LABD

Master Info M1 2015-2016

Cours 6 : Programmer avec XQuery

XSL versus XQuery

- Définir un « vrai » langage de requêtes pour XML

- Par rapport à XSLT, l'objectif de transformation est le même et certains éléments sont communs (XPath, XML-Schema).

- XQuery : point de vue ”bases de données”,

- XSLT : point de vue ”gestion de documents”.

Historique

✓1998 : atelier de travail W3C pour XML Query

✓1999 : groupe de travail W3C sur XML Query

✓2000 : le groupe de travail définit les requêtes, les cas d'utilisation et le modèle de donnée.

✓2001 : le groupe de travail publie un premier *draft*

✓2007 : XQuery 1.0

✓2014 : XQuery 3.0

XQuery (1.0)

- n'est pas un dialecte XML
- est un sur-ensemble de XPath 2.0
- utilise le même modèle de données que XPath : séquences d'items, un item étant un nœud ou une valeur atomique.
- utilise le typage de XML-Schema
- permet de définir des fonctions, et utiliser toutes les fonctions prédéfinies pour XPath.

Règles générales pour XQuery

- XQuery est un langage sensible à la casse. Les mots clés sont en minuscules.

- Chaque expression a une valeur, et pas d'effet de bord.

- Les expressions sont composables.

- Les expressions peuvent générer des erreurs. Les commentaires sont possibles `(: un commentaire :)`

Qu’est ce qu’une requête XQuery?

Une requête est une expression qui

- Lit une séquence de fragments XML ou de valeurs atomiques.

- Retourne une séquence de fragments XML ou de valeurs atomiques.

Qu’est ce qu’une requête XQuery?

Les formes principales que peuvent prendre une expression XQuery sont :

- Expressions de chemins
- Constructeurs
- Expressions FLWOR
- Expressions de listes
- Conditions
- Expressions quantifiées
- Expressions de types de données
- Fonctions

Contexte d’évaluation

Les expressions sont évaluées relativement à un contexte:

- Espace de nom
- Variables
- Fonctions
- Date et heure
- Item contexte (nœud courant)
- Position (dans la séquence en train d'être traitée)
- Taille de cette séquence

Les expressions de chemin

Vous connaissez déjà XQuery !

Une expression de chemin XPath est une requête XQuery :

Un chemin retourne une liste composée de nœuds ou de valeurs atomiques provenant d'un ou plusieurs documents XML.

Exemple

TP5/6, question 2 : extraire le sous-arbre des clubs.

```
xquery version "1.0";
doc("championnat.xml")//clubs
```

Autre exemple

TP5/6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";
<html>
  <body>
    <h1>Les clubs de Ligue 1 -- saison 2013-2014</h1>
    <table>
      <thead><tr><th>ville</th><th>club</th></tr></thead>
      <tbody>
      { for $c in doc("championnat.xml")//clubs/club return
        <tr>
          <td>{$c/ville/text()}</td>
          <td>{$c/nom/text()}</td>
        </tr>
      }
      </tbody>
    </table>
  </body>
</html>
```

Autre exemple

TP5/6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";

        for $c in doc("championnat.xml")//clubs/club return
              $c/ville/text()
```

Autre exemple

TP5/6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";

        for $c in doc("championnat.xml")//clubs/club return

         <td>{$c/ville/text()}</td>
```

Autre exemple

TP5/6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";

        for $c in doc("championnat.xml")//clubs/club return

          <td>{$c/ville/text()}</td>
          <td>{$c/nom/text()}</td>
```

Autre exemple

TP5/6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";

 for $c in doc("championnat.xml")//clubs/club return
<tr>
  <td>{$c/ville/text()}</td>
  <td>{$c/nom/text()}</td>
</tr>
```

Autre exemple

TP5/6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";
```

```
<tbody>
{ for $c in doc("championnat.xml")//clubs/club return
  <tr>
    <td>{$c/ville/text()}</td>
    <td>{$c/nom/text()}</td>
  </tr>
}
</tbody>
```

Autre exemple

TP5/6, question 3 : construire un tableau HTML à partir des clubs.

```
xquery version "1.0";
<html>
  <body>
    <h1>Les clubs de Ligue 1 -- saison 2013-2014</h1>
    <table>
      <thead><tr><th>ville</th><th>club</th></tr></thead>
      <tbody>
      { for $c in doc("championnat.xml")//clubs/club return
        <tr>
          <td>{$c/ville/text()}</td>
          <td>{$c/nom/text()}</td>
        </tr>
      }
      </tbody>
    </table>
  </body>
</html>
```

Constructeur

- Dans l'exemple précédent, les noms des éléments sont connus, on peut donc écrire les balises telles quelles.

- On peut définir des éléments ou attributs dont le nom et le contenu sont calculés (comme en XSLT).

```
element
  {fn:node-name($e)}
  {$e/@*, 2 * fn:data($e)}
```

si `$e` a pour valeur
`<length units="inches">5</length>`,
alors le résultat de cette expression est l'élément :
`<length units="inches">10</length>`.

Autres exemples

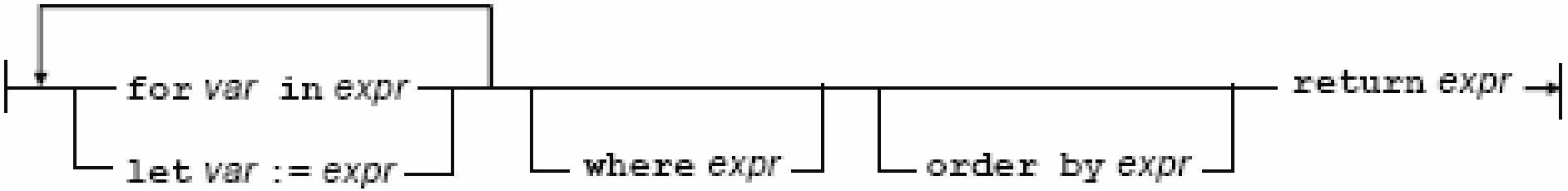
```
element book {
  attribute isbn {"isbn-0060229357" },
  element title { "Harold and the Purple Crayon"},
  element author {
    element first { "Crockett" },
    element last {"Johnson" }
  }
}
```

équivalent à

```
<book isbn="isbn-0060229357">
  <title>Harold and the Purple Crayon</title>
  <author>
    <first>Crockett</first>
    <last>Johnson</last>
  </author>
</book>
```

Expression FLWOR

Une expression FLWOR lie des variables, applique des prédicats, et construit un nouveau résultat.



FOR and LET clauses generate a list of tuples of bound variables, preserving input order.

WHERE clause applies a predicate, eliminating some of the tuples

ORDER BY clause imposes an order on the surviving tuples

RETURN clause is executed for each surviving tuple, generating an ordered list of outputs

Expression FLWOR

- **`for $x in Expr`** la variable **`$x`** prend successivement chacune des valeurs des items de la séquence retournée par **`Expr`**

- **`let $x := Expr`** la variable **`$x`** prend la valeur de la séquence retournée par **`Expr`**.

- On peut définir une clause **`for`** ou **`let`** avec plusieurs variables :
`for $i in (1, 2), $j in (3, 4) return ($i , $j)`
a pour résultat la séquence `(1 3 1 4 2 3 2 4)`

- **`where Expr`** on ne garde que les valuations des variables qui satisfont l'expression

Expression FLWOR

- **`order by`** permet de trier le résultat. Plus précisément, on trie les tuples de valeurs associés aux variables.

```
for $e in $employees
order by $e/salary descending, $e/name ascending
return $e/name
```

- **`return Expr`** l'expression est évaluée pour chaque tuple de valeurs associé aux variables, et les résultats de ces évaluations sont concaténés pour former la séquence résultat (comme si on avait l'opérateur virgule).

Exemple

```
for $s in (<one/>, <two/>, <three/>) return
<out>{$s}</out>
```

a pour résultat

```
<out><one/></out><out><two/></out><out><three/></out>
```

Exemple

```
let $s := (<one/>, <two/>, <three/>)
return <out>{$s}</out>
```

a pour résultat

```
<out><one/><two/><three/></out>
```

Exemple

```
xquery version "1.0";
declare default element namespace "http://labd/art";
let $art := doc("./essais/artistes.xml")
let $cat := doc("./essais/catalogue-1.xml")
return
  <art>
    <artistes>{
      for $a in $art//artiste
        let $o := $cat//oeuvre[auteur/nom = $a/nom]
        return
          <artiste>
            {$a/*}
            <nb-oeuvres>{count($o)}</nb-oeuvres>
          </artiste>
        }</artistes>
    {$art/art/mouvements}
  </art>
```

Exemple (suite)

on ne conserve que les artistes qui ont au moins 2 oeuvres

```
<artistes>{
  for $a in $art//artiste
  let $o := $cat//oeuvre[auteur/nom eq $a/nom]
  where count($o) ge 2
  return
    <artiste>
      {$a/*}
      <nb-oeuvres>{count($o)}</nb-oeuvres>
    </artiste>
  }
</artistes>
```

Jointures entre documents

```
for $p in doc("www.irs.gov/taxpayers.xml")//person
for $n in doc("neighbors.xml")//neighbor[ssn = $p/ssn]
return
<person>
   <ssn> { $p/ssn/text() } </ssn>
   { $n/name }
   <income> { $p/income/text() } </income>
</person>
```

Autre exemple

```
for $d in doc("depts.xml")//deptno
  let $e := doc("emps.xml")//employee[deptno = $d]
  where count($e) >= 10
  order by avg($e/salary) descending
  return
  <big-dept>
    { $d,
    <headcount>{count($e)}</headcount>,
    <avgsal>{avg($e/salary)}</avgsal>
     }
  </big-dept>
```

Résultat : Retourne la liste des départements avec plus de 10 employés, classés par salaire moyen

Numéroter en `XQuery`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<stock>
  <occases>
    <item prix="2000">
      <marque>Renault</marque>
      <modele>Clio II</modele>
      <annee>1998</annee>
    </item>
    <item prix="4800">
      <marque>Peugeot</marque>
      <modele>307</modele>
      <annee>2001</annee>
    </item>
    <item prix="15000">
      <marque>Renault</marque>
      <modele>Clio IV</modele>
      <annee>2010</annee>
    </item>
  </occases>
</stock>
```

Numéroter en `XQuery`

- À l'aide d'une requête `XQuery`, on souhaite extraire de ce fichier les modèles de la marque `"Renault"` dans des éléments de nom `p` en numérotant les noms de modèle. Concrètement, à partir du fichier précédent, on veut obtenir :

```
<?xml version="1.0" encoding="UTF-8"?>
   <p>1. Clio II</p>
   <p>2. Clio IV</p>
```

```
<stock>
   <occases>
      <item prix="2000">
         <marque>Renault</marque>
         <modele>Clio II</modele>
         <annee>1998</annee>
      </item>
      <item prix="4800">
         <marque>Peugeot</marque>
         <modele>307</modele>
         <annee>2001</annee>
      </item>
      <item prix="15000">
         <marque>Renault</marque>
         <modele>Clio IV</modele>
         <annee>2010</annee>
      </item>
   </occases>
</stock>
```

Numéroter en `XQuery`

- Un programmeur (débutant) en `XQuery` propose la requête suivante :

```
xquery version "1.0";
let $count := 0
for $prod in //item[marque eq "Renault"]
let $count := $count + 1
return <p>{$count}. {data($prod/modele)}</p>
```

- Dites pourquoi cette requête ne convient pas et donnez le résultat de l'application de celle-ci sur le fichier `okaz.xml`.

Numéroter en `XQuery`

```
xquery version "1.0";
let $count := 0
for $prod in //item[marque eq "Renault"]
let $count := $count + 1
return <p>{$count}. {data($prod/modele)}</p>
```

```
<stock>
   <occases>
      <item prix="2000">
         <marque>Renault</marque>
         <modele>Clio II</modele>
         <annee>1998</annee>
      </item>
      <item prix="4800">
         <marque>Peugeot</marque>
         <modele>307</modele>
         <annee>2001</annee>
      </item>
      <item prix="15000">
         <marque>Renault</marque>
         <modele>Clio IV</modele>
         <annee>2010</annee>
      </item>
   </occases>
</stock>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p>1. Clio II</p>
<p>2. Clio IV</p>
```

Numéroter en `XQuery`

```
xquery version "1.0";
let $count := 0
for $prod in //item[marque eq "Renault"]
let $count := $count + 1
return <p>{$count}. {data($prod/modele)}</p>
```

```
<stock>
   <occases>
      <item prix="2000">
         <marque>Renault</marque>
         <modele>Clio II</modele>
         <annee>1998</annee>
      </item>
      <item prix="4800">
         <marque>Peugeot</marque>
         <modele>307</modele>
         <annee>2001</annee>
      </item>
      <item prix="15000">
         <marque>Renault</marque>
         <modele>Clio IV</modele>
         <annee>2010</annee>
      </item>
   </occases>
</stock>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p>1. Clio II</p>
<p>1. Clio IV</p>
```

Numéroter en `XQuery`

- Un collègue du programmeur précédent propose plutôt :

```
xquery version "1.0";
for $prod in //item[marque eq "Renault"]
return <p>{$prod/position()}. {data($prod/modele)}</p>
```

- Dites pourquoi cette requête ne convient pas plus et donnez le résultat de l'application de celle-ci sur le fichier `okaz.xml`

Numéroter en `XQuery`

```
xquery version "1.0";
for $prod in //item[marque eq "Renault"]
return <p>{$prod/position()}.
{data($prod/modele)}</p>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p>?. Clio II</p>
<p>?. Clio IV</p>
```

```
<stock>
   <occases>
      <item prix="2000">
         <marque>Renault</marque>
         <modele>Clio II</modele>
         <annee>1998</annee>
      </item>
      <item prix="4800">
         <marque>Peugeot</marque>
         <modele>307</modele>
         <annee>2001</annee>
      </item>
      <item prix="15000">
         <marque>Renault</marque>
         <modele>Clio IV</modele>
         <annee>2010</annee>
      </item>
   </occases>
</stock>
```

Numéroter en `XQuery`

```
xquery version "1.0";
for $prod in //item[marque eq "Renault"]
return <p>{$prod/position()}.
{data($prod/modele)}</p>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p>1. Clio II</p>
<p>1. Clio IV</p>
```

```
<stock>
   <occases>
      <item prix="2000">
         <marque>Renault</marque>
         <modele>Clio II</modele>
         <annee>1998</annee>
      </item>
      <item prix="4800">
         <marque>Peugeot</marque>
         <modele>307</modele>
         <annee>2001</annee>
      </item>
      <item prix="15000">
         <marque>Renault</marque>
         <modele>Clio IV</modele>
         <annee>2010</annee>
      </item>
   </occases>
</stock>
```

Numéroter en `XQuery`

Pour réaliser ce type de numérotation, les expressions `FLOWR` de `XQuery` disposent en fait d'une syntaxe particulière qui permet de définir une variable de numérotation dans un `for`. Cette variable, placée derrière le mot clef `at`, est liée à une valeur entière qui représente justement le numéro d'ordre dans l'itération. Ainsi la requête

```
xquery version "1.0";
for $prod at $count in //item[marque eq "Renault"]
return <p>{$count}. {data($prod/modele)}</p>
```

appliquée au fichier `okaz.xml` fournit bien comme résultat

```
<?xml version="1.0" encoding="UTF-8"?>
<p>1. Clio II</p>
<p>2. Clio IV</p>
```

Numéroter en `XQuery`

- Oui... mais ça ne marche pas toujours aussi simplement dans tous les cas! Donnez le résultat de la requête

```
xquery version "1.0";
for $prod at $count in //item
where $prod/marque eq "Renault"
order by $prod/annee descending, $count ascending
return <p>{$count}. {data($prod/modele)}</p>
```

appliquée au fichier `okaz.xml`

Numéroter en `XQuery`

```
xquery version "1.0";
for $prod at $count in //item
where $prod/marque eq "Renault"
order by $prod/annee descending, $count
ascending
return <p>{$count}. {data($prod/modele)}</p>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p>?. Clio IV</p>
<p>?. Clio II</p>
```

```
<stock>
   <occases>
      <item prix="2000">
         <marque>Renault</marque>
         <modele>Clio II</modele>
         <annee>1998</annee>
      </item>
      <item prix="4800">
         <marque>Peugeot</marque>
         <modele>307</modele>
         <annee>2001</annee>
      </item>
      <item prix="15000">
         <marque>Renault</marque>
         <modele>Clio IV</modele>
         <annee>2010</annee>
      </item>
   </occases>
</stock>
```

Numéroter en `XQuery`

```
xquery version "1.0";
for $prod at $count in //item
where $prod/marque eq "Renault"
order by $prod/annee descending, $count
ascending
return <p>{$count}. {data($prod/modele)}</p>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<p>3. Clio IV</p>
<p>1. Clio II</p>
```

```
<stock>
   <occases>
      <item prix="2000">
         <marque>Renault</marque>
         <modele>Clio II</modele>
         <annee>1998</annee>
      </item>
      <item prix="4800">
         <marque>Peugeot</marque>
         <modele>307</modele>
         <annee>2001</annee>
      </item>
      <item prix="15000">
         <marque>Renault</marque>
         <modele>Clio IV</modele>
         <annee>2010</annee>
      </item>
   </occases>
</stock>
```

Numéroter en `XQuery`

- Écrire une requête `XQuery` qui retourne les modèles de marque `"Renault"`, triés par année décroissante en les numérotant de manière satisfaisante. Cette requête, appliquée sur le document `okaz.xml` doit fournir le résultat :

```
<?xml version="1.0" encoding="UTF-8"?>
<p>1. Clio IV</p>
<p>2. Clio II</p>
```

Bien sûr, la requête doit fonctionner sur tout fichier de même structure que le fichier `okaz.xml`.

solution

```
xquery version "1.0";
for $p at $count in
    for $prod in //item
    where $prod/marque eq "Renault"
    order by $prod/annee descending
    return $prod
return <p>{$count}. {data($p/modele)}</p>
```

Opérations sur des listes (rappel)

- XPath gère des listes de valeurs : `(7,9, <onze/>)`
- XQuery dispose d'opérateurs pour gérer les listes
  - Concaténation
  - Opérations ensemblistes (union, intersection, différence)
  - Fonctions (`remove`, `index-of`, `count`, `avg`, `min`, `max` etc…)

Exemple

```
xquery version "1.0";
let $doc := doc("bib.xml")
for $p in distinct-values($doc//publisher)
let $a := avg($doc//book[publisher = $p]/price)
return
  <publisher>
   <name>{$p}</name>
   <avgprice>{$a}</avgprice>
  </publisher>
```

Résultat : Liste chaque éditeur et le prix moyen de leurs livres

Autres expressions

- `unordered {`expr`}`
- `if (`expr1`) then` expr2 `else` expr3
- `some` var `in` expr1 `satisfies` expr2
- `every` var `in` expr1 `satisfies` expr2

unordered

```
unordered {
  for $p in fn:doc("parts.xml")/parts/part[color = "Red"],
      $s in fn:doc("suppliers.xml")/suppliers/supplier
  where $p/suppno = $s/suppno
  return
    <ps>
       { $p/partno, $s/suppno }
    </ps>
}
```

if then else (rappel)

```
for $h in doc("library.xml")//holding
return
  <holding>
    { $h/title, if ($h/@type = "Journal")
                  then $h/editor
                  else $h/author }
  </holding>
```

some (rappel)

```
for $b in doc("bib.xml")//book
  where some $p in $b//paragraph satisfies
    (contains($p,"sailing") and
    contains($p,"windsurfing"))
  return $b/title
```

every (rappel)

```
for $b in doc("bib.xml")//book
  where every $p in $b//paragraph satisfies
    contains($p,"sailing")
  return $b/title
```

Structure d’une requête

✓Le Prologue contient:

- Déclarations de Namespace
- Importations de schémas
- Importation de Modules
- Définitions de fonctions
- Déclarations de variable globales et externes

✓Le Corps contient: Une expression qui définit le résultat de la requête

Les espaces de nom

✓En XQuery, tous les noms sont des noms qualifiés.

✓Les déclarations sont faites dans le prologue.

```
declare namespace acme = "http://www.acme.com/names" ;

declare default element namespace "http:://www.bar.art" ;
```

Importation de schéma

✓Un espace de nom est défini par un schéma

✓On lie l'espace de nom au préfixe et on importe le schéma par l'instruction suivante

```
import schema namespace acme =
"http://www.acme.com/names"
at
"http://www.acme.com/schemas/names.xsd" ;
```

Espaces de nom prédéfinis

✓`http://www.w3.org/2001/XMLSchema` associé au préfixe `xs:`

✓`http://www.w3.org/2005/xpath-functions`, contenant toutes les fonctions XPATH, associé au préfixe `fn:`

✓il existe aussi le préfixe `local:` pour les fonctions définies par l'utilisateur

Définition de fonctions

✓À faire apparaître dans le prologue

✓Les fonctions ne peuvent pas être surchargées.

✓En XML on n'est pas obligé de tout typer explicitement. XQuery tente de forcer le type à celui attendu par un cast automatique.

Exemple: abs($x) attend un argument numérique

- Si $x est un nombre, retourner sa valeur absolue
- Si $x n'a pas de type, le transformer en nombre
- Si $x est un nœud, extraire sa valeur puis la traiter comme ci-dessus

Il y a donc incompatibilité avec la surcharge !

Exemple

```
define function local:depth($n as node()) as
xs:integer
{
  (: A node with no children has depth 1 :)
  (: Else, add 1 to max depth of children :)
  if (empty($n/*)) then 1
  else max(for $c in $n/* return depth($c)) + 1
} ;
```

Exemple

```
(: determine si une l'année d'une date est bissextile :)
declare function local:bissextile($dt as xs:date) as xs:boolean {
    let $i := fd:annee($dt)
    return $i mod 4 = 0 and ($i mod 100 != 0 or $i mod 400 =0)
};
```

```
(: ajoute des jours à une date :)
declare function local:ajoute-jour($dt as xs:date , $j as xs:integer)
   as xs:date{
    if ($j = 0) then $dt
    else fd:ajoute-jour(fd:ajoute-un-jour($dt) , $j - 1)
} ;
```

Modules

Une requête peut utiliser plusieurs modules

- ✓Module principal
  - Contient la requête
  - Est exécutable
- ✓Module bibliothèque
  - Définit les fonctions et variables
  - Déclare son espace de noms
  - Peut être importé
  - Exporte ses variables et ses fonctions dans son espace de noms
- ✓Importation de modules

Exemple de module bibliothèque

```
xquery version "1.0" ;

(: Un module doit toujours déclarer un namespace :)
module namespace fa = "http://fil.univ-lille1.fr/miage-fa-fc/library" ;

(: le namespace par défaut est celui exporté par le schéma des données de base:)
declare default element namespace "http://fil.univ-lille1.fr/miage-fa-fc" ;

import module namespace fd = "http://fil.univ-lille1.fr/miage-fa-fc/dates"
at "http://www.fil.univ-lille1.fr/FORMATIONS/MIAGE-FC-FA/schemas/dates.xq" ;

(: Le fichier qui contient toutes les données :)
declare variable $fa:miage :=
  doc("http://www.fil.univ-lille1.fr/FORMATIONS/MIAGE-FC-FA/schemas/miage-fa-fc.xml")
;

(: retourne le début d'une semaine de séminaire :)
declare function fa:debut-seminaire($semaine as xs:integer)
   as xs:date{
      let $d := $fa:miage//trimestre[@num="T0"]/du
      return
      if ($semaine = 1) then $d
      else fd:ajoute-jour($d , 7)
};

...
```

Exemple d'importation de module

```
xquery version "1.0" ;
import module
  namespace b = "http://fil.univ-lille1.fr/miage-fa-fc/library"
  at "http://www.fil.univ-lille1.fr/FORMATIONS/MIAGE-FC-FA/
  schemas/miage-fa-fc.xq" ;

  b:edt("M1")
```

```
<?xml version="1.0" encoding="UTF-8"?>
<formation annee="2012">
  <creneaux>
    <creneau>
      <trimestre>T1</trimestre>
      <jour>lundi</jour>
      <de>09:00:00</de><a>12:00:00</a>
      <salle>M5-A2</salle><salle>M5-B13</salle>
      <intervenant>Jacques Dupond</intervenant>
      <matiere>QSI</matiere>
    </creneau>
    <creneau>
      <trimestre>T1</trimestre>
      <jour>lundi</jour>
      <de>13:30:00</de><a>15:30:00</a>
      <salle>M5-A2</salle>
      <intervenant>Pauline Durand</intervenant>
      <matiere>CGEST</matiere>
    </creneau>
    <creneau>
      <trimestre>T3</trimestre>
      <jour>lundi</jour>
      <de>10:00:00</de><a>12:00:00</a>
      <salle>M5-A2</salle><salle>M5-B13</salle>
      <intervenant>Pierre Martin</intervenant>
      <matiere>PEPIT</matiere>
    </creneau>
    <creneau>
      <trimestre>T2</trimestre>
      <jour>mardi</jour>
      <de>09:00:00</de><a>12:00:00</a>
      <salle>MDL-B5-</salle>
      <intervenant>Vincent Lebrun</intervenant>
      <matiere>ANG3</matiere>
    </creneau>
  </creneaux>
</formation>
```

Exercice

1. Donnez des requêtes `XQuery` pour sélectionner sur tout fichier de même structure que le fichier `formation.xml` :

```
declare variable $doc := doc("formation.xml") ;
```

- le nom du premier cours de la journée, le mardi au deuxième trimestre

```
$doc//creneau[trimestre="T2"][jour="mardi"][not(//creneau[trimestre="T2"]
                [jour="mardi"]/de < de)]/matiere/text()
```

- le nombre de salles différentes utilisées durant les deux premiers trimestres

```
count(distinct-values($doc//creneau[trimestre ne "T3"]/salle))
```

```
<?xml version="1.0" encoding="UTF-8"?>
<matieres>
  <matiere>
    <nom-court>QSI</nom-court>
    <nom>Qualite des Systemes d'Information</nom>
    <formation>M2</formation>
    <trimestre>T1</trimestre>
    <semestre>S3</semestre>
    <ects>4</ects>
    <volume>30</volume>
    <intervenant>Jacques Dupond</intervenant>
  </matiere>
  <matiere>
    <nom-court>CGEST</nom-court>
    <nom>Controle de Gestion</nom>
    <formation>M2</formation>
    <trimestre>T1</trimestre>
    <semestre>S3</semestre>
    <ects>3</ects>
    <volume>20</volume>
    <intervenant>Pauline Durand</intervenant>
  </matiere>
  <matiere>
    <nom-court>ANG3</nom-court>
    <nom>Anglais</nom>
    <formation>M2</formation>
    <trimestre>T2</trimestre>
    <semestre>S3</semestre>
    <ects>3</ects>
    <volume>30</volume>
    <intervenant>Vincent Lebrun</intervenant>
  </matiere>
  <matiere>
    <nom-court>PEPIT</nom-court>
    <nom>Projet d'Ensemble des Pratiques Informatiques Transversales</nom>
    <formation>M2</formation>
    <trimestre>T3</trimestre>
    <semestre>S4</semestre>
    <ects>7</ects>
    <volume>40</volume>
    <intervenant>Laurent Lama</intervenant>
  </matiere>
</matieres>
```

Exercice (suite)

2. Écrivez une requête `XQuery` qui calcule, à partir du fichier `matieres.xml`, le volume horaire global de la formation `"M2"` (c'est à dire la somme des volumes horaires (élément volume) de toutes les matières de la formation), réparti par trimestre comme ci-dessous :

```
<?xml version="1.0" encoding="UTF-8"?>
<volumes formation="M2">
        <volume trimestre="T1">50</volume>
        <volume trimestre="T2">30</volume>
        <volume trimestre="T3">40</volume>
</volumes>
```

Solution

```
xquery version "1.0";
declare variable $m := doc("matieres.xml") ;
<volumes formation="M2">
{
  for $t in distinct-values($m//trimestre)
  let $mt := $m//matiere[formation="M2"][trimestre=$t]/volume
   order by $t
  return <volume trimestre="{$t}">{sum($mt)}</volume>
}
</volumes>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<volumes formation="M2">
        <volume trimestre="T1">50</volume>
        <volume trimestre="T2">30</volume>
        <volume trimestre="T3">40</volume>
</volumes>
```

XQuery 3.0

1. `group by` clause in FLWOR Expressions (**3.10.7 Group By Clause**).
2. `tumbling window` and `sliding window` in FLWOR Expressions (**3.10.4 Window Clause**).
3. `count` clause in FLWOR Expressions (**3.10.6 Count Clause**).
4. `allowing empty` in **3.10.2 For Clause**, for functionality similar to outer joins in SQL.
5. `try/catch` expressions (**3.15 Try/Catch Expressions**).
6. Dynamic function call (**3.2.2 Dynamic Function Call**).
7. Inline function expressions (**3.1.7 Inline Function Expressions**).
8. Private functions (**4.18 Function Declaration**).
9. Switch expressions (**3.13 Switch Expression**).
10. Computed namespace constructors (**3.9.3.7 Computed Namespace Constructors**).
11. Output declarations (**2.2.4 Serialization**).
12. Annotations (**4.15 Annotations**).
13. Function assertions in function tests.
14. A string concatenation operator (**3.6 String Concatenation Expressions**).
15. A mapping operator (**3.17 Simple map operator (!)**).