Cours Cosme2

La Publication avec XSLT

Simon Gabay

Lyon, 23 avril 2019

I. Les fonctions XPath

C'est quoi?

Les fonctions XPath:

1. se terminent par un `()` (et commencent parfois par préfixe `fn:`)
2. retournent
    - une valeur booléenne (*vrai* vs *faux*)
    - Un nombre
    - Une chaîne de caractères
    - Des nœuds
3. peuvent contenir des *arguments* (qui sont placés entre parenthèses).

Passons en revue quelques fonctions qui peuvent être utiles.

Les fonctions sans argument

Quelques exemples:

> fn:last()

W3C

> fn:position()

W3C

Lorsqu'on utilise une fonction:

1. on retire le préfixe `fn:`
2. on la place souvent comme prédicat (pour mémoire: `axe:nœud[predicat]`), ainsi `//l[last()]` renvoie le dernier `l` de chaque nœud qui en contiendrait un (et `(//l)[last()]` renvoie le dernier élément `l` de toute la pièce).
3. on peut ajouter des opérateurs booléens, ainsi `//l[position()=1]` renvoie le premier `l` d'une série, et `//l[last()-1]` l'avant-dernier.

Les fonctions à un argument

Un exemple:

> fn:count()

W3C

Pour pouvoir compter, il faut savoir ce que l'on compte: cette information est précisée au moyen d'un *argument*, qui est passé entre les `()` de la fonction. La syntaxe d'une fonction est décrite de la manière suivante:

> fn:count($arg)

Ainsi, `count(//l)` compte tous les `l` du document.

Les fonctions à deux arguments

> fn:contains()

W3C

> fn:starts-with()

W3C

> fn:ends-with()

W3C

> fn:substring-after()

W3C

Prenons l'exemple de `contains()` : il permet de contrôler que

1. une chaîne de caractères...
2. ... contient un/des caractère(s) précis.

Nous avons donc besoin de préciser:

1. Là où nous cherchons
2. Ce que nous cherchons

Pour cela, nous allons ajouter des *arguments* à la fonction XPath en précisant l'un (`$arg1`) et l'autre (`$arg2`), en suivant la syntaxe suivante:

```
contains($arg1, $arg2)
```

ATTENTION, XML/XPath est sensible à la casse!

Exercice I

Pour le nœud suivant:

```
<root>
  <a>1</a>
  <c>2</c>
  <a>3</a>
  <a>4</a>
  <a>5</a>
</root>
```

Que renvoient:

> count(a)

> //c[last()]

> //a[position()=2]

Quel est le résultat obtenu avec ces expressions XPath?

> contains('test', 's')

> starts-with('test', 's')

> substring-after('test', 's')

Pour l'arbre précédemment présenté, que renvoie cette requête?

> //a[contains(., '4')]

Et maintenant dans notre fichier `andromaque_c.xml` :

> //sp/@who[contains(., 'h')]

> //speaker[starts-with(., 'H')]

II. Les conditionnelles en XSLT

II.a `xsl:if`

L'élément `xsl:if` permet de réaliser un test conditionnel pour le contenu du fichier XML. Il obéit à la syntaxe suivante:

```
<xsl:template match="monElement">
  <xsl:if test="conditionQueJeDéfinis">
   ... Si la condition que j'ai posée est validée, alors
  </xsl:if>
</xsl:template>
```

W3C

`xsl:if` contient un attribut obligatoire `@test` qui contient une expression XPath. Cette expression XPath doit renvoyer une valeur booléenne : *true* ou *false*. Cette valeur est obtenue à partir de trois grands types de données:

1. Un nombre (est-il pair ou impair?)
2. Un fragment de l'arbre XML (est-ce un nœud parent?)
3. Une chaîne de caractères (contient-elle une lettre précise?)

Ce test est notamment fait à partir de fonctions XPath comme celles que nous avons vu auparavant.

Exercice II.a

Les vers de notre document sont numérotés à l'aide d'un `@n`

```
<l n="13" xml:id="v13">Combien dans cét exil ay-je <c ana=
<l n="14" xml:id="v14">Combien à vos malheurs ay-je donné
<l n="15" xml:id="v15">Craignant toûjours pour vous quelq
```

Vous pouvez donc

1. numéroter les lignes d'*Andromaque*, mais uniquement les dizaines (10, 20, 30, 40, 50).
2. numéroter les lignes qui sont un multiple de 5.

Corrigé `xsl_etape_2-1.xsl` .

II.b `xsl:choose`

L'élément `xsl:choose` ressemble énormément à `xsl:if`. Il permet de réaliser une série de tests conditionnels (`xsl:if` ne permettant d'en effectuer qu'un seul) et doit contenir *a minima* un `xsl:when`. Sa syntaxe minimale est donc la suivante:

```
<xsl:template match="monElement">
  <xsl:choose>
    <xsl:when test="conditionQueJeDéfinis">
      ... Si la condition que j'ai posée est validée, alo
    </xsl:when>
  </xsl:choose>
</xsl:template>
```

W3C

Contrairement à `xsl:if`, il est possible de proposer une liste de conditions alternatives:

```
<xsl:choose>
  <xsl:when test="premiereCondition">
    ... Si la première condition que j'ai posée est valid
  </xsl:when>
  <xsl:when test="secondeCondition">
    ... Si la seconde condition que j'ai posée est validé
  </xsl:when>
    <xsl:when test="troisiemeCondition">
    ... Si la troisième condition que j'ai posée est vali
  </xsl:when>
</xsl:choose>
```

Il est aussi possible d'ajouter un élément `xsl:otherwise`, optionnel, permettant de prévoir le cas où aucune des conditions prélablement définies n'est remplie – très utile pour éviter des oublis.

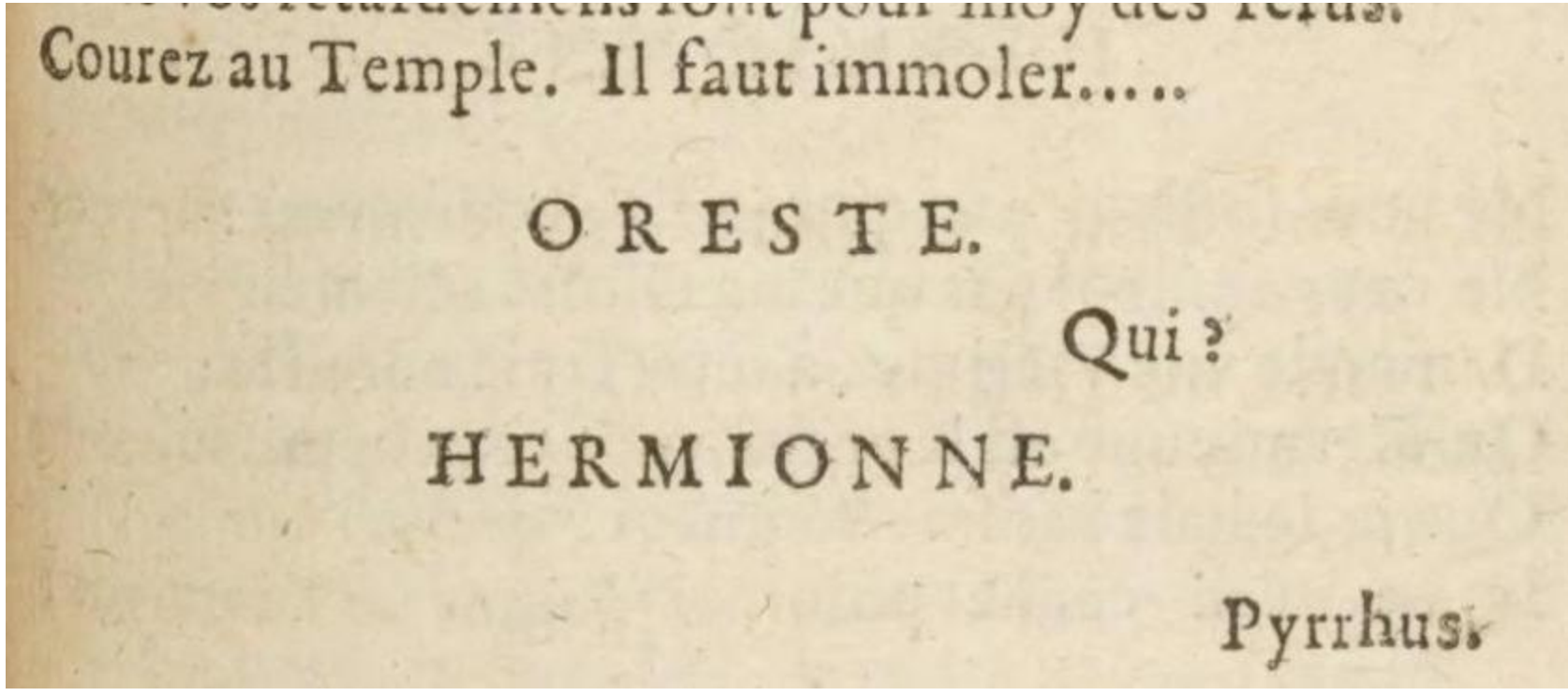
```
<xsl:choose>
  <xsl:when test="premiereCondition">
    ... Si la première condition que j'ai posée est valid
  </xsl:when>
  <xsl:when test="secondeCondition">
    ... Si la seconde condition que j'ai posée est validé
  </xsl:when>
    <xsl:when test="troisiemeCondition">
    ... Si la troisième condition que j'ai posée est vali
  </xsl:when>
  <xsl:otherwise>
    ... Dans tous les autres cas, alors…
  </xsl:otherwise>
</xsl:choose>
```

Exercice II.b

En regardant notre fichier XML de pus près, on remarque que les antilabes (morcellement du vers sur plusieurs répliques) ont été identifiées avec l'attribut `@part` :

```
    <l n="1176" xml:id="v1176" part="I">Courez au Temple.
  </lg>
</sp>
<sp who="#oreste" xml:id="A4S3_7">
  <speaker rend="center">ORESTE.</speaker>
  <lg>
    <l n="1176" part="M">Qui?</l>
  </lg>
</sp>
<sp who="#hermione" xml:id="A4S3_8">
  <speaker rend="center">HERMIONNE.</speaker>
  <lg>
    <l n="1176" part="F"><persName ref="#pyrrhus">Pyrrhus
  </lg>
</sp>
```

Noter les antilabes est important pour l'analyse stylistique de l'œuvre, mais aussi pour sa mise en page:


Courez au Temple. Il faut immoler.....

ORESTE.

Qui ?

HERMIONNE.

Pyrrhus.

Notre css a prévu cette situation

```
.verse {
    margin-left: 2em;
}

.verseM {
    margin-left: 8em;
}

.verseF {
    margin-left: 8em;
}
```

Il s'agit donc d'attribuer la bonne classe (`verse` , `verseM` ou `verseF`) dans le fichier HTML en fonction de `l @part` dans le fichier XML.

Corrigé: `xsl_etape_2-2.xsl` .

III. L'art de se compliquer la vie – ou ajouter une note

L'élément `xsl:attribute` permet d'ajouter un attribut à un élément – un élément HTML si l'on transforme notre document XML en page web, ou un élément XML si l'on modifie notre fichier . Il obéit à la syntaxe suivante:

```
<xsl:template match="monElement">
  <xsl:attribute name="nomAttribut">
    Valeur de l'attribut
  </xsl:attribute>
</xsl:template>
```

W3C

L'élément `xsl:text` fonctionne de la même manière. Il obéit à la syntaxe suivante:

```
<xsl:text>Ici je mets ce que je veux</xsl:text>
```

W3C

Ainsi, le code suivant

```
<xsl:template match="monElement">
  <span class="uneClasse">
    Mon contenu
  </span>
</xsl:template>
```

est équivalent à

```
<xsl:template match="monElement">
  <span>
    <xsl:attribute name="class">
      uneClasse
  </xsl:attribute>
  Mon contenu
</xsl:template>
```

qui est équivalent à

```
<xsl:template match="monElement">
  <span>
    <xsl:attribute name="class">
      <xsl:text>uneClasse</xsl:text>
  </xsl:attribute>
  Mon contenu
</xsl:template>
```

Alors pourquoi les utiliser? Ils offrent une plus grande marge de manœuvre quant à leur contenu. Ainsi:

1. `xsl:text` permet de mieux controler les espaces et les retours à la ligne du texte ajouté.
2. `xsl:attribute` permet de récupérer comme valeur d'attribut des données via `xsl:value-of` par exemple.

Exercice III

Tranformez les notes afin de les afficher dans une info-bulle, et que les liens vers gallica s'ouvrent dans un ouvel onglet.

1. Pour l'ouverture dans un nouvel onglet... cherchez sur internet! (indice: ça se passe dans `@target`). Trouvez deux manières différentes d'encoder les attributs.
2. Afin d'afficher le texte dans une info-bulle, vous pouvez utiliser `tooltip`, qui fonctionne de la manière suivante:

```
 <div class="tooltip">Appel de note
  <span class="tooltiptext">Contenu de la note</span>
</div>
```

Corrigé: `xsl_etape_2-final.xsl`.