XPATH – XSLT

Yannick Prié
UFR Informatique – Université Lyon 1

UE2.2 – Master SIB M1 – 2005-2006

Objectif du cours

- Xpath
  - syntaxe permettant de désigner des informations dans un arbre XML
  - sous la forme de chemins (*paths*)
- XSL – XSLT
  - expression en XML de transformations à opérer sur un arbre XML
    - transcodage d'un document XML vers un autre
    - présentation de documents XML aux utilisateurs

Plan

- **XPATH**
- XSLT

XPath

- Spécification W3C
- Version actuelle : 1.0 (16/11/1999)
  - Xpath2.0
    - CR : Candidate Recommendation depuis 3/11/05
- Objectif :
  - localiser des documents / identifier des sous-structures dans ceux-ci
- Utilisé par d'autres spécifications XML
  - XPointer, XQuery, XSLT…

Localisation de documents XML

- La localisation du document XML est prise en charge par une URL
- Uniform Resource Locator

  `protocole://adresse/chemin`
- Ressources locales

  `file:///2005-2006/XML-CM3.ppt`
- Ressources distantes

  `http://www.univ-lyon1.fr/`

  `http://liris.cnrs.fr/yprie/ens/O5-06/SIB-RDD/CM1-2pp.pdf`

Exemples d'utilisations

- Ressources locales

```
<!ENTITY ent SYSTEM
  "file:///c:/ents/ent.xml">
<!ENTITY ent SYSTEM "../folder/ent.xml">
```

- Ressources distantes

```
<!ENTITY ent SYSTEM
  "http://w3c.org/ents/ent.xml">
<!ENTITY ent SYSTEM
  "ftp://w3c.org/gifs/pic.gif">
```

Repérer des fragments XML

- Avec les identificateurs XML
  - Utilisation des attributs identificateurs uniques (type d'attribut ID)

```
<link href="../files/detail.xml#part3">
  Link to Part 3 </link>
```

```
...                                    DTD
<!ELEMENT el ...>
<!ATTLIST el label ID #REQUIRED>
...
```

```
...
<el label="part3">
...                                    xml
```


Les limites des identificateurs

- Les identificateurs ne sont pas suffisants
  - donner un identificateur unique à chaque élément peut être pénible
  - l'identité des éléments peut ne pas être connue
  - les identificateurs ne peuvent pas identifier des fragments de texte
  - il peut être peu pratique d'identifier des objets en listant tous leurs identificateurs

Contexte et éléments XML

- La signification d'un élément peut dépendre de son contexte
  - `<book><title>…</title></book>`
    `<person><title>…</title></person>`
- Supposons que l'on cherche le titre d'un livre, pas le titre d'une personne
- Idée
  - exploiter le contexte séquentiel et hiérarchique de XML pour spécifier des éléments par leur contexte (*i.e.* leur position dans la hiérarchie)
  - exemple : `book/title ≠ person/title`

Xpath : principe général

- Décrire un modèle de chemin dans un arbre XML
  → expression
- Recupérer les nœuds qui répondent à ce chemin en utilisant l'expression
  → résultat de l'application de l'expression à l'arbre XML
- Une expression sera utilisée et appliquée au sein de différentes syntaxes
  - URL : `http://abc.com/getQuery?/book/intro/title`
  - XSL : `<xsl:pattern match="chapter/title">…</xsl:pattern>`
  - Xpointer :

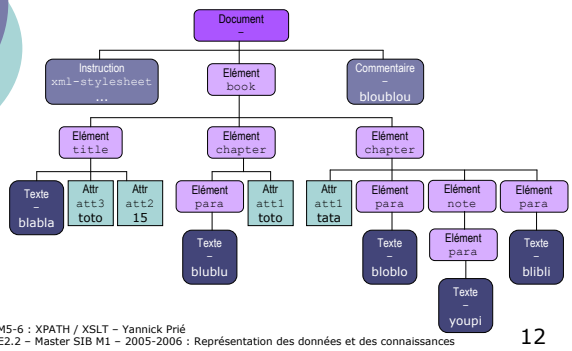
```
<link href="./doc.xml#xptr(book/intro/title)">
  Link to introductory title
</link>
<!-- désignation comme valeur de l'attribut href de
l'élément title situé dans l'élément intro, situé dans
l'élément book, éléments qui se trouvent dans le fichier
doc.xml, lui-même situé dans le dossier où se trouve le
fichier XML en cours -->
```


Document/arbre/nœuds Xpath

- Dans XML
  - nœud = élément = noeud de l'arbre XML
- Dans Xpath
  - extension de la notion de noeud
  - l'arbre XPATH contient toutes les informations repérables dans ses noeuds de différentes types
    - `Document` = noeud racine
    - Noeuds éléments
    - Noeuds attributs
    - Noeuds textes
    - Noeuds instructions de traitement
    - Noeuds commentaires
    - (Noeuds espaces de nom)

(adapté d'après le cours de T. Accary)

Exemple de référence



Version XML

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet href="fichier.xsl"
   type="text/xsl"?>
<book>
<title att3="toto" att2="15">blabla</title>
<chapter att1="toto">
<para>blublu</para>
</chapter>
<chapter att1="tata">
<para>bloblo</para>
<note>
<para>youpi</para>
</note>
<para>blibli</para>
</chapter>
</book>
<!-- bloublou -->
```


Chemins de localisation

- Les expressions identifient des noeuds par leur position dans la hiérarchie
- Permet de
  - monter/descendre dans la hiérarchie de l'arbre XML
  - aller voir les voisins (frères) d'un noeud
  - *en fait : suivre des axes*
- Un chemin peut être
  - relatif
    - à partir de l'endroit où l'on est
  - absolu
    - à partir de la racine

Chemins relatifs

- On se place dans le contexte d'un noeud
- A partir de là, on explore l'arbre XML, et on garde les noeuds qui vérifient l'expression
- Exemple
  - **`para`** (ou **`child::para`**) sélectionnera les fils du noeud courant qui ont le nom '**`para`**'

```
<chapter>            //noeud courant
  <para>…</para>     //Sélectionné
  <note>
    <para>…</para>   //Non sélectionné
  <note>
  <para>…</para>     //Sélectionné
</chapter>
```


Chemins absolus

- Expression identique aux chemins relatifs, mais
  - tout chemin absolu commence par '/'
  - signifie qu'on part de l'élément racine
- Exemple
  - Trouver tous les éléments '**`para`**' dans un document
    - **`//para`**
    - **`/descendant-or-self::node()/para`**

Chemins à plusieurs étapes

- Séparer les étapes par des '/'
- Exemple
  - **`book/title`** (version courte)
  - **`child::book/child::title`** (version longue)
  - depuis le noeud courant, on sélectionne d'abord `book`, qui devient le contexte courant, puis on sélectionne `title`

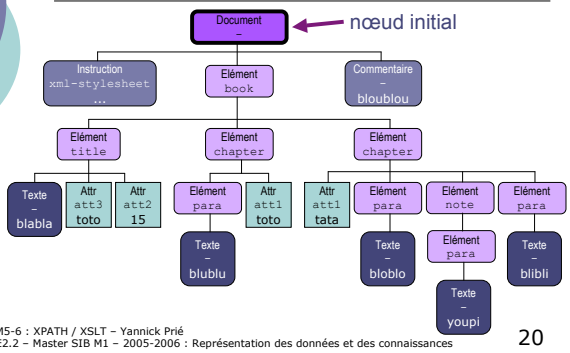
Notion d'étape Xpath

- Une étape contient trois composants
  `Axe :: Filtre [ Prédicat ]`
  - axe
    - sens de parcours des nœuds
  - filtre
    - type des nœuds retenus
  - prédicats (on peut les enchaîner)
    - propriétés satisfaites par les nœuds retenus
- Exemple
  - **`child :: chapter [@att1 = "toto"]`**
- Remarques
  - il existe une syntaxe bavarde (verbose) et une syntaxe raccourcie, plus pratique
  - possibilité de multiples expressions séparées par '|'
    - équivalent d'un OU

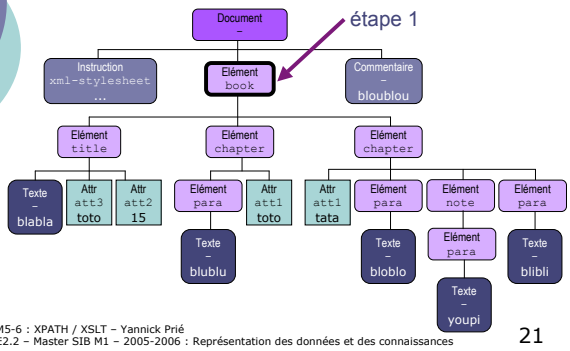
Evaluation d'une expression Xpath

- Expression = séquence d'étapes
- On part du nœud contexte (ou de la racine), on évalue l'étape 1 : récupération d'un ensemble de nœuds
- Pour chacun de ces nœuds
  - il devient le nœud contexte
  - on évalue l'étape 2 : récupération d'un ensemble de nœuds
    - pour chacun de ces nœuds
      - ...

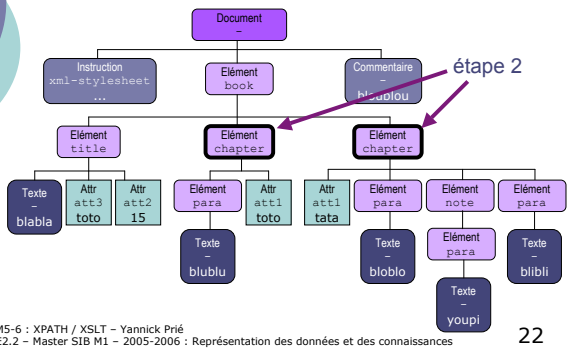
/child::book/child::chapter/attribute::att1
/book/chapter/@att1 (expression raccourcie)



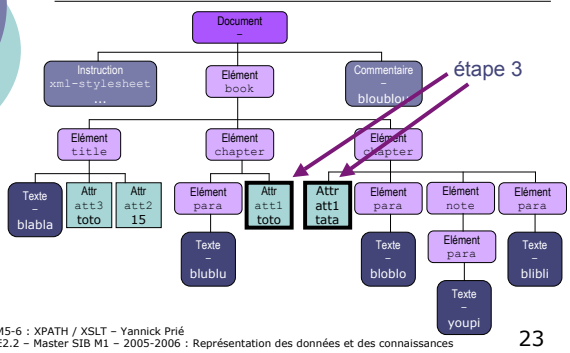
/child::book/child::chapter/attribute::att1
/book/chapter/@att1



/child::book/child::chapter/attribute::att1
/book/chapter/@att1



/child::book/child::chapter/attribute::att1
/book/chapter/@att1

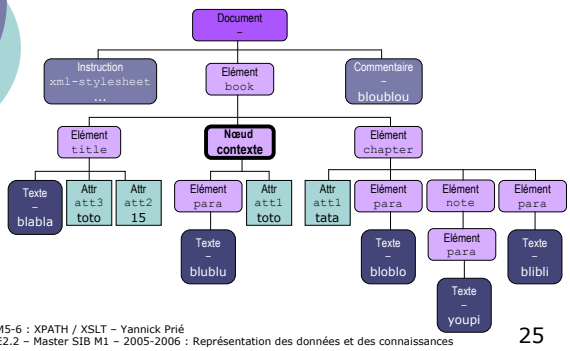


Axe :: Filtre [Prédicat]

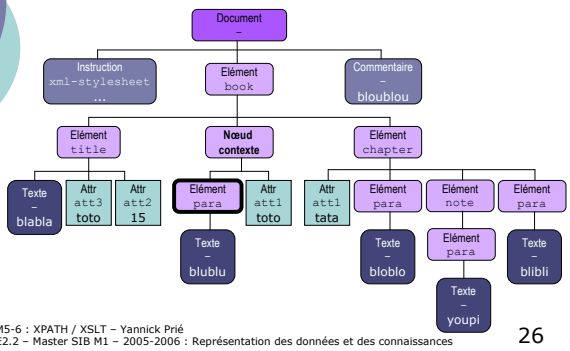
Axes : directions à suivre

- self:: (abrégé : .)
- child:: (abrégé : *rien*)
- attribute:: (abrégé :@)
- parent:: (abrégé : ..)
- descendant::
- descendant-or-self:: (abrégé : //)
- ancestor::
- ancestor-or-self::
- following::
- following-sibling::
- preceding::
- preceding-sibling::

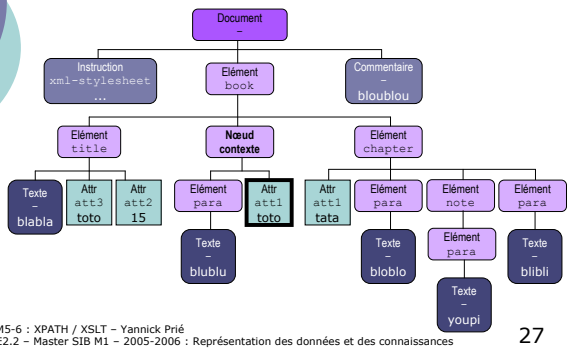
Axe self : `self::node()`



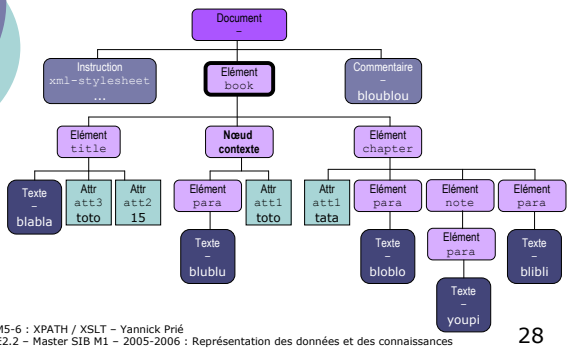
Axe child : `child::node()`



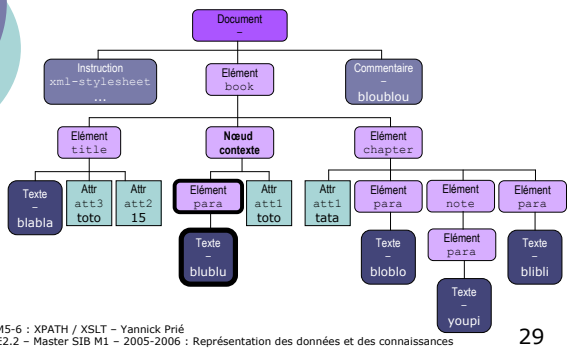
Axe attribute : `attribute::att1`



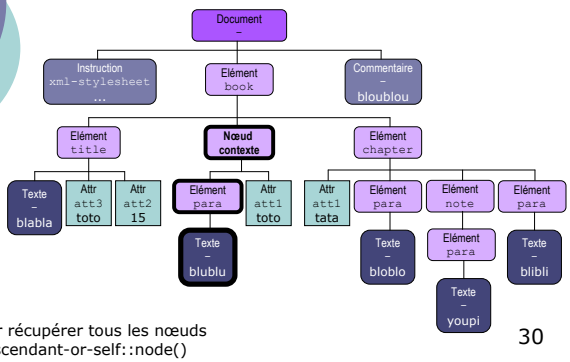
Axe parent : `parent::book`



Axe descendant : `descendant::node()`

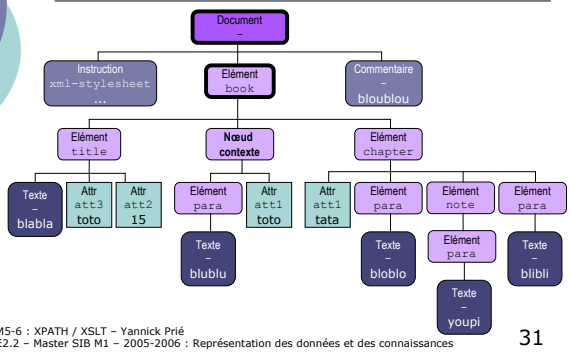


Axe descendant-or-self : `descendant-or-self::node()`

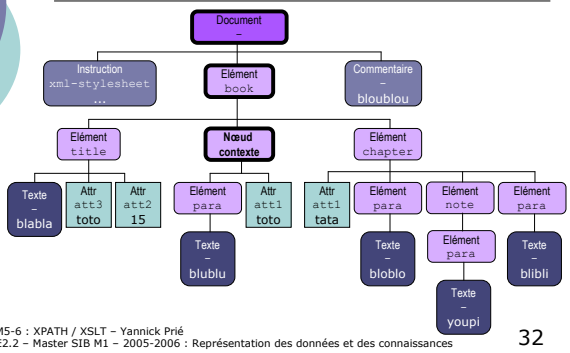


Pour récupérer tous les nœuds /descendant-or-self::node()

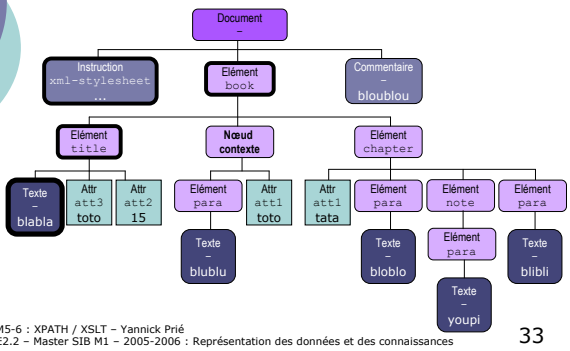
Axe ancestor : `ancestor::node()`



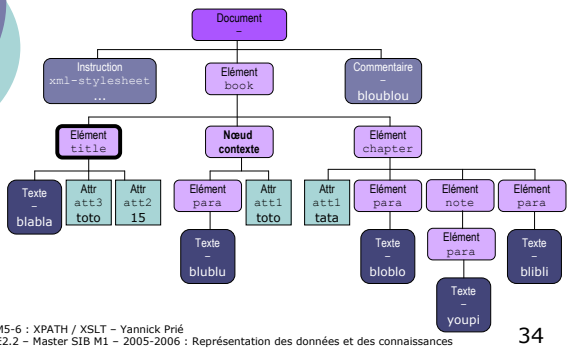
Axe ancestor-or-self : `ancestor-or-self::node()`



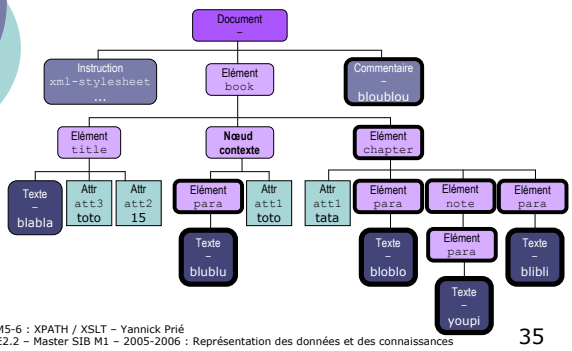
Axe preceding : `preceding::node()`



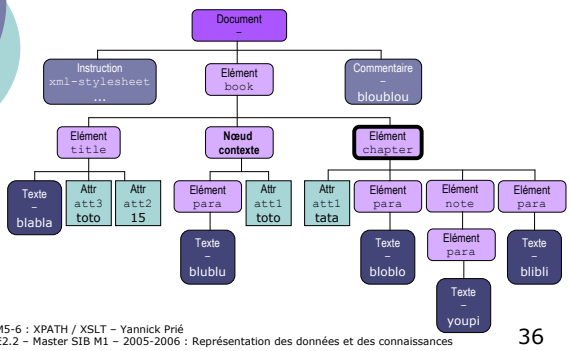
Axe preceding-sibling : `preceding-sibling::node()`



Axe following : `following::node()`



Axe following-sibling : `following-sibling::node()`



Filtres

- Filtrage par le nom
  - Éléments
  - Attributs
  - Instructions de traitement
- Filtrage par le type

Filtrage par le nom

- Nom connu
  - /book/chapter/note
- Nom inconnu
  - Utiliser le joker '`*`' pour tout élément simple
    - `A/*/B` permet de trouver `A/C/B` et `A/D/B`
    - version longue : `child::*`
  - Utilisation de plusieurs astérisques, plusieurs niveaux de correspondance
    - attention à contrôler ce qu'il se passe
      - nombre de niveaux
      - éléments trouvés
- Pour un attribut
  - @nom-attribut

Filtrage par le type

- text()
  - Contenu textuel
- comment()
  - Commentaire
- processing-instruction()
  - Instruction de traitement
- node()
  - Tout type de nœud
- @*
  - Toutes les valeurs de tous les attributs

Quelques exemples

- `chapter//para` (noeud contexte = book)
  `child::chapter/descendant-or-self::node()/child::para`
- `.//para` (noeud contexte = book)
  `self::node()/descendant-or-self::node()/child::para`
- `../title` (noeud contexte = chapter)
  `parent::node()/child::title`
- `note | /book/title` (noeud contexte = 2ème chapter)
- `./*` (noeud contexte = book)
- `/comment()`
- `.//para/text()` (noeud contexte = book)
- `/descendant::node()/@att2`

Filtres avec prédicats

- Les chemins de localisation ne sont pas forcément assez discriminants
  - peuvent fournir une liste de noeuds
- On peut filtrer ces listes avec des prédicats
  - le prédicat est indiqué entre crochets '`[ ]`'
- Le prédicat le plus simple utilise la fonction `position()`
  - `para[position() = 1]` `//1er para`
  - `chapter[2]` `//2eme chapter`
- Possibilité de combiner les tests avec '`and`' et '`or`'
  - `//*[ self::chapter and @att1="tata" ]`
    (`self::chapter` : test1 ; `@att1="tata"` : test2)

Tests sur les positions / texte

- `last()`
  - Récupère le dernier noeud dans la liste
- `count()`
  - Evalue le nombre d'items dans la liste
    `child::chapter [count(child::para) = 2]`
- `string(…)`
  - Récupère le texte d'un élément en enlevant toutes balises

Exemples

- `/book/chapter[@att1]`
  - les nœuds chapter qui ont un attribut att1
- `/book/chapter[@att1="tata"]`
  - les nœuds chapter qui ont un attribut att1 valant 'tata'
- `/book/chapter/descendant::text()[position()=1]`
  - Le premier nœud de type Text descendant d'un /book/chapter
  - s'abrège en /book/chapter/descendant::text()[1]
- `/book/chapter[count(para)=2]`
  - Les nœuds chapter qui ont deux enfants de type para
- `//chapter[child::note]`
  - Les nœuds chapter qui ont des enfants note

Prédicat : divers

- Pour les booléens
  - not(), and, or
- Pour les numériques
  - <, >, != (différent)
  - +, -, *, div (division entière), mod (reste idv entière)
  - number() pour essayer de convertir
  - autres opérateurs : `round()`, `floor()`, `ceiling()`
- Exemples
  - `para [not(position() = 1)]`
  - `para [position() = 1 or last()]`
  - `//node()[number(@att2) mod 2 = 1]`
    - les nœuds avec un attribut att2 impair

Tests sur les chaînes

- Possibilité de tester si les chaînes contiennent des sous-chaînes
  - `<note>hello there</note>`
    - `note [contains(text(), "hello")]`
  - `<note><b>hello</b> there</note>`
    - l'expression précédente ne fonctionne pas `(note/text()` donne `"there")`
    - utiliser plutôt `note[contains(., "hello")]`
    - '.' est le noeud courant, et on parcourera tous les enfants

Tests sur les chaînes (2)

- `starts-with(chaine, motif)`
  - `note[starts-with(., "hello")]`
- `string(chaine)`
  - `note[contains(.,string("pi"))]`
- `string-after(chaine, terminateur)`
- `string-before(chaine, terminateur)`
- `substring(chaine, offset, longueur)`

Tests sur les chaînes (3)

- `normalize(chaine)`
  - enlève les espaces en trop
- `translate(chaine, source, replace)`
  - `translate(., "+", "plus")`
- `concat(strings)`
- `string-length(string)`

Encore des exemples

- `/book/chapter/child::para[child::note or text()]`
  - Tout élément para fils de chapter ayant au moins un fils note ou un fils text
- `/descendant::chapter[attribute::att1 or @att2]`
  - Tout élément chapter ayant un attribut att1 ou att2
- `//*[note]`
  - Tout élément ayant un fils note
- `* [self::note or self::para]` (dans le contexte de chapter)
  - Tout élément note ou para fils du nœud contexte

Quelques fonctions

- S'appliquent sur un ensemble de noeuds
  - id(liste identificateurs) : récupère les éléments ayant ces identificateurs
    - nécessité d'avoir DTD / schéma
    - Ex. id('id54' '678')
  - count()
    - compte le nombre de nœuds. Ex. count(//para)
  - max()
    - rend la valeur maximale
  - sum()
    - rend la somme (les nœuds doivent correspondre à des valeurs numériques, traductibles par number())
  - distinct-values() (Xpath 2.0)
    - élimine les doublons

Conclusion

- Xpath permet de retrouver toutes sortes d'information dans les documents XML
  - requêtes
  - transformation : lire une information sous une forme, l'écrire sous une autre forme → XSL/XSLT
- Nous avons vu les grands principes
  - pour la description systématique de la syntaxe
    - sites de références
  - pour plus d'exemples
    - sites avec tutoriaux
- Ce cours : présentation de XPATH 1.0
  - des améliorations dans XPATH 2.0