

# eXtensible Stylesheet Language



## XSLT : principes et exemple

## Fonctionnement :

- XPath dans XSLT
- Règles par défaut
- Pattern matching
- Activation des règles : priorité, précédence, sélection

## Pilotage de XSLT :

- Production de l'arbre de sortie
- Instructions de contrôle
- Typologies des structures, transformations régies par les données
- Passage de paramètres
- Variables, result tree fragments
- Indexation, méthode de Muench
- Éléments de haut niveau
- Traitement des blancs
- Espaces de nommage
- Numérotation
- Contrôle de la sérialisation de l'arbre de sortie

## Production HTML

- génération de liens

## Fonctionnalités avancées

- Intégration de multiples sources de données
- Structures de données internes

## Limitations de XSLT

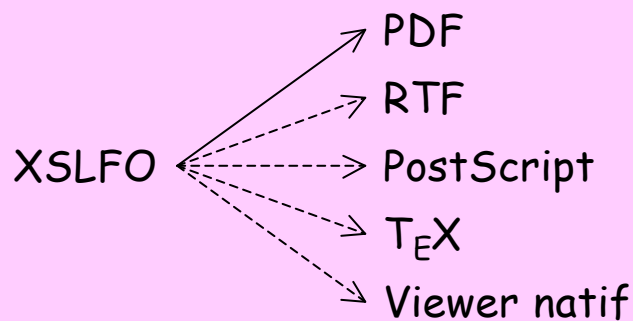
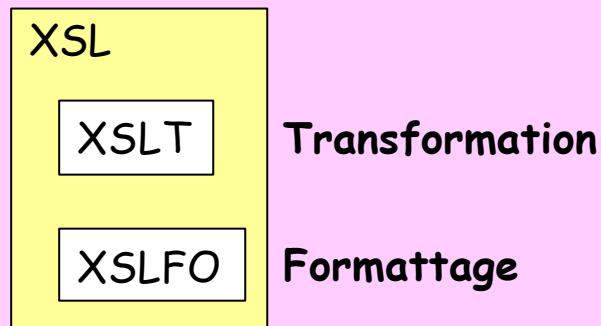
## Association d'une feuille de style XSLT

## Programmation XSLT

## XSLT

## XSLFO

- blocs
- gabarits des pages
- flots de contenus

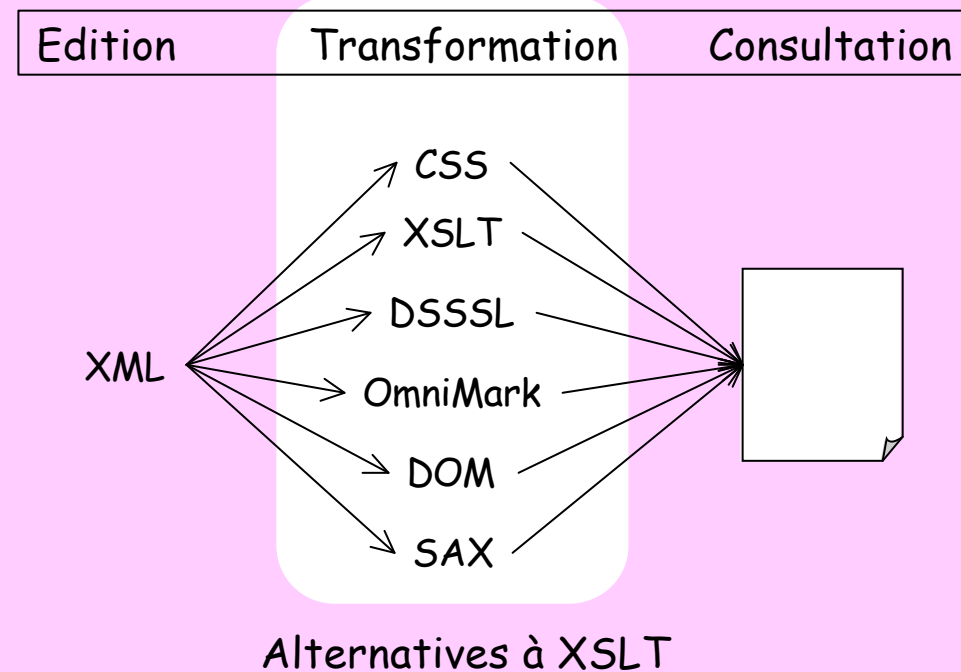


## Processeurs XSLT

Xalan, Saxon  
MSXML2.4.x

## Convertisseurs FO

FOP (Open Source : <http://xml.apache.org/fop>) XSLFO vers PDF  
 RenderX's XEP (Commercial : <http://www.renderx.com>) XSLFO vers PDF et PostScript  
 jfor (Open Source : <http://sourceforge.net/projects/jfor>) XSLFO vers RTF



<http://www.w3.org/TR/xslt>

## XSL Transformation

XSLT utilise une grammaire XML  $\longrightarrow$  Un document XSLT est un document XML

XSLT est un langage déclaratif (peu procédural) basé sur des règles

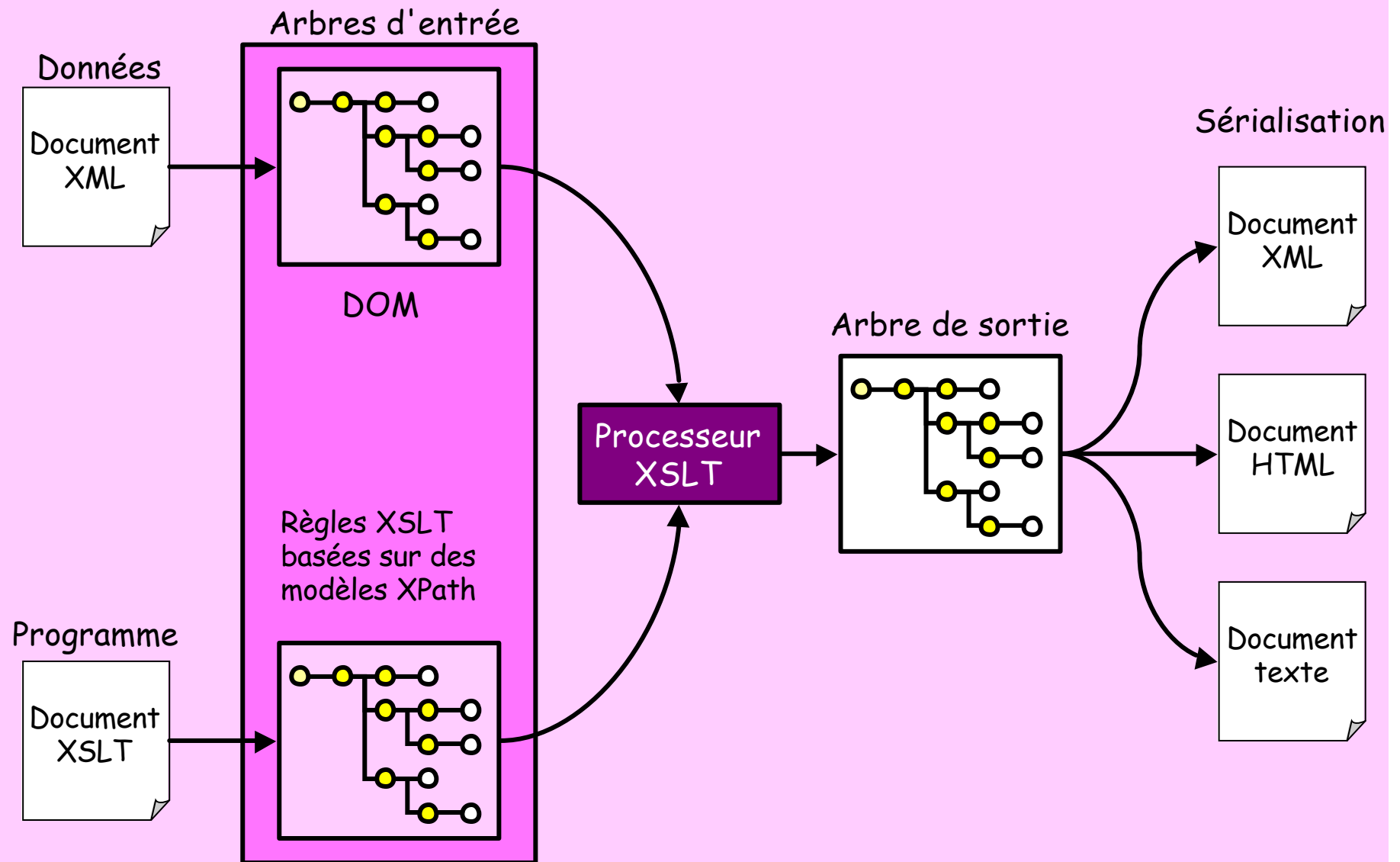
Langage de programmation :

- dédié au traitement des documents XML (transformations de structure)
- basé sur des définitions de règles à appliquer au document source

Chaque règle produit un résultat

XSLT :

- permet de manipuler des données faiblement typées (booléens, chaînes, nombres, ensembles de nœuds, fragments de résultat)
- fournit un ensemble d'opérations de déclaration et de soumission de règles comme `<xsl:template>`, `<xsl:apply-templates>`, de tri `<xsl:sort>`, de formatage du résultat `<xsl:output>`...
- offre des instructions de contrôle du déroulement du programme (`<xsl:if>`, `<xsl:for-each>`, `<xsl:choose>`...)
- utilise intensivement XPath



Certains processeurs peuvent traiter en entrée des arbres DOM ou des événements SAX

Certains processeurs peuvent directement délivrer un arbre DOM ou des événements SAX à une application

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours>
  <Titre>Cours XML</Titre>
  <Auteur>    <Nom>Poulard</Nom>
              <Prénom>Philippe</Prénom>
</Auteur>
  <Description>Ce cours aborde les concepts de base mis en œuvre dans XML.
</Description>
</Cours>
```



```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="Cours/Titre"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="Cours/Titre"/></h1>
        <xsl:apply-templates select="Cours/Auteur"/>
        <xsl:apply-templates select="Cours/Description"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="Auteur">
    <p align="right">par <xsl:value-of select="Nom"/>
    &#160;<xsl:value-of select="Prénom"/></p>
  </xsl:template>
  <xsl:template match="Description">
    <p><xsl:value-of select="."/></p>
  </xsl:template>
</xsl:stylesheet>
```

Cours XML - Web navig...

# Cours XML

par Poulard Philippe

Ce cours aborde les concepts de base  
mis en œuvre dans XML.

```
<html xmlns="http://www.w3.org
/TR/xhtml1/strict">
  <head>
    <title>Cours XML</title>
  </head>
  <body>
    <h1>Cours XML</h1>
    <p align="right">par
    Poulard&nbsp;Philippe</p>
    <p>Ce cours aborde les
    concepts de base mis en
    œuvre dans XML. </p>
  </body>
</html>
```



S'utilise dans les attributs : remplacer les < par &lt;

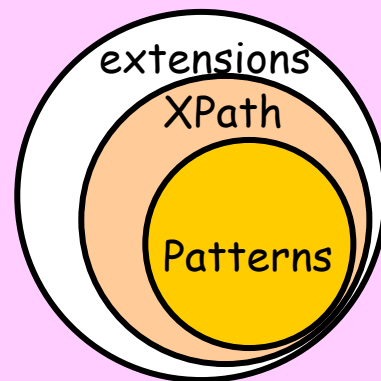
```
<xsl:if test="position() < 5">
```

Non

```
<xsl:if test="position() &lt; 5">
```

Oui

- On utilise un sous-ensemble de XPath : les patterns
- On y ajoute quelques fonctionnalités propres à XSLT



#### Fonctions XSLT :

- `current()`
- `document()`
- `element-available()`
- `format-number()`
- `function-available()`
- `generate-id()`
- `key()`
- `system-property()`
- `unparsed-entity-uri()`

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<Cours niveau="débutant">
  <!-- le meilleur cours sur XML ;o) -->
  <Titre>Cours XML</Titre>
  <Auteur>    <Nom>Poulard</Nom>
              <Prénom>Philippe</Prénom>

  </Auteur>
  <?acheter du pain pour ce soir?>
  <Description>Ce cours aborde les concepts de base mis en œuvre dans XML.
  </Description>
</Cours>
```



```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
```



Cours XML - Web navig...

Cours XML Poulard Philippe Ce cours aborde les concepts de base mis en œuvre dans XML.

Une feuille de style XSLT minimaliste permet de produire un résultat.  
Ce résultat est obtenu grâce à des règles par défaut.

```
<?xml version="1.0"?>
Cours XML Poulard Philippe Ce
cours aborde les concepts de
base mis en œuvre dans XML.
```

## Règles par défaut

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

<xsl:template match="text() | @"*>
  <xsl:value-of select="."/>
</xsl:template>

<xsl:template match="processing-instruction() | comment()"/>
```



```

<xsl:template match="prénom">
  .../...
</xsl:template>
<xsl:template match="nom">
  .../...
</xsl:template>
<xsl:template match="auteur/nom">
  .../...
</xsl:template>
<xsl:template match="auteur/nom[@rôle='principal']">
  .../...
</xsl:template>
<xsl:template match="rédacteur/nom">
  .../...
</xsl:template>
<xsl:template match="//chapitre/nom">
  .../...
</xsl:template>

```

```
<xsl:apply-templates select="nom"/>
```

```

<?xml version="1.0" encoding="ISO-8859-1"?>
<livre><titre>La philosophie chinoise</titre>
  <auteur rôle="principal">
    <nom>Doe</nom><prénom>John</prénom>
  </auteur>
  <auteur>
    <nom>Doe</nom><prénom>Jane</prénom>
  </auteur>
  <chapitre><titre>Introduction</titre>
    <p>Le grand <nom>Lao Tseu</nom> a dit : <cite>il faut trouver la voie</cite>
  </p>
  .../...

```

## Appel de règle

```
<xsl:call-templates name="nom-de-règle" />
```

## Définition de règle

```
<xsl:template name="nom-de-règle">
  .../...
</xsl:template>
```

Le  
contexte  
ne  
change pas

```
<xsl:apply-templates select="expression" />
```

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

Le  
contexte  
devient  
celui de  
l'expression

```
<xsl:apply-templates select="expression"
  mode="catégorie" />
```

```
<xsl:template match="pattern"
  mode="catégorie">
  .../...
</xsl:template>
```



**Expression** : désigne un ensemble de nœuds avec XPath

**Pattern** : condition sur les nœuds pour déclencher la règle

**Pattern** : "sous-ensemble" de XPath

- retourne toujours un ensemble de nœud (node-set)
- les étapes de localisation ne doivent concerner que les axes `child` ou `attribute` ou `descendent` en utilisant la notation `//`



**Contexte** :  
Donné par la fonction  
`current()`

Quand plusieurs règles sont candidates, un algorithme de sélection permet de choisir celle qui sera appliquée, selon :

- le mode
- la priorité
- la préséance

Priorité explicite :

```
<xsl:template match="pattern"
  priority="x">
  .../...
</xsl:template>
```

Priorité par défaut (selon le pattern)

Patterns à une seule étape de localisation :

foo-élément	0
@bar-attribut	0
ns-blah:foo-élément	0
ns-blah:@bar-attribut	0
processing-instruction("foo")	0
ns-blah:*	-0,25
@ns-blah:*	-0,25
*	-0,5
@*	-0,5
text()	-0,5
node()	-0,5
comment()	-0,5
processing-instruction()	-0,5

Autres patterns :

0,5

plus une règle est  
spécifique, plus sa  
priorité est élevée

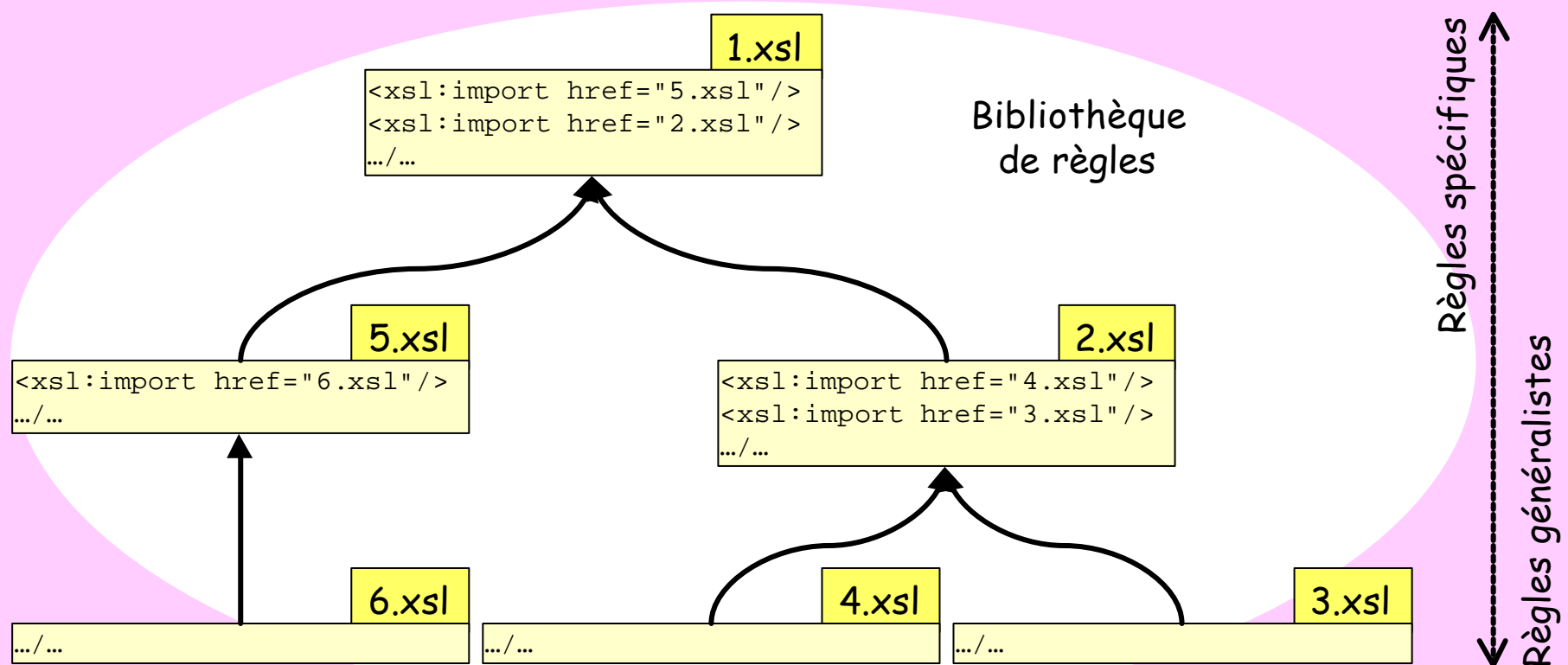
Plus la valeur est grande,  
plus la priorité est élevée

Préséance : s'applique aux règles importées par `xsl:import`

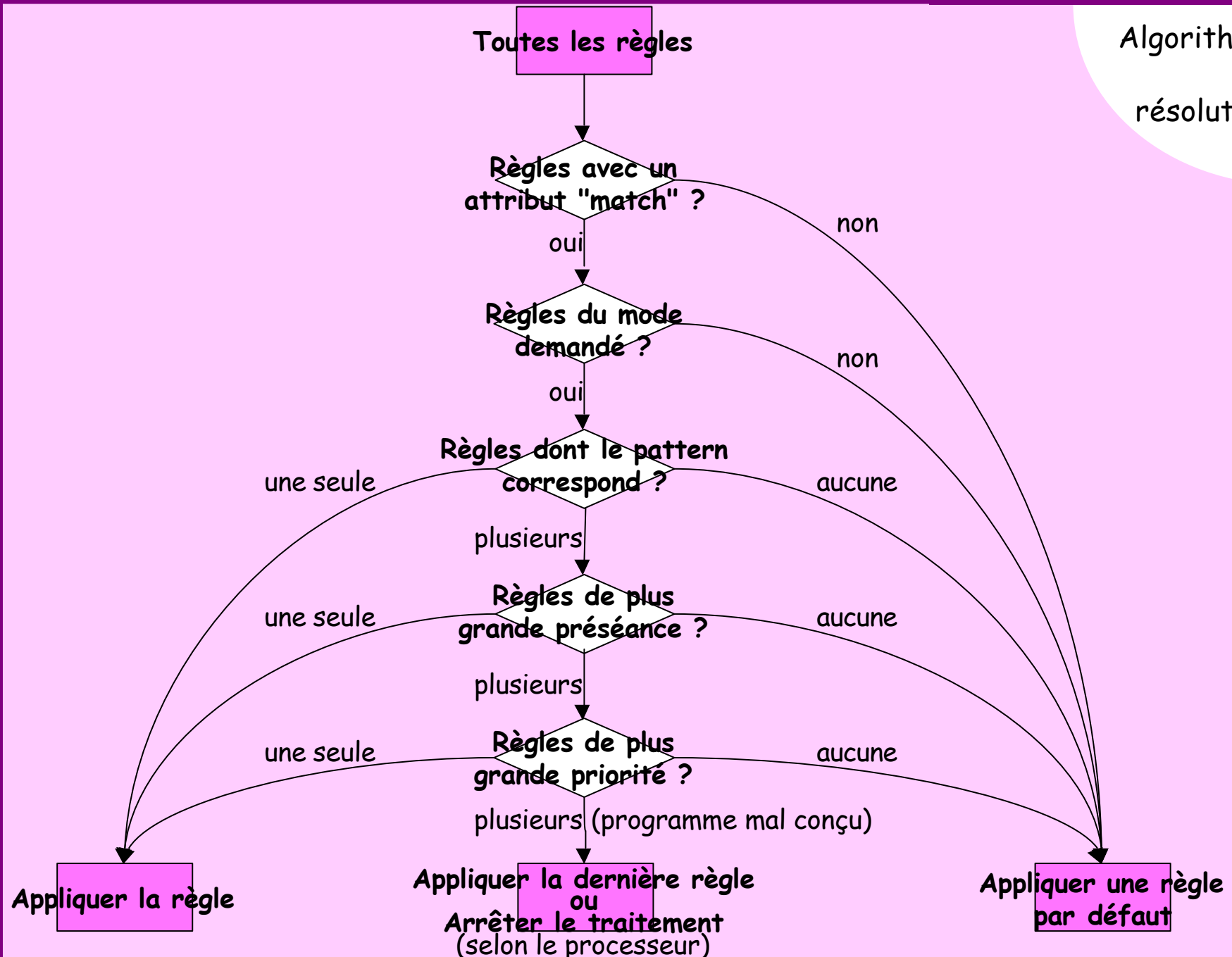
Règle de préséance (comme avec les CSS et la directive `@import`) :

- la dernière règle importée à la primeur sur les autres
- les règles d'un document importateur prévalent sur celles d'un document importé (surcharge des règles)

XSL



Algorithme  
de  
résolution



## Production d'éléments

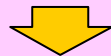
Peut se faire :

- avec des éléments littéraux
- avec des instructions `xsl:element`
- par copie avec les instructions `xsl:copy` et `xsl:copy-of`

```
<xsl:template match="bar">
  <foo>
    .../...
  </foo>
</xsl:template>
```

ou

```
<xsl:template match="bar">
  <xsl:element name="foo">
    .../...
  </xsl:element>
</xsl:template>
```



Arbre de sortie :

```
<foo>
  .../...
</foo>
```

```
<xsl:text>
  <xsl:element>
    <xsl:attribute>
<xsl:processing-instruction>
  <xsl:comment>

  <xsl:attribute-set>
    <xsl:copy>
  <xsl:copy-of>
  <xsl:value-of>
  <xsl:number>
```

Copie du nœud courant, et de sa valeur

```
<xsl:template match="bar">
  <xsl:copy>
    .../...
  </xsl:copy>
</xsl:template>
```

Ne copie pas les attributs

Copie d'un nœud en profondeur, avec tous les sous-nœuds, et leur valeur

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

## Calcul de valeurs

Avec `xsl:value-of`

```
<xsl:template match="livres">
  <table border="1">
    <xsl:apply-templates/>
  </table>
</xsl:template>

<xsl:template match="livre">
  <tr>
    <td><xsl:value-of select="titre"/></td>
    <td><xsl:value-of select="auteur/nom"/></td>
    <td><xsl:value-of select="auteur/prénom"/></td>
  </tr>
</xsl:template>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<livres>
  <livre>
    <auteur>
      <nom>Victor</nom>
      <prénom>Hugo</prénom>
    </auteur>
    <titre>Les misérables</titre>
  </livre>
  <livre>
    <titre>Germinal</titre>
    <auteur>
      <nom>Emile</nom>
      <prénom>Zola</prénom>
    </auteur>
  </livre>
</livres>
```

```
<xsl:text>
<xsl:element>
<xsl:attribute>
<xsl:processing-instruction>
<xsl:comment>

<xsl:attribute-set>
  <xsl:copy>
  <xsl:copy-of>
  <xsl:value-of>
  <xsl:number>
```

```
<table>
  <tr>
    <td>Les misérables</td>
    <td>Victor</td>
    <td>Hugo</td>
  </tr>
  <tr>
    <td>Germinal</td>
    <td>Emile</td>
    <td>Zola</td>
  </tr>
</table>
```

### Web navigator

Les misérables	Victor	Hugo
Germinal	Emile	Zola

## Production d'attributs

Peut se faire :

- avec des éléments littéraux
- avec des instructions `xsl:attribute`
- avec des jeux d'attributs prédéfinis

```
<xsl:template match="bar">
  <foo>
    <xsl:attribute name="param">value</xsl:attribute>
    .../...
  </foo>
</xsl:template>
```

```
<xsl:template match="bar">
  <xsl:element name="foo" use-attribute-set="foo-attributes">
    .../...
  </xsl:element>
</xsl:template>
```

```
<xsl:template match="bar">
  <foo param="value">
    .../...
  </foo>
</xsl:template>
```

```
<xsl:text>
<xsl:element>
<xsl:attribute>
<xsl:processing-instruction>
<xsl:comment>
<xsl:attribute-set>
  <xsl:copy>
  <xsl:copy-of>
  <xsl:value-of>
  <xsl:number>
```

Contenus  
d'attributs  
calculés

```
<img>
  <xsl:attribute name="src">
    <xsl:text>img/</xsl:text>
    <xsl:value-of select="image/@name" />
  </xsl:attribute>
</img>
```

```
<image name="photo.jpg">
```

## Attribute Value Template

```

```



Forme abrégée

```

```



## Production d'attributs

### Ensembles d'attributs

Avec `xsl:attribute-set`

```
<xsl:text>
<xsl:element>
<xsl:attribute>
<xsl:processing-instruction>
<xsl:comment>
```

```
<xsl:attribute-set>
  <xsl:copy>
  <xsl:copy-of>
  <xsl:value-of>
  <xsl:number>
```

```
<xsl:template match="chapter/title">
  <fo:block xsl:use-attribute-set="title-style">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:template match="section/title">
  <fo:block xsl:use-attribute-set="title-style">
    <xsl:apply-templates/>
  </fo:block>
</xsl:template>
<xsl:attribute-set name="title-style">
  <xsl:attribute name="font-size">12pt</xsl:attribute>
  <xsl:attribute name="font-weight">bold</xsl:attribute>
  <xsl:attribute name="text-align">center</xsl:attribute>
</xsl:attribute-set>
```

## Production de texte

Avec `xsl:text`

Permet de contrôler avec précision les contenus des éléments de sortie

```
<xsl:template match="/">
  (c) Philippe Poulard
</xsl:template>
```

```
<xsl:template match="/">
  <xsl:text>(c) Philippe Poulard</xsl:text>
</xsl:template>
```



```
(c) Philippe Poulard
```

```
<xsl:text>
  <xsl:element>
    <xsl:attribute>
      <xsl:processing-instruction>
        <xsl:comment>

      <xsl:attribute-set>
        <xsl:copy>
          <xsl:copy-of>
            <xsl:value-of>
              <xsl:number>
```

A éviter :

```
<xsl:template match="/">
  <xsl:text disable-output-escaping="yes">&lt;malformé&gt;</xsl:text>
</xsl:template>
```



```
<malformé>
```

## Production de commentaire

Avec `xsl:comment`

```
<xsl:template match="foo">
  <!-- début du commentaire -->
  <xsl:comment> commentaire produit en sortie </xsl:comment>
  <!-- fin du commentaire -->
</xsl:template>
```



```
<!-- commentaire produit en sortie -->
```

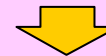
```
<xsl:text>
<xsl:element>
<xsl:attribute>
<xsl:processing-instruction>
<xsl:comment>

<xsl:attribute-set>
  <xsl:copy>
    <xsl:copy-of>
      <xsl:value-of>
        <xsl:number>
```

## Production d'instructions de traitement

Avec `xsl:processing-instruction`

```
<xsl:template match="/">
  <xsl:processing-instruction name="xml-stylesheet">
    <xsl:text>type="text/xml" href="style.xml"</xsl:text>
  </xsl:processing-instruction>
  <xsl:apply-templates />
</xsl:template>
```



```
<?xml-stylesheet type="text/xml" href="style.xml"?>
```

```
<xsl:text>
<xsl:element>
<xsl:attribute>
<xsl:processing-instruction>
<xsl:comment>

<xsl:attribute-set>
  <xsl:copy>
  <xsl:copy-of>
  <xsl:value-of>
  <xsl:number>
```

## Exécution conditionnelle

Avec `xsl:if`

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:apply-templates select="équipes/équipe"/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="équipe">
    <tr>
      <xsl:if test="position() mod 2 = 0">
        <xsl:attribute name="bgcolor">yellow</xsl:attribute>
      </xsl:if>
      <td><xsl:apply-templates /></td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

Que se passe-t-il si on remplace  
par `<xsl:apply-templates />` ?

```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes>
  <équipe cn="fr">OM</équipe>
  <équipe cn="it">Milan AC</équipe>
  <équipe cn="fr">PSG</équipe>
  <équipe cn="en">Arsenal</équipe>
  <équipe cn="it">Juventus</équipe>
</équipes>
```

```
<xsl:if>
  <xsl:choose>
    <xsl:for-each>
```

```
<xsl:apply-templates>
  <xsl:call-template>
  <xsl:apply-imports>
    <xsl:message>
  <xsl:fallback>
```

Web nav...

OM
Milan AC
PSG
Arsenal
Juventus

## Alternative

Avec xsl:choose

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:apply-templates/>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="équipe">
    <tr>
      <td>
        <xsl:apply-templates/>
        (<xsl:choose>
          <xsl:when test="@cn='fr'">France</xsl:when>
          <xsl:when test="@cn='en'">Angleterre</xsl:when>
          <xsl:when test="@cn='it'">Italie</xsl:when>
          <xsl:otherwise>??</xsl:otherwise>
        </xsl:choose>)
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes>
  <équipe cn="fr">OM</équipe>
  <équipe cn="it">Milan AC</équipe>
  <équipe cn="fr">PSG</équipe>
  <équipe cn="en">Arsenal</équipe>
  <équipe cn="it">Juventus</équipe>
</équipes>
```



Web navigator

OM (France)
Milan AC (Italie)
PSG (France)
Arsenal (Angleterre)
Juventus (Italie)

```
<xsl:if>
<xsl:choose>
<xsl:for-each>

<xsl:apply-templates>
  <xsl:call-template>
  <xsl:apply-imports>
    <xsl:message>
  <xsl:fallback>
```

## Boucle

Avec `xsl:for-each`

```

        <xsl:if>
        <xsl:choose>
        <xsl:for-each>

```

```

<xsl:apply-templates>
  <xsl:call-template>
  <xsl:apply-imports>
  <xsl:message>
  <xsl:fallback>

```

```

<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:for-each select="équipes/équipe">
            <tr>
              <td>
                <xsl:apply-templates/>
                (<xsl:choose>
                  <xsl:when test="@cn='fr'">France</xsl:when>
                  <xsl:when test="@cn='en'">Angleterre</xsl:when>
                  <xsl:when test="@cn='it'">Italie</xsl:when>
                  <xsl:otherwise>??</xsl:otherwise>
                </xsl:choose>)
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>

```

Tri

xsl:apply-templates  
xsl:sort

ou

xsl:for-each  
xsl:sort

```
<xsl:apply-templates select="équipes/équipe">
  <xsl:sort select="@cn" order="descending"/>
  <xsl:sort select="."/>
</xsl:apply-templates>
```

```
<xsl:if>
  <xsl:choose>
    <xsl:for-each>
      <xsl:apply-templates>
        <xsl:call-template>
          <xsl:apply-imports>
            <xsl:message>
              <xsl:fallback>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:for-each select="équipes/équipe">
            <xsl:sort select="@cn" order="descending"/>
            <xsl:sort select="."/>
            <tr>
              <td>
                <xsl:apply-templates/>
                (<xsl:choose>
                  <xsl:when test="@cn='fr'">France</xsl:when>
                  <xsl:when test="@cn='en'">Angleterre</xsl:when>
                  <xsl:when test="@cn='it'">Italie</xsl:when>
                  <xsl:otherwise>??</xsl:otherwise>
                </xsl:choose>)
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```



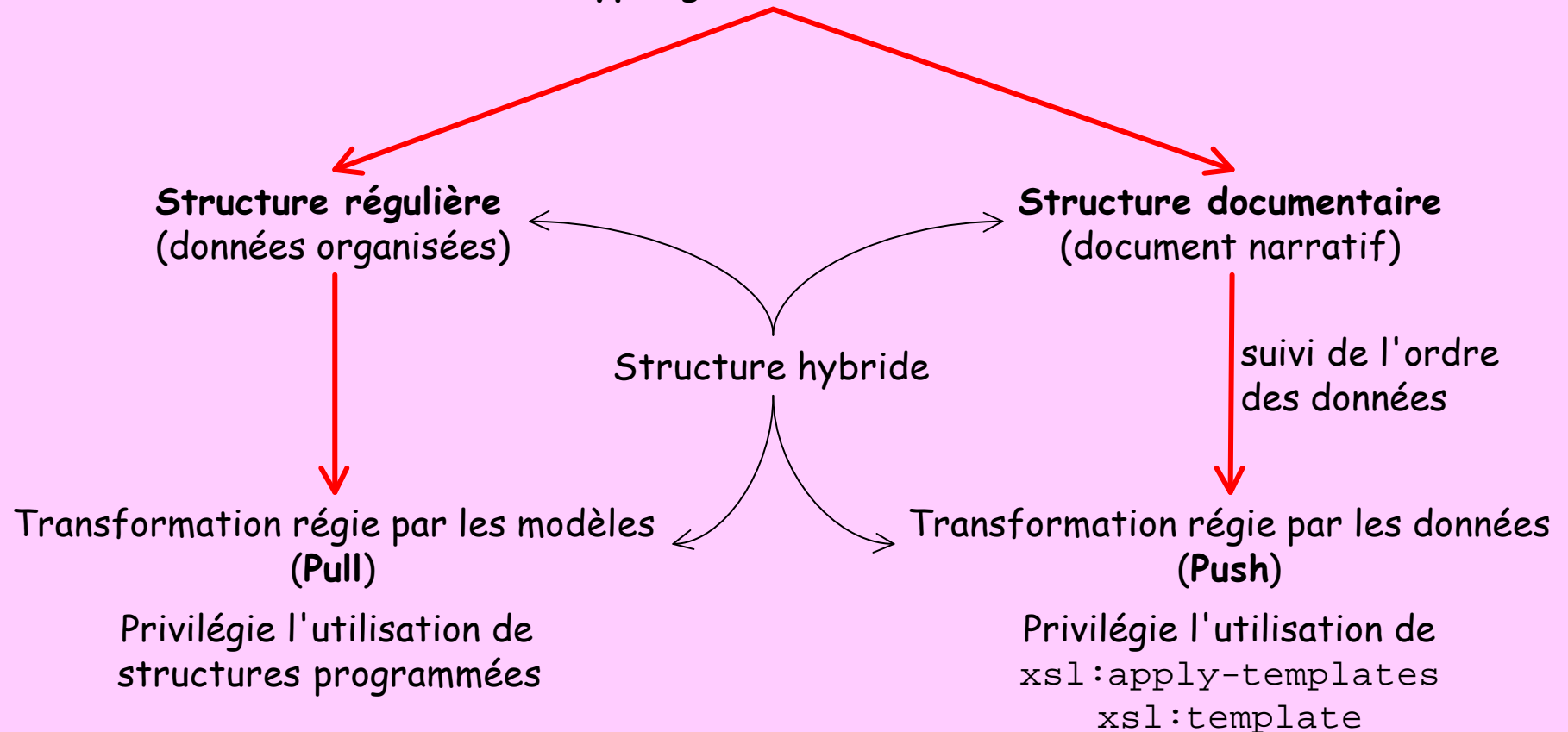
```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes>
  <équipe cn="fr">OM</équipe>
  <équipe cn="it">Milan AC</équipe>
  <équipe cn="fr">PSG</équipe>
  <équipe cn="en">Arsenal</équipe>
  <équipe cn="it">Juventus</équipe>
</équipes>
```



```
<commande id="kh34jl">
  <article id="4l34m" prix="120" qté="5">
  <article id="df40t" prix="210" qté="1">
  <article id="lnk23" prix="330" qté="4">
</commande>
```

```
<vers>Maître <b>corbeau</b> sur un
arbre perché tenait dans son bec un
fromage</vers>
<vers>Maître <i>renard</i> par l'odeur
alléché .../</vers>
```

## Typologie de structure



La plupart des documents sont des structures hybrides. Les structures purement documentaires ou régulières se trouvent dans des cas particuliers

## Structures

- contenant des ensembles d'éléments hétérogènes
- avec des éléments optionnels
- récursives
- dont l'ordre est important

Exemple : que se passe-t-il si on triait un poème dans l'ordre alphabétique de ses vers ?

A l'opposé, une structure régulière, telle qu'une commande, peut être :

- non triée,
- triée par date
- triée par nom d'item
- triée par prix
- ...

## Procédures

xsl:call-template  
 xsl:with-param  
 xsl:param

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:for-each select="équipes/équipe">
            <tr>
              <td>
                <xsl:apply-templates/>
                (<xsl:call-template name="label-nation">
                  <xsl:with-param name="code-nation" select="@cn"/>
                  </xsl:call-template>)
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
  <xsl:template name="label-nation">
    <xsl:param name="code-nation"/>
    <xsl:choose>
      <xsl:when test="$code-nation='fr'">France</xsl:when>
      <xsl:when test="$code-nation='en'">Angleterre</xsl:when>
      <xsl:when test="$code-nation='it'">Italie</xsl:when>
      <xsl:otherwise>??</xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

<xsl:if>  
 <xsl:choose>  
 <xsl:for-each>

<xsl:apply-templates>  
 <xsl:call-template>  
 <xsl:apply-imports>  
 <xsl:message>  
 <xsl:fallback>

Les template nommés fournissent une structure de programmation équivalente aux **procédures** des langages procéduraux. Ils acceptent le passage de paramètres

## Paramètres

xsl:param

Permet d'agir de l'extérieur

<xsl:variable>  
<xsl:param>

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="filtre-nation" select="'fr'"/>
  <xsl:template match="/">
    <html>
      <body>
        <h1>Equipes en
          <xsl:call-template name="label-nation">
            <xsl:with-param name="code-nation" select="$filtre-nation"/>
          </xsl:call-template >
        </h1>
        <ul>
          <xsl:for-each select="équipes/équipe[@cn=$filtre-nation]">
            <li><xsl:apply-templates/></li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
  <xsl:template name="label-nation">
    <xsl:param name="code-nation"/>
    <xsl:choose>
      <xsl:when test="$code-nation='fr'">France</xsl:when>
      <xsl:when test="$code-nation='en'">Angleterre</xsl:when>
      <xsl:when test="$code-nation='it'">Italie</xsl:when>
      <xsl:otherwise>??</xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

Valeur modifiable  
par programme externe

Le mode  
d'interaction  
dépend du  
processeur et du  
langage

Web navigator

**Equipes en France**

- OM
- PSG

`filtre-nation='fr'`

Web navigator

**Equipes en Italie**

- Juventus
- Milan AC

`filtre-nation='it'`

Web navigator

**Equipes en Angleterre**

- Arsenal

`filtre-nation='en'`

```
<xsl:variable>  
<xsl:param>
```

## Définition de variables

xsl:variable

On peut assigner à une variable :

- le résultat d'une expression XPath (boolean, number, string, node-set)
- du contenu (result-tree-fragment)  
(Représentation sérialisée d'un résultat)

Exemples :

```
<xsl:variable name="organisme" select="FIFA">
```

```
<xsl:variable name="age-du-capitaine" select="22">
```

```
<xsl:variable name="équipes-françaises" select="//équipes/équipe[@cn='fr']">
```

```
<xsl:variable name="libellé-nation">  
  <xsl:call-template name="label-nation">  
    <xsl:with-param name="code-nation" select="@cn"/>  
  </xsl:call-template>  
</xsl:variable>
```

```
<xsl:variable name="lieu">  
  Stade de France  
  <nation>fr</nation>  
</xsl:variable>
```

<xsl:variable>  
<xsl:param>

## Utilisation de variables

S'utilise dans les expressions XPath

```
<xsl:value-of select="$organisme" />
```

```
<xsl:for-each select="joueur[@age < $age-du-capitaine]">
```

```
<xsl:apply-templates select="$équipes-françaises/joueurs/joueur" />
```

```
<xsl:value-of select="$libellé-nation" />
```

```
<xsl:value-of select="$lieu" />
```

Portée :

- variables globales : définies par des éléments `xsl:variable` de premier niveau, fils de `xsl:stylesheet`.et  
⇒ Visibles dans tout le programme
- variables locales : définies dans les corps de règle.  
⇒ Visibles dans tous les nœuds following-sibling et descendant du nœud `xsl:variable`

On ne peut pas définir 2 variables ayant le même nom si leurs portées se chevauchent (shadowing) à moins que l'une d'elles soit une variable globale ou que les 2 soient globales sans la même préséance.

```
<xsl:variable name="i" select="1" />
<xsl:for-each select="équipe">
  <xsl:variable name="i" select="$i + 1" />
</xsl:for-each>
<p>Il y a <xsl:value-of select="$i" /> équipes.</p>
```



Provoque une erreur

## Result tree fragment

Ne peut pas s'utiliser comme un node-set

```
<xsl:variable name="lieu">
  Stade de France
  <nation>fr</nation>
</xsl:variable>
```

```
<xsl:variable>
  <xsl:param>
```

```
<xsl:apply-templates select="$lieu/nation"/>
```

Non

Provoque une erreur

Les processeurs XSLT utilisent souvent une extension non prévue dans le standard qui permette de convertir un result-tree-fragment en node-set.

```
<xsl:apply-templates select="node-set($lieu)/nation"/>
```

Son utilisation dépend du processeur

Xalan-Java

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xalan="http://xml.apache.org">
.../...
  <xsl:apply-templates select="xalan:node-set($lieu)/nation"/>
.../...
```

MSXML2.4.x

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:msxsl="urn:schemas-microsoft-com:xslt">
.../...
  <xsl:apply-templates select="msxsl:node-set($lieu)/nation"/>
.../...
```

## Message

`xsl:message` permet de renvoyer un message à la console, ou dans les logs, ou dans une boîte de dialogue...

```
<xsl:choose>
  <xsl:when test="$code-nation='fr'">France</xsl:when>
  <xsl:when test="$code-nation='en'">Angleterre</xsl:when>
  <xsl:when test="$code-nation='it'">Italie</xsl:when>
  <xsl:otherwise>
    <xsl:message terminate="yes">
      Code nation inconnu
    </xsl:message>
  </xsl:otherwise>
</xsl:choose>
```

```
      <xsl:if>
      <xsl:choose>
      <xsl:for-each>

<xsl:apply-templates>
  <xsl:call-template>
  <xsl:apply-imports>
    <xsl:message>
  <xsl:fallback>
```



## Erreurs

`xs:fallback` s'utilise dans un contexte où la cible de déploiement des programmes XSLT n'est pas connue du développeur

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <xsl:for-each select="//équipe/@cn">
      <xsl:if test="not(..following-sibling::*[@cn=current()])">
        <xsl:result-document format="html" href="equipe-{}.html">
          <xsl:fallback>
            <xsl:message terminate="yes">
              Impossible de créer un fichier.
              L'instruction <xsl:result-document> n'est pas implémentée
            </xsl:message>
          </xsl:fallback>
        <html>
          <body>
.../...
```

```
<xsl:if>
<xsl:choose>
<xsl:for-each>
```

```
<xsl:apply-templates>
<xsl:call-template>
<xsl:apply-imports>
<xsl:message>
<xsl:fallback>
```

## Indexation

L'indexation avec `xsl:key` permet d'augmenter les performances de certains traitements

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:param name="code-nation" select="'fr'"/>
  <xsl:key name="équipes-par-nation" match="équipe" use="@cn"/>
  <xsl:template match="équipes">
    <html>
      <body>
        <ul>
          <xsl:for-each select="key('équipes-par-nation',$code-nation)">
            <li><xsl:value-of select="équipe"/></li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

On peut retrouver les équipes d'une nation, mais on ne peut pas obtenir la liste des nations ® méthode de Muench

## Génération d'ID

Fonction `generate-id(node)`

Retourne un identifiant unique (dans le document) pour le nœud passé en argument, ou le premier nœud du node-set passé en argument, ou du nœud courant s'il n'y a pas d'argument

La méthode de Muench (Muenchian method, by Steve Muench) :

- permet de grouper des données
- permet de sélectionner des valeurs distinctes

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:key name="group" match="équipe" use="@cn"/>
  <xsl:template match="équipes">
    <html>
      <body>
        <ul>
          <xsl:for-each select="équipe[generate-id()=generate-id(key('groupe',@cn)[1])]">
            <li><xsl:value-of select="équipe"/></li>
          </xsl:for-each>
        </ul>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

ou

```
<xsl:for-each select="équipe[count(. |key('groupe',@cn)[1])=1]">
```

Dans un node-set, les nœuds sont uniques

Sous la racine `xsl:stylesheet`, on trouve :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
  <xsl:namespace-alias/>
  <xsl:output/>
  <xsl:preserve-space/>
  <xsl:strip-space/>
  <xsl:attribute-set/>
  <xsl:key/>
  <xsl:decimal-format/>
  <xsl:param/>
  <xsl:variable/>
  <xsl:import/>
  <xsl:include/>
  <xsl:template/>
</xsl:stylesheet>
```

} directives de traitement du processeur

} définitions d'ensembles de données

} directives d'importation

définitions de règles

`xsl:transform` et `xsl:stylesheet` sont synonymes

## Contrôle des éléments en entrée :

```
<équipe xml:space="preserve">
```

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"/>
  <xsl:preserve-space elements="équipe joueur"/>
  <xsl:strip-space elements="équipes joueurs"/>
</xsl:stylesheet>
```

## Contrôle des éléments en sortie :

```
<xsl:text></xsl:text>
```

```
<xsl:text disable-output-escaping="yes"></xsl:text>
```

```
<xsl:text>&#x9;</xsl:text>
```

```
<xsl:text>&#xA;</xsl:text>
```

```
<xsl:text>&#x20;</xsl:text>
```

```
<xsl:text>&#160;</xsl:text>
```

espace insécable

Normalisation des contenus d'éléments (les valeurs d'attributs sont toujours normalisées) : `normalize-space()`

Un nœud texte utilisé seul permet de s'assurer qu'aucun autre contenu ne sera produit :

```
<xsl:template match="foo">❗
  ® <xsl:text>ici</xsl:text>❗
</xsl:template>
```



production du texte "ici"  
sans ❗ ® avant ni ❗ après

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:apply-templates select="équipes/équipe"/>
        </table>
      </body>
    </html>
  </xsl:template>
  .../...
</xsl:stylesheet>
```

Ne sélectionne rien

Quand on ajoute un espace de nommage par défaut dans un document, les feuilles de style qui le traitent doivent aussi le déclarer...

...en utilisant obligatoirement un préfixe (sinon, les données ne sont pas adressables par XPath)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:foot="http://www.fifa.fr/championnat">
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:apply-templates select="foot:équipes/foot:équipe"/>
        </table>
      </body>
    </html>
  </xsl:template>
  .../...
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="iso-8859-1"?>
<équipes xmlns="http://www.fifa.fr/championnat">
  <équipe cn="fr">OM</équipe>
  <équipe cn="it">Milan AC</équipe>
  <équipe cn="fr">PSG</équipe>
  <équipe cn="en">Arsenal</équipe>
  <équipe cn="it">Juventus</équipe>
</équipes>
```

Tous les espaces de nommage déclarés sont reproduits en sortie, sauf :

- celui associé au processeur XSLT
- ceux exclus nominativement avec `exclude-result-prefixes`

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:foot="http://www.fifa.fr/championnat"
  exclude-result-prefixes="foot">
  <xsl:template match="/">
    <html>
      <body>
        <table border="1">
          <xsl:apply-templates select="foot:équipes/foot:équipe"/>
        </table>
      </body>
    </html>
  </xsl:template>
  .../...
</xsl:stylesheet>
```

Exclusion de plusieurs déclarations d'espaces de nommage :

```
exclude-result-prefixes="foot joueurs matchs"
```

Exclusion de la déclaration d'espace de nommage par défaut :

```
exclude-result-prefixes="#default"
```

Les alias utilisés dans la feuille de style peuvent être renommés dans le document produit :

```
<xsl:namespace-alias
  stylesheet-prefix="pàv"
  result-prefix="planche-à-voile"/>
```

## Formattage des nombres

```
<xsl:decimal-format/>
```

```
format-number()
```

} La représentation des nombres est variable selon les pays

```
<xsl:decimal-format name="european" decimal-separator=',' grouping-separator='.'/'>
<xsl:value-of select="format-number(24535.2, '###.###,00', 'european')"/>
```



24.535,20

## Numérotation

```
<h2>Sommaire</h2>
<p>
  <xsl:for-each select="section">
    <xsl:number level="multiple" format="1." count="section"/>
    <a href="#{generate-id()}">
      <xsl:value-of select="title" />
    </a>
  <xsl:for-each>
</p>
```

```
<xsl:number/>
```

Permet de numéroté les  
tables des matières et  
indexes

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<content>
  <section><title>Objectifs</title>
    <p>.../...</p>
  </section>
  <section><title>Prérequis</title>
    <p>.../...</p>
  </section>
  <section><title>Manipulat
    <section><title>Démonta
      <p>.../...</p>
    </section>
    <section><title>Nettoyage
      <p>.../...</p>
    </section>
  </section>
```

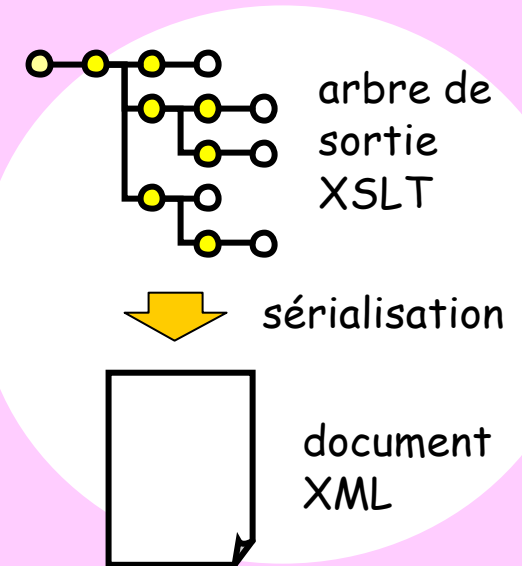
Web navig...

### Sommaire

- 1 [Objectifs](#)
- 2 [Prérequis](#)
- 3 [Manipulation](#)
  - 3.1 [Démontage](#)
  - 3.2 [Nettoyage](#)



```
<xsl:output
  method = "html"
  version = "4.0"
  encoding = "iso-8859-1"
  omit-xml-declaration = "yes"
  standalone = "yes"
  doctype-public = "-//W3C//DTD XHTML 1.0 Transitional//EN"
  doctype-system = "DTD/xhtml11-transitional.dtd"
  cdata-section-elements = "script style"
  indent = "yes"
  media-type = "text/html"
/>
```



```
<xsl:output method = "xml"/>
```

```
<xsl:output method = "text"/>
```

```
<xsl:output method = "html"/>
```

Il est recommander d'utiliser `xsl:text` pour produire le résultat

La mise au format HTML dépend des implémentations et des autres paramètres.

On peut obtenir la réécriture des caractères non ASCII par des appels d'entités (`&eacute;` au lieu de `é`)...

```
<xsl:output method = "maMethode"/>
```

A programmer soit-même...

Exemple : `<xsl:output method = "pdf"/>`

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="Cours/Titre"/></title>
      </head>
      <body>
        .../...
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>

```

### Syntaxe "simplifiée" de la feuille de style

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<html xsl:version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
  <head>
    <title><xsl:value-of select="Cours/Titre"/></title>
  </head>
  <body>
    .../...
  </body>
</html>

```

## Les entités binaires

Fonction `unparsed-entity-uri (node)`

img/flipper.jpg

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE zoo [
  <!ATTLIST dauphin photo ENTITY>
  <!NOTATION jpg SYSTEM "images/jpeg">
  <!ENTITY flipper SYSTEM "img/flipper.jpg" NDATA jpg>
]>
<zoo>
  <aquarium>
    <dauphin id="jhgtr13" photo="flipper">
      <nom>Flipper</nom>
      .../...
    </dauphin>
  </aquarium>
</zoo>
```

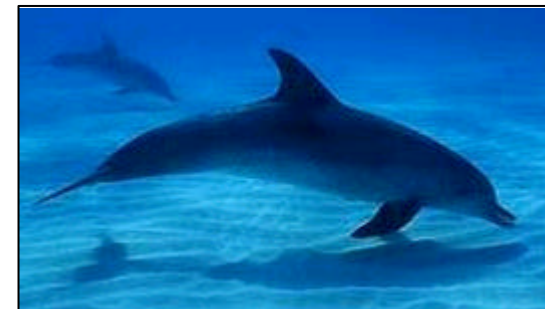


```
<xsl:template match="dauphin">
  <h2><xsl:value-of select="nom" /></h2>
  <p>
    <xsl:if test="@photo">
      
    </xsl:if>
  </p>
</xsl:template>
```

```
<h2>Flipper</h2>
<p>
  
</p>
```

Web navigator

Flipper



## Fonction

generate-id(node)

Construction d'une  
table des matières

```
<?xml version="1.0" encoding="ISO
.../...
<content>
  <section><title>Objectifs</title>
    <p>Ce document décrit comment
      monter et démonter un blah
    </p>
  </section>
  <section><title>Prérequis</title>
    <p>Le lecteur doit avoir une
      connaissance des blah appr
    </p>
    <p>En particulier, il doit ma
      les blah bleus et les blah
      de compétition.
    </p>
  </section>
  <section><title>Manipulation</t
    <info>Penser à vérifier les b
      avant toute manipulation.
    </info>
    <step>Enlever l'eau sale du b
    <step>Dévisser le capot infér
    <step>Cliquer sur OK.</step>
  </section>
.../...
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns="http://www.w3.org/TR/xhtml1/strict">
<xsl:template match="content">
  <html>
    <head>
      <title><xsl:value-of select="title"/></title>
    </head>
    <body>
      <h1><xsl:value-of select="title"/></h1>
      <h2>Sommaire</h2>
      <ul>
        <xsl:for-each select="//section">
          <li><a href="#{generate-id(title)}">
            <xsl:value-of select="title"/>
          </a></li>
        </xsl:for-each>
      </ul>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>
<xsl:template match="/content/title"/>
<xsl:template match="title">
  <h2><a name="{generate-id()}">
    <xsl:value-of select="."/></a>
  </xsl:template>
<xsl:template match="section">
  <xsl:apply-templates/>
</xsl:template>
<xsl:template match="p">
  <p><xsl:apply-templates/></p>
</xsl:template>
<xsl:template match="b">
```

Blah - Web...

**Les blah**

Sommaire :

- [Objectifs](#)
- [Prérequis](#)
- [Manipulation](#)

**Objectifs**

Ce document décrit comment monter et démonter un blah.

**Prérequis**

Le lecteur doit avoir une connaissance des

La fonction `document()` permet de réaliser des transformations simultanément sur plusieurs sources de données

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rapport>
  <analyse>
    <p>Cette année, la <b>croissance</b>
    est au rendez-vous,
    .../...
  </p>
</analyse>
</rapport>
```

ventes.xml

```
<?xml version="1.0"
  encoding="ISO-8859-1" ?>
<ventes>
  <charcuterie>23</charcuterie>
  <fromages>45</fromages>
  <primeurs>78</primeurs>
</ventes>
```

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  >
  <xsl:template match="/">
    <xsl:variable name="mes-ventes"
      select="document('ventes.xml')"/>
    <xsl:apply-templates select="$mes-ventes/ventes"/>
    <xsl:apply-templates select="rapport"/>
  </xsl:template>
</xsl:stylesheet>
```

Rapport - Web Explorer

- charcuterie : 15%
- fromages : 31%
- primeurs : 54%

Cette année, la **croissance** est  
au rendez-vous ...



La fonction `document()` utilisée sans argument permet d'obtenir la feuille de style elle-même, et donc d'y adresser des informations

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:date="*** Traitement des dates ***">
  <date:noms-mois>
    <date:mois court="jan">janvier</date:mois>
    <date:mois court="fév">février</date:mois>
    .../...
    <date:mois court="déc">décembre</date:mois>
  </date:noms-mois>

  <xsl:template name="date:nom-mois">
    <!--retourne le nom du mois en clair à partir du numéro de mois-->
    <xsl:param name="mois" select="0"/>
    <xsl:value-of select="document('')/*/*date:noms-mois/date:mois[$mois]"/>
  </xsl:template>
</xsl:stylesheet>
```

```
<xsl:variable name="année" select="substring-before($date, '-')"/>
<xsl:variable name="mois" select="substring-before(substring-after($date, '-'), '-')"/>
<xsl:variable name="jour" select="substring-after(substring-after($date, '-'), '-')"/>
<xsl:variable name="nom-mois">
  <xsl:call-template name="date:nom-mois">
    <xsl:with-param name="mois" select="number($mois)"/>
  </xsl:call-template>
</xsl:variable>
<xsl:value-of select="$jour" />
<xsl:value-of select="$nom-mois" />
<xsl:value-of select="$année" />
```

## Transformation à l'identique :

copie.xslt

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:template match="@* | node()">
    <xsl:copy>
        <xsl:apply-templates select="@* | node()" />
    </xsl:copy>
</xsl:template>
</xsl:stylesheet>
```

## Utile :

- pour changer l'encodage
- pour sérialiser un arbre DOM
- à l'import dans une feuille de style qui fait de l'adaptation de structure

## Exemple : conversion des attributs en éléments

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:import href="copie.xslt"/>
<xsl:output method="xml" version="1.0" indent="yes" encoding="ISO-8859-1"/>
<xsl:template match="@*">
    <xsl:element name="{local-name(.)}" namespace="{namespace-uri(.)}">
        <xsl:value-of select="." />
    </xsl:element>
</xsl:template>
</xsl:stylesheet>
```

## Typage incontrôlable

```
<xsl:variable name="expressionXPath">/doc/section[3]/section[2]</xsl:variable>
.../...
<xsl:apply-templates select="$expressionXPath" />
```


 Non

Cet exemple ne fonctionne pas

- la variable `expressionXPath` est une **chaîne de caractères**
- la valeur de l'attribut `select` doit être une **expression XPath**
- il n'y a pas de fonction de **transtypage** d'une **chaîne de caractères** en une **expression XPath**

## Fragmentation des résultats impossible

Il n'est **pas possible** de **scinder** la **sortie** du **processeur XSLT** en plusieurs tranches

Il n'est **pas possible** dans XSLT de **spécifier** la **nature** du **flux de sortie**

(pour écrire dans un fichier par exemple)

Cela doit être réalisé par des **processus externes** à XSLT,  
ou par des **extensions propriétaires** :

Apache Xalan

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:redirect="org.apache.xalan.xslt.extensions.Redirect"
  extension-element-prefixes="redirect">
.../...
<xsl:template match=".../...">
  <redirect:write select="$filename">
    <!-- toutes les sorties sont écrites dans le fichier $filename -->
  </redirect>
</xsl:template>
```



Comment ? Par l'utilisation d'une **instruction de traitement** spécifique :  
xml-stylesheet

Convient : à XSLT et aux CSS

CSS : rendu pauvre (dans l'ordre du document, sur les éléments seulement)

Syntaxe : (inclure dans le document XML)

```
<?xml-stylesheet href="mystyle.css" type="text/css"?>
```

```
<?xml-stylesheet href="mystyle.xsl" type="text/xsl"?>
```

Pour associer plusieurs feuilles de style en fonction du média :

```
<?xml-stylesheet href="mystyle.xsl" type="text/xsl"?>  
<?xml-stylesheet href="mystyle.wml.xsl" type="text/xsl" media="wap"?>
```

Ce qu'on ne peut pas faire :



- passer des paramètres
- brancher un URIResolver
- maîtriser la sérialisation (PDF, GIF, PNG, JPEG)
- transformer un nœud autre que le document



A éviter : les feuilles de styles sont applicables à une classe de documents (DTD)  
Ce qui est commun doit être factorisé : déporter l'association d'une feuille de style  
au niveau du processus de publication des documents, au lieu du document lui-même.

Bibliothèques de règles existantes :

- manipulation de chaînes
- dates

...

<http://xslt1.sourceforge.net>

XSLT design patterns :

- Muenchian method
- Kaysian method
- Wendel Piez method
- Oliver Becker's method

Extensions :

"standardisation" des extensions

<http://www.exslt.org>

bibliothèques mathématiques,  
manipulation de chaînes...

Organisation :

Modulariser les programmes :

- utiliser `xsl:import` et `xsl:include`
- distinguer les règles fonctionnelles des règles de formatage
- utiliser les espaces de nommage pour les collections de règles

Performances

éviter d'utiliser `//` (qui est un gouffre pour les performances)  
si on connaît le chemin

Réseau

éviter de surcharger le résultat  
avec de l'indentation inutile, qui  
ne devrait servir qu'à des fins de  
debuggage

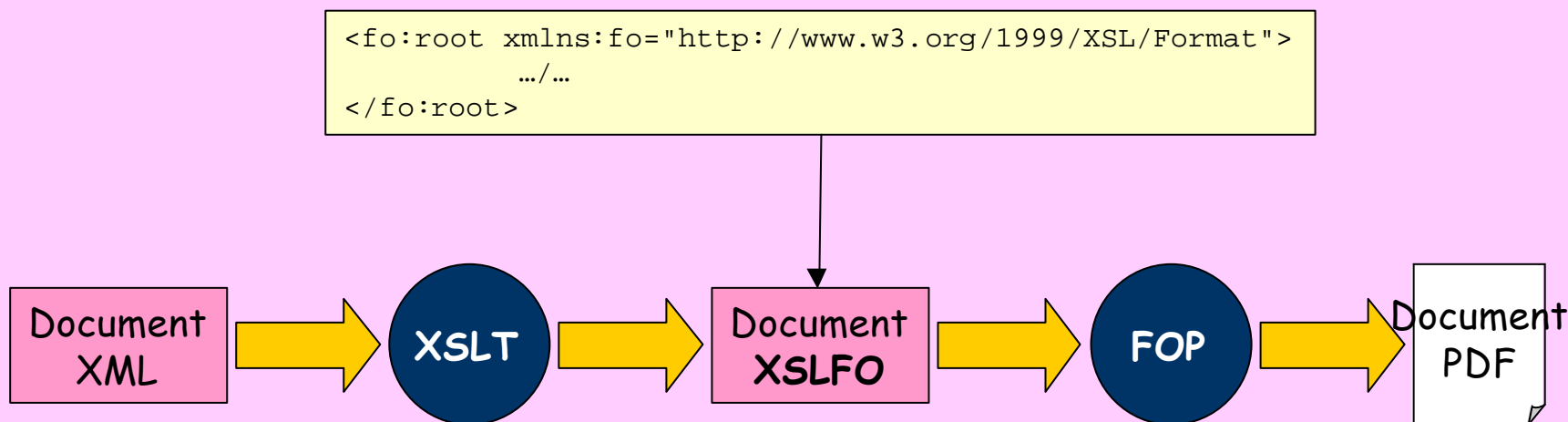
```
<xsl:output method="html" indent="no"/>
```

## Formattage

<http://www.w3.org/TR/xsl/>

XSLFO est une application XML (un jeu de balises) qui permet de décrire la mise en page d'un document.  
S'inspire de CSS.

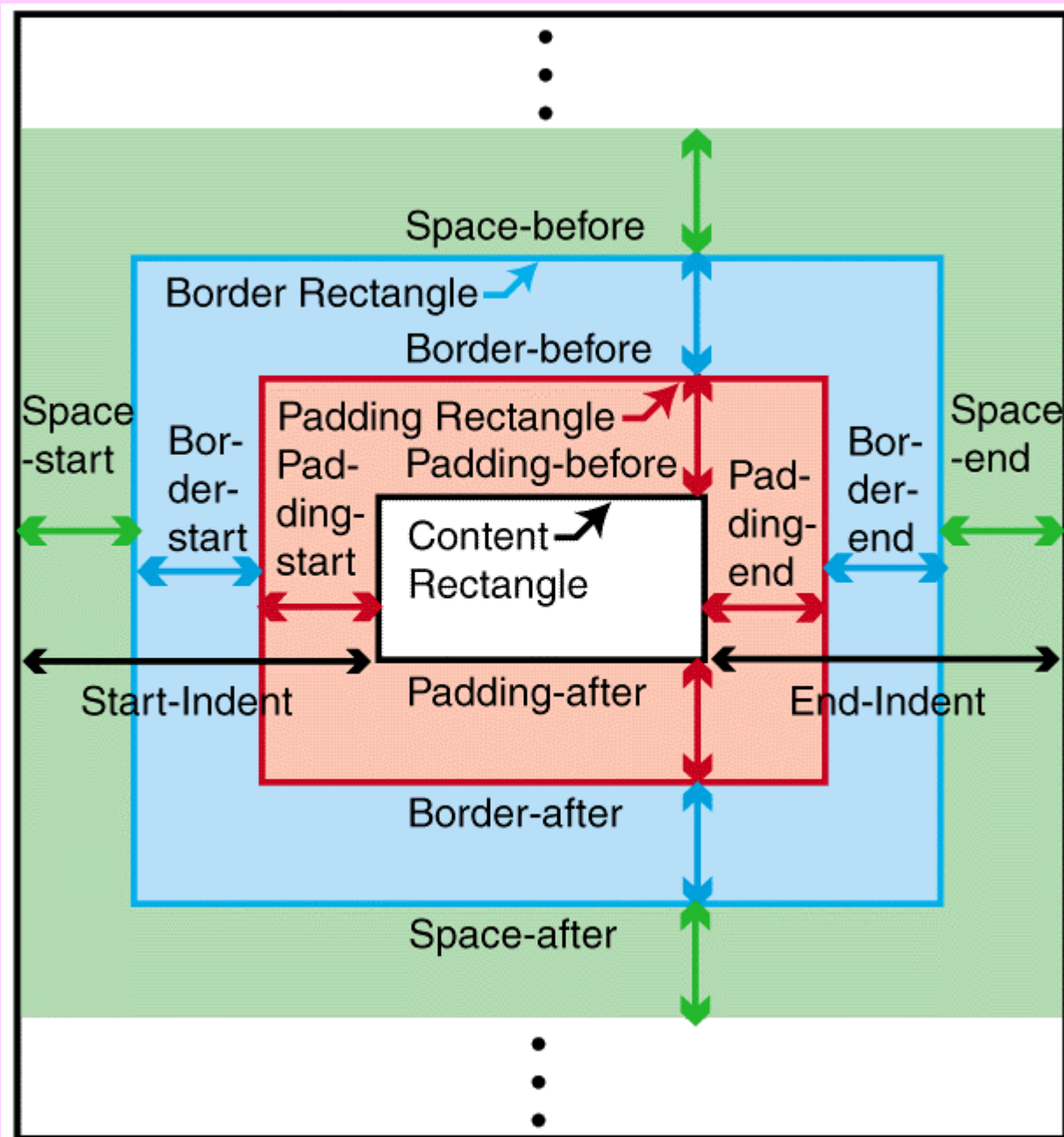
## Chaîne de traitement

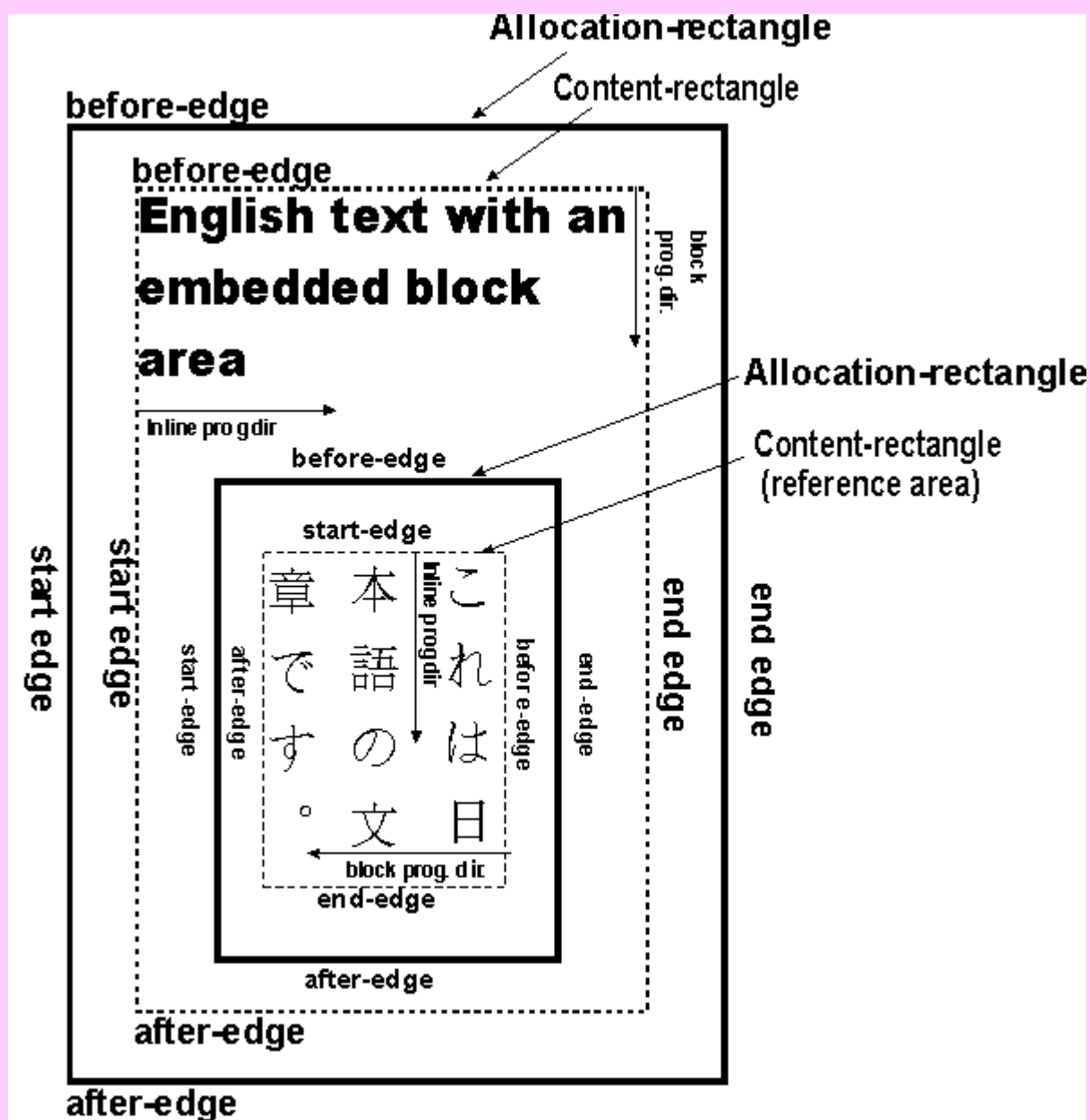


Formatting  
Objects  
Processor



<http://xml.apache.org/fop>





```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
  <fo:layout-master-set>
    <!-- gabarit des pages -->
  </fo:layout-master-set>
  <fo:page-sequence master-reference="content">
    <!-- data -->
  </fo:page-sequence>
</fo:root>
```

## Un seul gabarit dans XSLFO 1.0

```
<fo:simple-page-master
  margin-right="1cm"
  margin-left="1cm"
  margin-bottom="1cm"
  margin-top="1cm"
  page-width="21cm"
  page-height="29.7cm"
  master-name="content"
>
</fo:simple-page-master>
```

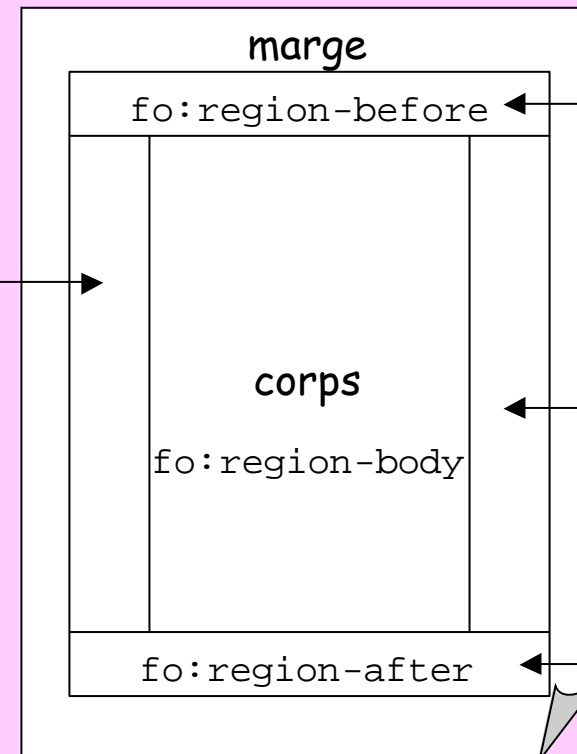
5 régions dont la disposition est déterminée par le sens de l'écriture :

- Hébreu
- Chinois traditionnel

→ une disposition spécifique des régions est prévue

page rectangulaire, largeur et hauteur fixes, marges

début de région  
fo:region-start



région avant

fin de région  
fo:region-end

région après

L'attribut `region-name` des éléments `fo:region-xxx` permet de nommer les régions (par défaut, leur nom est `xsl-region-xxx`).  
L'attribut `extent` permet de définir la taille des régions

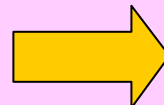
```
<fo:flow flow-name="xsl-region-body">
  <fo:block>
    <!-- ici il y a du contenu -->
  </fo:block>
</fo:flow>
```

Un "bloc" par mise en forme spécifique

L'extrapolation des propriétés de `fo:block` à partir de CSS est souvent triviale

## CSS

```
.foo {
  display: block;
  font-family: Arial;
  font-size: 20pt;
  font-weight: bold;
  text-align: center;
}
```



## XSLFO

```
<fo:block
  font-family="Arial"
  font-size="20pt"
  font-weight="bold"
  text-align="center"
>
```

Autres éléments :

•listes

`fo:list-block`

└─ `fo:list-item`

└─ `fo:list-item-label`

└─ `fo:list-item-body`

