vers sa notice en début du document

# Solution possible I

```
1 <xsl:template match="persName[@ref]">
2   <xsl:variable name="id_persName" select="translate(@ref, '#', '')"
3     <xsl:choose>
4       <xsl:when test="//person[@xml:id = $id_persName]">
5         <!--faire un lien vers cette notice-->
6         <a href="{@ref}">
7           <xsl:apply-templates/>
8         </a>
9       </xsl:when>
10      <xsl:otherwise>
11        <xsl:apply-templates/>
12      </xsl:otherwise>
13    </xsl:choose>
14  </xsl:template>
```

## Solution possible – suite

La règle plus haut ne fonctionne pas non plus. Pourquoi? Il faut modifier la règle qui gère les <l> :

```
1 <xsl:template match="l">
2     <div class="verse" id="{@xml:id}">
3         <xsl:apply-templates select="text() | figure/desc[@type =
4             'letter']/text() |c/text()"/>
5     </div>
6 </xsl:template>
```

Pour qu'elle s'applique aussi aux enfants <persName>.

# Solution possible - fin

La nouvelle règle sur <l> sera donc :

```
1 <xsl:template match="l">
2   <div class="verse" id="{@xml:id}">
3     <!--Maj: appliquer les regles sur toutes les balises mentionnees
        + sur les persName-->
4       <xsl:apply-templates select="text() | figure/desc[@type =
        'letter']/text() |c/text() | persName"/>
5     </div>
6 </xsl:template>
```

Pour qu'elle s'applique aussi aux enfants <persName>.

# Utiliser des variables pour remplacer des chaînes de caractères

Comment, au sein d'une même chaîne de caractère, remplacer plusieurs sous-chaînes? La méthode qui suit est utile, en particulier, pour passer d'un langage à un autre avec des règles d'échappement distinctes. Quelques exemples pour passer du XML à  $\text{\LaTeX}$  :

```
1 <xsl:template match="text()">
2     <xsl:variable name="remplacement1" select="replace(., '&', '\&')"/>
3     <xsl:variable name="remplacement2" select="replace($remplacement1
4         , '-', '--')"/>
5     ...
6     <xsl:variable name="remplacementN" select="replace(...)" />
7     <xsl:value-of select="$remplacementN" />
8 </xsl:template>
```



# Exercice

Créer une règle de remplacement récursif pour afficher de manière lisibles les URL, qui échappent dans le fichier XML les accents graves et aigus :

`https://fr.wikipedia.org/wiki/S%C3%A9n%C3%A8que`

`https://fr.wikipedia.org/wiki/Sénèque`

# Solution possible

```
1 <a href="{descendant::ref[@type = 'wiki']/@target}">
2   <!--Remplacement recursif des caracteres echappes dans l'URL-->
3   <xsl:variable name="remplacement1" select="replace(bibl/ref[@type
4     = 'wiki']/@target, '%C3%A9', 'é')"/>
5   <xsl:variable name="remplacement2" select="replace($remplacement1
6     , '%C3%A8', 'è')"/>
7   <xsl:value-of select="$remplacement2"/>
8 </a>
```

## Bonus I – Plusieurs documents d'entrée. fn :collection()

On va appeler ici la fonction XPath collection(). Sa syntaxe est la suivante :

fn:collection(string)

```
1 <xsl:value-of select="collection('...Hugo/romans/*.xml')//teiHeader//  
  title"/>
```

## Bonus I – Plusieurs documents de sortie.

On utilise ici un élément XSL appelé result-document :

```
1 <xsl:template match="...">
2     <xsl:result-document href="fiches_temoins/fiche_temoin_{
3         $id_temoin}.html">
4         <html>
5             ....
6         </html>
7     </xsl:result-document>
</xsl:template>
```