Bases de données et web – CM6

L3 Informatique & L3 MIASHS

XQuery

XQuery est un langage de requête pour le XML promu par la W3C : `https://www.w3.org/XML/Query/`. D'une certaine manière, XQuery est l'équivalent (en terme de requête) de SQL.

Comparaison avec XSLT :

- XSLT est un langage procédural performant pour la transformation de documents XML.
- XQuery est un langage déclaratif, performant pour effectuer des requêtes sur des bases de données relativement larges.

Remarque : dans beaucoup de cas, XSLT et XQuery peuvent être utilisés de manière interchangeable. Le choix entre l'un des deux est donc lié au contexte et/ou au goût.

Principes de XQuery

XQuery répond aux règles suivantes :

- ***Composition*** : XQuery se base sur les **expressions**, qui peuvent être composées afin de former une requête riche.
- ***Typage*** : XQuery peut être associé à un XML Schema afin d'interpréter les requêtes. Cependant, XQuery peut également fonctionner sur des documents sans XML Schema.
- ***Compatibilité avec XPath*** : XQuery est une extension de XPath 2.0. De ce fait, n'importe quelle expression XPath est également une expression XQuery.
- ***Analyse statique*** : inférence de type (recherche automatique des types), réécriture et optimisation : le moteur d'évaluation des requêtes XQuery joue sur la nature déclarative de XQuery afin d'optimiser les requêtes automatiquement.

D'un point de vue syntaxique, XQuery a pour but d'être concis et simple, à la manière de SQL.

Modèle de données

Une **valeur** est une **séquence** de 0 à *n* **objets**. Un **objet** est soit un nœud soit une valeur atomique. Il existe 7 types de nœuds :

- ***Document*** : nœud racine du document.
- ***Élément*** : nœud balise.
- ***Attribut*** : nœud attribut associé à un élément.
- ***Texte*** : nœud texte.
- ***Commentaire*** : nœud commentaire.
- ***Prédicat XML*** : il est possible avec XQuery d'accéder ou de générer un prologue XML. C'est un type de nœud spécial.
- ***Espace de noms*** : nœud espace de noms.

Le modèle est assez général : tout est une **séquence** d'**objets**. Cela est valable du cas le plus simple, par exemple un entier, au cas le plus complexe, une collection de documents XML.

Exemples de valeurs

Les exemples suivants sont des **valeurs** en XQuery :

- `47` : une séquence d'un seul objet (une valeur atomique).
- `</a>` : une séquence d'un seul objet (un nœud élément).
- `(1, 2, 3)` : une séquence de 3 valeurs atomiques.
- `(47, <a/>, "Hello")` : une séquence de 3 objets, chacun étant de type différent.
- `()` : la séquence vide.
- un document XML en entier.
- plusieurs documents XML (**collection** de document XML).

Les séquences en détails

Il n'y a aucune distinction entre un objet et une séquence de taille 1. Tout est une séquence.

Une séquence ***ne peut pas*** être imbriquée dans une autre séquence.

La notion de valeur nulle n'existe pas dans le modèle de données d'XQuery. Une valeur est présente ou ne l'est pas.

Une séquence peut être vide.

Une séquence peut contenir des objets de types différents (voir exemple précédent).

Une séquence est ordonnée. Deux séquences avec le même ensemble d'objets mais ordonnées différemment, sont différentes.

Quelques aspects de syntaxe en XQuery

XQuery est un langage sensible à la casse. Les requêtes XQuery sont des compositions d'expressions. Une expression produit une valeur et ne modifie pas le contexte ni les variables.

Les commentaires sont inscrit comme suit : `(:Ceci est un commentaire:)`.

Le Hello World en XQuery est simple, il suffit d'inscrire la requête suivante : `'Hello World'`

Ceci est en fait interprété comme l'expression retournant la valeur *Hello World*, qui est ici une chaîne de caractères.

Un autre exemple d'expression simple : `1+1` retournera 2.

Les expressions XQuery

Une expression XQuery prend une valeur (une séquence d'objets) et retourne une valeur.
Une expression peut prendre plusieurs formes :

- une expression décrivant un chemin (XPath);
- un constructeur;
- une expression ***FLWOR***;
- une condition;
- une expression à base de quantificateurs;
- une expression sur les types de données;
- des fonctions.

Quelques expressions simples :

- ***Des littéraux*** : `"Hello"`, `47`, `4.7`, `4.7E+2`.
- ***Des valeurs construites*** : `date('2008-03-15')`, `true()`, `false()`.
- ***Des variables*** : `$x`.
- ***Des séquences construites*** : `(1, (2, 3), (), (4, 5))`, équivalent à `(1, 2, 3, 4, 5)`.

Un document XML est également une expression. Le résultat de l'expression est l'expression elle même.

Accéder à des documents et collections

Une requête prend en général en entrée un ou plusieurs documents XML. XQuery identifie son entrée avec les fonctions suivantes :

- doc() : prend en paramètre l'URI du document XML et retourne le document XML.
- collection() : prend en paramètre une URI et retourne tous les nœuds document XML contenu dans le fichier pointé sous forme de séquence.

Par exemple, la requête suivante va retourner tous les clubs du fichier XML `championnat.xml` :

```
doc('championnat.xml')//club
```

Remarque : la fonction doc() retourne le nœud document du fichier XML (avec le prologue donc) et est de type **Document**.

XPath et XQuery

N'importe quelle expression XPath est une requête XQuery. Par exemple, pour afficher la journée numéro 4 de `championnat.xml` :

```
doc('championnat.xml')//journee[@num=4]
```

Le résultat de cette requête affiche tout le sous-arbre enraciné à la journée numéro 4 du fichier XML.

Remarque : si on récupère une collection de fichiers XML grâce à la fonction `collection()` à la place de `doc()`, l'expression XPath est évaluée pour chaque objet de la séquence retournée par la fonction `collection()`.

Les constructeurs

XQuery permet la construction de nouveau éléments dont le contenu peut être un mélange de balises, valeurs littérales et résultats d'expression XQuery.

Par exemple, requête suivante :

```
1 <test>
2    {doc('championnat.xml')//journee[@num=4]/date/text()}
3 </test>
```

a pour résultat : <test>2013−08−31</test>.

Une expression peut être utilisée à n'importe quel niveau d'une requête, et un constructeur peut contenir des expressions.

Remarque : une expression doit être entourée d'accolades afin d'être reconnue et exécutée lors du lancement de la requête.

Sans les accolades, le résultat est :
<test>doc('championnat.xml')//journee[@num=4]/date/text()</test>

Autres exemples de constructeurs :

`<chapitre ref='[{1 to 5, 7, 9}]'>` est équivalent à
`<chapitre ref='[1 2 3 4 5 7 9]'>`

Sans les accolades, le résultat est différent. L'expression n'est pas interprétée et est donc lue comme une simple chaîne de caractères.

`<chapitre ref='[1 to 5, 7, 9]'>` est équivalent à
`<chapitre ref='[1 to 5, 7, 9]'>`

On peut également spécifier des détails lors de la création d'un élément, comme par exemple ci-dessous en indiquant la présence d'un attribut avec sa valeur.

`<article>{$monArticle/@id}</article>` va créer un élément de la forme :
`<article id="271"></article>`

Les variables

Les variables sont des noms faisant référence à une valeur, comme classiquement en programmation. Elles peuvent être utilisées dans n'importe quelle expression dans son bloc de code.

```
1  declare variable $id := 3;
2  declare variable $nom := "Jean";
3  declare variable $travail := <travail>Astronaute</travail>;
4  declare variable $base := 5000;
5
6  <employe id="{$id}">
7      <nom>{$nom}</nom>
8      {$travail}
9      <salaire>{$base + 1000}</salaire>
10 </employe>
```

a pour résultat :

<employe id="3"><nom>Jean</nom><travail>Astronaute</travail><salaire>6000</salaire></employe>

Les expressions FLWOR

Les expressions ***FLWOR*** sont les expressions les plus puissantes en XQuery. Une expression FLWOR (prononcé *flower*) est une expression composée des éléments suivants :

- itération sur des séquences (***F***or);
- définition et initialisation de variables (***L***et);
- application de prédicats (***W***here);
- ordonnancement du résultat (***O***rder);
- construction du résultat (***R***eturn).

```
1  for $journee in doc('championnat.xml')//journee
2  let $date := $journee/date/text()
3  where $journee/@num < 5
4  order by $date descending
5  return
6  <journee>
7     {$journee/@num}
8  </journee>
```

a pour résultat :

<journee num="4"/><journee num="3"/><journee num="2"/><journee num="1"/>

for et let

Ces deux clauses permettent de définir et initialiser des variables. Cependant :

- ***for*** définit successivement la variable pour chaque objet de la séquence.
- ***let*** définit la variable comme étant égale à toute la séquence. Par exemple `let $x := //journees` affecte le sous-arbre `journees` à la variable x. Il est également possible de placer le let avant le for.

Remarque : le for peut itérer sur une séquence hétérogène, par exemple : `for $a doc('championnat.xml')//*` associe à chaque tour de boucle un élément du document. Tous les éléments du fichier vont être parcouru.

Le traitement d'un tel for devra donc être adapté au fait que la variable de boucle ne soit pas toujours le même élément. On peut par exemple ajouter un where demandant la présence d'une balise pour être certain de traiter un élément attendu.

for et return

La combinaison de ***for*** et ***return*** définit une expression : le for définit la séquence en entrée et le return définit la séquence en sortie.
Une simple boucle :

```
1  for $i in (1 to 10) return $i
```

Une boucle imbriquée :

```
1  for $i in (1 to 10) return
2      for $j in (1 to 2) return $i * $j
```

Équivalent syntaxique :

```
1  for $i in (1 to 10),
2      for $j in (1 to 2) return $i * $j
```

Combinaison de boucles :

```
1  for $i in (for $j in (1 to 10) return $j * 2)
2      return $i * 3
```

Remarque : le ***return*** est obligatoire dans une expression ***FLWOR***. La clause return est instanciée à chaque tour de boucle.

La clause where

La clause ***where*** est similaire au where en SQL. La différence étant dans le fait qu'une structure XML est plus flexible qu'une base SQL. Par exemple pour trouver l'identifiant de tous les clubs dont la ville est Lille :

```
1 for $c in doc('championnat.xml')/championnat/clubs/club
2 where $c/ville/text() = "Lille"
3 return $c/@id
```

Cela ressemble à une requête SQL, mais le prédicat where est interprété en suivant les règles XPath :

- si le chemin n'existe pas, le résultat du where est faux, et il n'y a pas d'erreur de type.
- si l'expression XPath retourne plusieurs nœuds, le resultat est vrai si il y a au moins un nœud pour lequel le where est vrai.

Jointure en XQuery

Il est possible de combiner les expressions FLWOR afin de réaliser des jointures facilement.

```
1  let $championnat := doc('championnat.xml')/championnat
2  for $club in $championnat/clubs/club
3      for $rencontre in $championnat//rencontre
4      where $club/@id = $rencontre/clubReceveur
5      return $rencontre
```

L'exemple ci-dessus effectue la jointure suivante : pour chaque club, on récupère la rencontre où celui-ci est club receveur et on l'affiche.

On peut exprimer la jointure comme un prédicat dans l'expression XPath du deuxième for ou dans la clause where comme dans cet exemple.

Opérations sur les listes

XQuery permet de manipuler les listes :

- concaténation;
- opérations ensemblistes (union, intersection, difference);
- quelques fonctions utilitaires (remove(), index-of(), count(), avg(), min(), max(), . . .).

Par exemple, pour récupérer le nombre de rencontres par journée :

```
1  for $journee in doc('championnat.xml')//journee
2  let $num := count($journee/rencontre)
3  return
4      <journee>
5          {$journee/@num}
6          <num_rencontre>{$num}</num_rencontre>
7      </journee>
```

les expressions if-then-else

La structure du ***if-then-else*** est standard. Exemple :

```
1  for $c in doc('championnat.xml')//club
2  return
3      <club>
4          {$c/nom,
5          if ($c/ville = "Lille") then $c/ville
6          else if ($c/ville = "Rennes") then "Rennes"
7          else "Autre ville"
8          }
9
10     </club>
```

Il est également possible d'imbriquer les if-then-else.

Les quantificateurs

En XQuery on peut utiliser les quantificateurs **some** et **every**, l'équivalent en français de **il existe** et **pour tout**. Ces quantificateurs sont utilisés en conjonction avec le mot clef **satisfies** permettant de spécifier ce que l'on cherche.

Par exemple pour retourner les journées possédant une rencontre dont le club receveur est GB.

```
1 for $j in doc('championnat.xml')//journee
2 where some $r in $j//rencontre satisfies $r/clubReceveur = "GB"
3 return $j
```

L'exemple suivant retourne toutes les journées pour lesquelles chacune des rencontres a un score nul (en club receveur ou club invité).

```
1 for $j in doc('championnat.xml')//journee
2 where every $r in $j//rencontre satisfies contains($r/score, "0")
3 return $j
```

Les fonctions

Exemple de fonctions effectuant la somme de deux entiers :

```
1  declare function local:somme($x as xs:integer, $y as xs:integer) as xs:integer
2  {
3      let $r := $x + $y
4      return $r
5  };
6
7  local:somme(1, 2)
```

Les fonctions ont des propriétés similaires à celles des autres langages de programmations. Notamment, par exemple, la récursion est possible. De plus, n'importe quel type de donnée XQuery peut être utilisé comme argument de fonction ou comme valeur de retour.

Quizz

Pour chaque tome de la collection donner son numéro, le nom de sa série et son titre sous forme d'attributs.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <collection>
3    <serie nom="Lanfeust de Troy">
4      <tome numero="1">
5        <scenariste>Arleston</scenariste>
6        <dessinateur>Tarquin</dessinateur>
7        <titre>L&apos;ivoire du Magohamoth</titre>
8      </tome>
9      <tome numero="2">
10       <dessinateur>Tarquin</dessinateur>
11       <titre>Thanos l&apos;incongru</titre>
12     </tome>
13     <editeur nom="Soleil production"/>
14   </serie>
15   <serie nom="Calvin &amp; Hobbes">
16     <tome numero="1">
17       <titre>Adieu, monde cruel</titre>
18     </tome>
19   </serie>
20   <serie nom="Léonard">
21     <tome numero="23">
22       <scenariste>De groot</scenariste>
23       <dessinateur>Turk</dessinateur>
24       <titre>Poil au génie</titre>
25     </tome>
26     <editeur nom="Le lombard"/>
27   </serie>
28 </collection>
```

```
for $tome in doc("collection.xml")//tome
let $serie := doc("collection.xml")//serie[tome=$tome]
return
  <tome numero="{$tome/@numero}" serie="{$serie/@nom}" titre="{$tome/titre/text()}"/>
```

Quizz

Pour chaque série, créez un élément série avec un attribut titre contenant le nom de la série et un attribut éditeur contenant le nom de son éditeur (la valeur de l'attribut editeur est "" s'il n'y a pas d'éditeur pour la série).

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <collection>
3    <serie nom="A">
4      <tome numero="1">
5        <titre>Alpha</titre>
6      </tome>
7      <tome numero="2">
8        <titre>Beta</titre>
9      </tome>
10   </serie>
11   <serie nom="B">
12     <tome numero="5">
13       <titre>Delta</titre>
14     </tome>
15     <tome numero="4">
16       <titre>Gamma</titre>
17     </tome>
18   </serie>
19   <serie nom="C">
20     <tome numero="1">
21       <titre>Epsilon</titre>
22     </tome>
23   </serie>
24 </collection>
```

```
for $serie in doc("collection.xml")//serie
return
        <serie
        titre="{$serie/@nom}"
        editeur="{$serie/editeur/@nom}"
        />
```

Python – XML

Utilisation de python et de l'API **ElementTree**.

ElementTree :

- API XML en python.
- Permet de parser un document XML.
- Facile à manipuler.

Cours extraits de `https://docs.python.org`

Document XML utilisé

```xml
<?xml version="1.0" encoding="utf-8"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

Lecture du document et récupération de la racine

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  # Importation de l'API.
5  import xml.etree.ElementTree as ET
6
7  # Lecture du fichier XML. L'arbre est stocké dans tree.
8  tree = ET.parse('country_data.xml')
9
10 # Récupération de la racine de l'arbre XML.
11 root = tree.getroot()
```

Lecture du document et récupération de la racine

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  # Importation de l'API.
5  import xml.etree.ElementTree as ET
6
7  # Lecture du fichier XML. L'arbre est stocké dans tree.
8  tree = ET.parse('country_data.xml')
9
10 # Récupération de la racine de l'arbre XML.
11 root = tree.getroot()
```

Récupération de la balise et des attributs

```
1  >>> root.tag
2  'data'
3
4  >>> root.attrib
5  {}
```

Lecture du document et récupération de la racine

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Importation de l'API.
import xml.etree.ElementTree as ET

# Lecture du fichier XML. L'arbre est stocké dans tree.
tree = ET.parse('country_data.xml')

# Récupération de la racine de l'arbre XML.
root = tree.getroot()
```

Récupération de la balise et des attributs

```
>>> root.tag
'data'

>>> root.attrib
{}
```

Itérer sur les nœuds enfants

```
# Itération sur les enfants d'un noeud.
>>> for child in root:
...     print child.tag, child.attrib
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

```
# On peut également accéder aux nœuds enfants via '[ ]'.
>>> root[0].attrib['name']
'Liechtenstein'

>>> root[1].attrib['name']
'Singapore'
```

Remarque : les attributs sont stockés sous forme de dictionnaire.

```
# Exemple de dictionnaire.
>>> print dict
{'attr1': '1', 'attr2': '2'}

>>> dict['attr1']
'1'

>>> dict['attr2']
'2'
```

Trouver des éléments et des attributs

```
1  # Itération sur tous les éléments correspondant
2  # à une balise donnée.
3  >>> for country in root.findall('country'):
4  ...     rank = country.find('rank').text
5  ...     name = country.get('name')
6  ...     print name, rank
7  Liechtenstein 1
8  Singapore 4
9  Panama 68
```

Trouver des éléments et des attributs

```
1  # Itération sur tous les éléments correspondant
2  # à une balise donnée.
3  >>> for country in root.findall('country'):
4  ...     rank = country.find('rank').text
5  ...     name = country.get('name')
6  ...     print name, rank
7  Liechtenstein 1
8  Singapore 4
9  Panama 68
```

Remarque : la fonction ***find*** retourne un élément. La fonction ***get*** retourne la valeur d'un attribut. On accède à la valeur d'un élément avec ***.text***.

Création de document XML

```
1  #!/usr/bin/env python
2  # -*- coding: utf-8 -*-
3
4  # Importation de l'API.
5  import xml.etree.ElementTree as ET
6
7  # Création de la racine.
8  root = ET.Element("root")
9
10 # Ajout d'un noeud enfant de la racine.
11 doc = ET.SubElement(root, "doc")
12
13 ET.SubElement(doc, "field1", attr1="blah", attr2="2").text = "some value1"
14 ET.SubElement(doc, "field2", attr1="asdfasd").text = "some value2"
15
16 tree = ET.ElementTree(root)
17 tree.write("filename.xml")
```

Création de document XML

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Importation de l'API.
import xml.etree.ElementTree as ET

# Création de la racine.
root = ET.Element("root")

# Ajout d'un noeud enfant de la racine.
doc = ET.SubElement(root, "doc")

ET.SubElement(doc, "field1", attr1="blah", attr2="2").text = "some value1"
ET.SubElement(doc, "field2", attr1="asdfasd").text = "some value2"

tree = ET.ElementTree(root)
tree.write("filename.xml")
```

```xml
<?xml version="1.0" encoding="utf-8"?>

<root>
    <doc>
        <field1 attr1="blah" attr2="2">some value1</field1>
        <field2 attr1="asdfasd">some value2</field2>
    </doc>
</root>
```

Fonction

```
1 # Définition d'une fonction prenant deux paramètres x et y et retournant la somme.
2 def somme(x, y):
3     result = x + y
4     return result
5
6 # Appel de fonction.
7 somme(1, 2)
```

Fonction

```
1 # Définition d'une fonction prenant deux paramètres x et y et retournant la somme.
2 def somme(x, y):
3     result = x + y
4     return result
5
6 # Appel de fonction.
7 somme(1, 2)
```

Remarque : l'indentation est **importante** en Python. C'est ce qui délimite un bloque de code.

Fonction

```
1  # Définition d'une fonction prenant deux paramètres x et y et retournant la somme.
2  def somme(x, y):
3      result = x + y
4      return result
5
6  # Appel de fonction.
7  somme(1, 2)
```

Remarque : l'indentation est **importante** en Python. C'est ce qui délimite un bloque de code.

Liste

```
1  # Création d'une liste vide.
2  myList = []
3
4  # Ajout d'un élément dans une liste.
5  myList.append('element')
6
7  # Itération sur une liste.
8  for myElement in myList:
9      print myElement
```

Boucle

```
1  # Boucle for.
2  for i in range(1, 10)
3      print i
```

Boucle

```
1  # Boucle for.
2  for i in range(1, 10)
3      print i
```

Remarque : la boucle ci-dessous affichera 1, 2, 3, 4, 5, 6, 7, 8, 9.

Boucle

```
1 # Boucle for.
2 for i in range(1, 10)
3     print i
```

Remarque : la boucle ci-dessous affichera 1, 2, 3, 4, 5, 6, 7, 8, 9.

Conditions

```
1 # Conditions.
2 if a == b:
3     print a
4 elif a > 3:
5     print b
6 elif a < 2 and b > 3:
7     print a, b
8 else:
9     print a+b
```

Exercice – Validation du championnat

```xml
<?xml version="1.0" encoding="utf-8"?>
<championnat>
  <clubs>
    <club id="LOSC">
      <nom>LOSC Lille Metropole</nom>
      <ville>Lille</ville>
    </club>
    <club id="SRFC">
      <nom>Stade Rennais FC</nom>
      <ville>Rennes</ville>
    </club>
  </clubs>
  <journees>
    <journee num="1">
      <date>2013-08-10</date>
      <rencontre>
        <clubReceveur>ACA</clubReceveur>
        <clubInvite>ASSE</clubInvite>
        <score>0 1</score>
      </rencontre>
      <rencontre>
        <clubReceveur>GB</clubReceveur>
        <clubInvite>ASMFC</clubInvite>
        <score>0 2</score>
      </rencontre>
    </journee>
```

Exercice

Vérifier que les clubs se rencontrent tous deux fois. Une fois à domicile et une fois à l'extérieur. Afficher les couples de clubs manquants.

```python
#!/bin/python
# -*- coding: utf-8 -*-

# Importation de l'API.
import xml.etree.ElementTree

# Chemin vers le fichier xml.
FILE = 'championnat.xml'

# Création du parseur.
document = xml.etree.ElementTree.parse(FILE)
p = document.getroot()

# Retourne la liste des clubs.
def get_list_club(parser):
    # La liste des IDs que l'on va mettre à jour au fur et à mesure.
    l = []

    # On va itérer sur tous les clubs.
    for c in parser.findall('clubs/club'):

        # On ajoute l'id trouvé dans la liste.
        l.append(c.attrib['id'])

    return l
```

```
26
27  # Cette fonction retourne vrai si la paire ordonnée (id1, id2) est présente
28  # dans une rencontre.
29  def is_pair_in_rencontres(id1, id2, parser):
30
31      # On itère sur toutes les rencontres.
32      for r in parser.findall('journees/journee/rencontre'):
33
34          # On récupère l'id du club receveur.
35          current_id1 = r.find('clubReceveur').text
36
37          # On récupère l'id du club invite.
38          current_id2 = r.find('clubInvite').text
39
40          # Si la paire ordonnée passée en paramètre est la même
41          # que celle qu'on vient de trouver, on retourne vraie.
42          if id1 == current_id1 and id2 == current_id2:
43              return True
```

```
44
45      # Si on arrive jusqu'ici, c'est que l'on a pas trouvé de paire
46      # correspondante. On retourne alors faux.
47      return False
48
49  def check_every_pair(list_ids, parser):
50      for id1 in list_ids:
51          for id2 in list_ids:
52              if id1 != id2:
53                  if not is_pair_in_rencontres(id1, id2, parser):
54                      print('Paire non présente : ' + id1 + ', ' + id2)
55
56
57  l = get_list_club(p)
58  check_every_pair(l, p)
```

Exercice

Recopier championnat.xml en enlevant le score.