

# Documents structurés

## Cours 6

---

Nassim ZELLAL

# XQuery

# Définitions

- **XQuery** est un langage de requête permettant d'extraire des informations d'un document XML, ou d'une collection de documents XML et de reconstruire de nouveaux documents XML.
- **BaseX** est un système de gestion de base de données XML permettant de stocker, d'interroger et de gérer des données au format XML de manière efficace.

# Expressions XQuery

- Expressions simples.
- Expressions complexes.

# Expressions simples

- Une expression simple :
  - ❑ valeurs atomiques ;
  - ❑ valeurs construites ;
  - ❑ variables.

## Valeurs atomiques

- Une valeur atomique est une instance d'un type atomique (types de données XQuery selon la recommandation/spécification XML Schema du W3C).
- Exemples :
- "Bonjour" → xs:string
- 1.5 → xs:decimal (tout nombre à virgule est vu par XQuery comme « xs:decimal ». Il est possible de convertir 1.5 en « xs:float »).
- 15 → xs:integer (15 est également vu comme un « xs:decimal »).
- "2008-12-01" → xs:date("2008-12-01") → xs:date
- "true" / "false" → xs:boolean("true") → xs:boolean
- **Remarque : Le préfixe « xs: » signifie XML Schema.**

## Valeurs construites

- Les valeurs construites sont obtenues par des fonctions :
- -----fonctions **constructeur**-----
- xs:date()
- xs:boolean()
- xs:integer() / xs:int()
- xs:string()
- xs:float()
- xs:double()
- -----fonctions **booléennes**-----
- fn:true() → true → 0 argument
- fn:false() → false → 0 argument
- -----fonctions de **conversion**-----
- fn:boolean()
- fn:string()
- fn:number() / il n'existe pas de fonction **constructeur**  
**xs:number()**

# Séquences

- Une séquence est une suite hétérogène d'items (nœuds et/ou valeurs atomiques). Elle s'écrit en utilisant des parenthèses et en séparant les items par une ",".
- La longueur d'une séquence correspond au nombre d'items qu'elle comporte. La séquence de longueur 0 est appelée "séquence vide" → ()
- Il ne peut pas y avoir de séquence contenue dans une séquence (pas d'imbrication de séquence). **La séquence** (39,(1,2), "table", <table/>) **est équivalente à** la séquence (39,1,2, "table", <table/>). L'imbrication est donc inutile.
- Une séquence est **immuable**.
- Exemples :
- Une séquence peut contenir des valeurs hétérogènes : (39, "table",<v/> )
- L'opérateur « **to** » permet de construire une séquence composée des entiers présents entre les deux bornes : 1 **to** 5



# Fonctions et séquences testées sur BaseX

- `xs:date("2012-01-01")`
  - `(:xs:boolean(1)`
  - `xs:boolean(0)`
  - `xs:integer("200")`
  - `xs:int("200")`
  - `xs:string(33)`
  - `xs:float(15.5)`
  - `xs:double(15.5)`
  - `fn:true()`
  - `fn:false()`
  - `(39, "table", <v/>)`
  - `(39, "table", <v/>) [1]`
- 
- `1 to 5 :)`

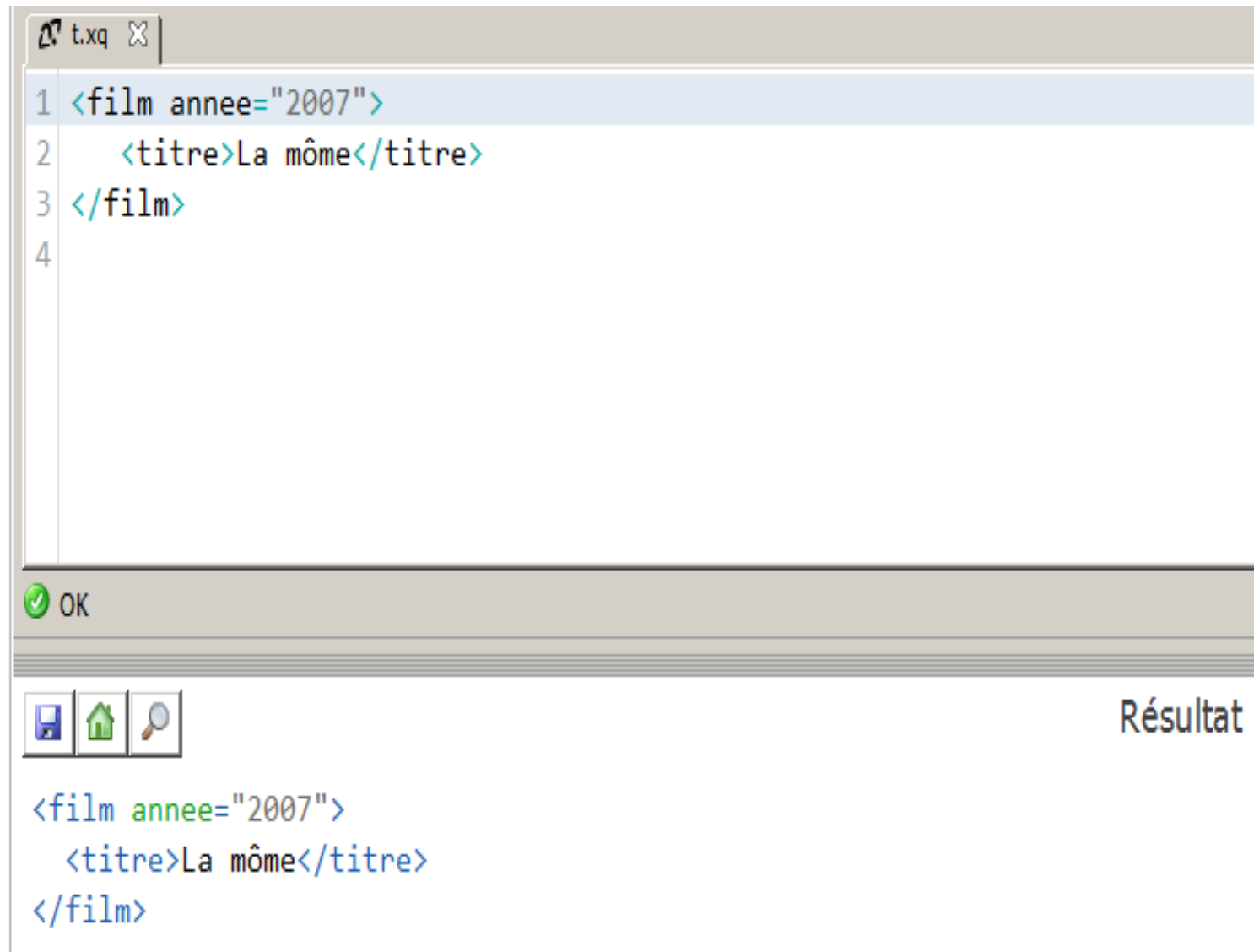
# Littéraux atomiques

- Littéraux de chaîne:
- "Bonjour tout le monde"
- Littéraux de numériques:
- 42,30.0
- Littéraux booléens:
- xs:boolean("true") / xs:boolean("false")
- Littéraux de date:
- xs:date("2024-12-03")
- Littéraux d'heure:
- xs:time("14:30:00")

# Littéraux de nœud

- Littéral de nœud :
- <film annee="2007">
- <titre>La même</titre>
- </film>

# Littéraux de nœud



The image shows a software interface with a text editor and a preview area. The text editor, titled 't.xq', contains the following XML code:

```
1 <film annee="2007">
2   <titre>La même</titre>
3 </film>
4
```

Below the editor is a status bar with a green checkmark and the text 'OK'. At the bottom, there is a toolbar with icons for saving, home, and search. To the right of the toolbar, the word 'Résultat' is displayed. The preview area shows the rendered XML structure:

```
<film annee="2007">
  <titre>La même</titre>
</film>
```

## La fonction doc()

- La fonction **doc("URI")** retourne le nœud racine (*root node*) du document identifié par un URI (Uniform Resource Identifier). Cette fonction renvoie un document (*document node - nœud de document / nœud document*).
- La fonction **doc("xpath-3.xml")** retourne donc le document XML.
- Exemple :
- **doc("xpath-3.xml")**
- Remarque : Si le document XML est déjà ouvert (administré) sur **BaseX**, il n'est pas nécessaire d'utiliser la fonction **doc()**, précisant le nom du fichier/document XML.

# La fonction doc()

Context: db:open("xp... Éditeur

t.xq

```
1 doc("xpath-3.xml")
```

OK 1 : 19

Resultat

Diagram illustrating the XML document structure (xpath-3.xml) and its corresponding DOM tree.

The XML document structure (xpath-3.xml) is shown as a tree diagram:

- Root: AAA
  - BBB
    - HHH
      - TTT
  - CCC
    - HHH
      - DDD
        - EEE
  - BBB
    - HHH
      - DDD
        - EEE
  - BCC
    - BCD
      - HHH
        - FFF
  - DDD
    - FFF
  - GGG
    - III
  - BBB
    - HHH
      - DDD
        - EEE
  - CCC
    - EEE

The DOM tree structure (xpath-3.xml) is shown as a table:

xpath-3.xml			
AAA			
BBB	CCC	BCC	
		BCD	
BBB		HHH	CDB
HHH	HHH		
HHH			
		DDD	
HHH	HHH	EEE	EFG
TTT		FFF	
GGG	III		CCC
BBB	JJJ	JJJ	DDD
	ddd		EEE
	gggg		DDD
			FFF
BBB			
DDD			
CCC			
DDD	EEE		

XML DOM Tree Structure (xpath-3.xml):

```
<AAA>  
  <BBB/>  
  <CCC/>  
  <BBB>  
    <HHH att="1"/>  
    <HHH att="2"/>  
    <HHH fromage="Gouda"/>  
    <HHH>  
      <TTT/>  
    </HHH>  
  <HHH/>  
</BBB>
```

# Variables

- Les variables ont un nom précédé du signe \$
- Une variable possède une valeur.
- Déclaration d'une variable **globale** avec « **declare variable** » :
- **declare variable** \$a := valeur;
- Exemples :
- **declare variable** \$a := "X";
- **declare variable** \$b := "Query";
- \$a,\$b,concat(\$a,\$b)
- #-----#
- **declare variable** \$s := ("X","Query");
- concat(\$s[1],\$s[2])#l'indexation d'une séquence commence à 1
- #-----#
- **declare variable** \$doc := doc('xpath-3.xml');
- **declare variable** \$a := <a/>;
- \$doc,\$a

# Variables

The screenshot shows a software interface for editing and executing XQuery code. At the top, a tab labeled 't.xq' is visible. The main text area contains the following code:

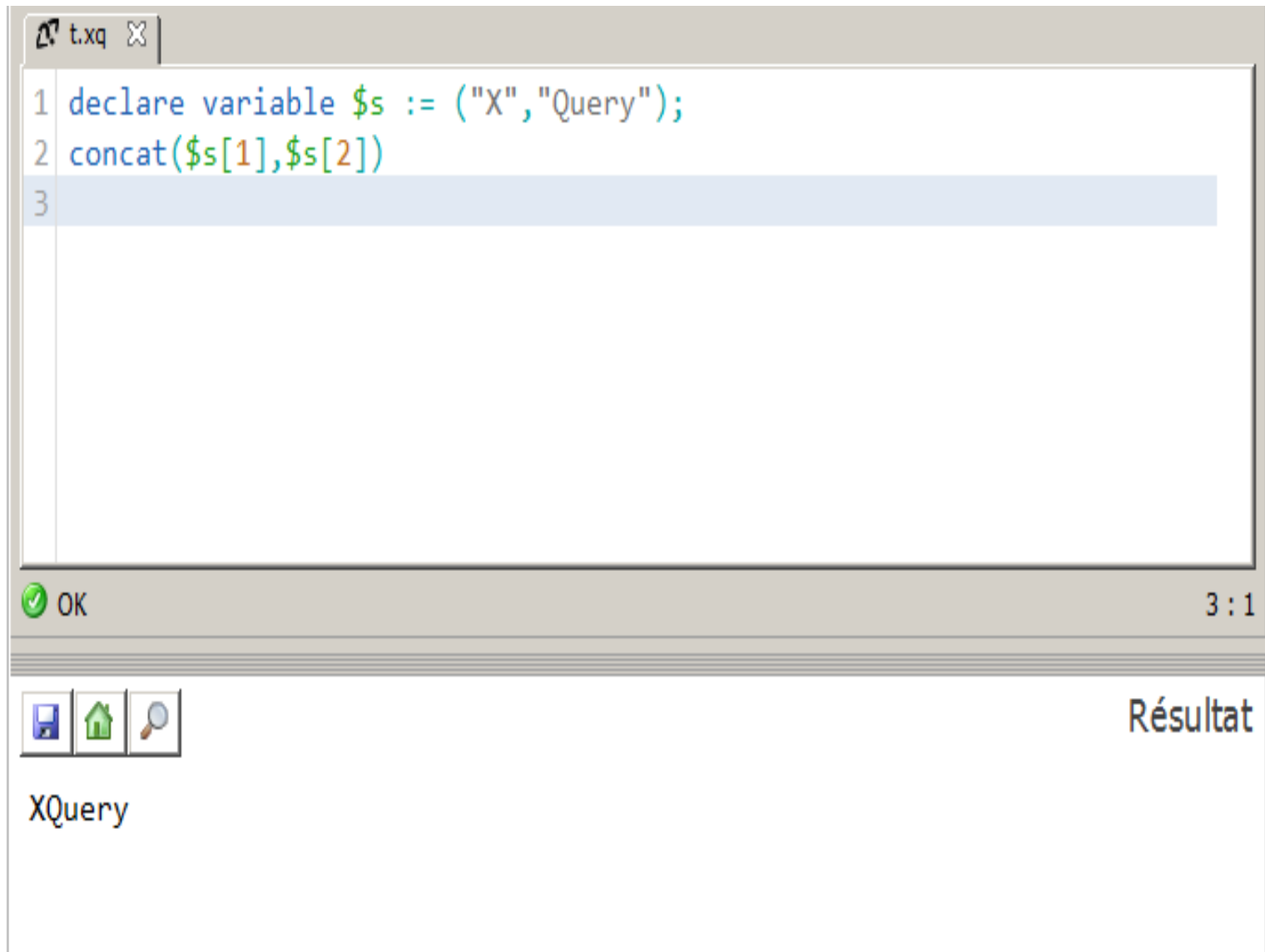
```
1 declare variable $a := "X";  
2 declare variable $b := "Query";  
3 $a,$b,concat($a,$b)  
4
```

Below the code editor, a status bar shows a green checkmark icon followed by 'OK' on the left and '4 : 1' on the right. At the bottom of the window, there is a toolbar with three icons: a floppy disk (save), a house (home), and a magnifying glass (search). To the right of the toolbar, the word 'Résultat' is displayed. Below these elements, the output of the query is shown as a list of three items:

- X
- Query
- XQuery



# Variables



The image shows a screenshot of an XQuery editor window. The window has a title bar with a file icon, the name 't.xq', and a close button. The main area is a text editor with three lines of code:   
1 declare variable \$s := ("X","Query");   
2 concat(\$s[1],\$s[2])   
3   
The third line is highlighted with a light blue background. Below the editor is a status bar with a green checkmark icon, the text 'OK', and the text '3 : 1'. At the bottom of the window is a toolbar with three icons: a folder, a house, and a magnifying glass. To the right of the toolbar is the word 'Résultat'. Below the toolbar is the text 'XQuery'.

```
1 declare variable $s := ("X","Query");  
2 concat($s[1],$s[2])  
3
```

OK 3 : 1

Résultat

XQuery

# Variables

```
1 declare variable $doc := doc('xpath-3.xml');  
2 declare variable $a:=<a/>;  
3 $doc,$a  
4
```

OK



```
<!--  
<DDD/>  
<EEE/>  
</CCC>  
</DDD>  
</BBB>  
<CCC>  
<DDD>  
<EEE>  
<DDD>  
<FFF/>  
</DDD>  
</EEE>  
</DDD>  
</CCC>  
</AAA>  
<a/>
```

# Variables

The screenshot shows a XQuery editor window titled 't.xq'. The code in the editor is as follows:

```
1 declare variable $doc := doc('xpath-3.xml');  
2 $doc  
3
```

Below the code editor, there is a status bar with a green checkmark and the text 'OK', and a zoom level indicator '1 : 1'.

The result pane, titled 'Résultat', displays the XML document loaded by the variable \$doc. The XML structure is as follows:

```
<AAA>  
  <BBB/>  
  <CCC/>  
  <BBB>  
    <HHH att="1"/>  
    <HHH att="2"/>  
    <HHH fromage="Gouda"/>  
    <HHH>  
      <TTT/>  
    </HHH>  
  <HHH/>  
</BBB>  
<BCC>  
  <BCD>  
    <HHH fromage="Cantal"/>  
    <CDB/>  
  </BCD>  
</BCC>  
<DDD>
```

## Variables et littéraux XML

- Dans l'exemple suivant, les accolades **{ }** indiquent ce qui doit être évalué.
- Exemple :
- **declare variable** \$m := (1,2);
- **<e>**
- **<o>{\$m[1]}</o>{\$m[2],\$m[1]}**
- **\$m[2] {\$m[2]}**
- **</e>**
- **\$m[2]** n'est pas évalué

# Variables et littéraux XML

```
1 declare variable $m := (1,2);  
2 <e>  
3     <o>{$m[1]}</o>{$m[2],$m[1]}  
4     $m[2] {$m[2]}  
5 </e>  
6
```

✓ OK



```
<e>  
  <o>1</o>2 1  
  $m[2] 2</e>
```

# Opérateurs

- Opérateurs arithmétiques : **+**, **-**, **\***, **div**, **mod**
- Opérateurs logiques/booléens : **and**, **or**, **not**
- Opérateurs de comparaison :
  - **eq**, **ne**, **lt**, **le**, **gt**, **ge** (comparaison de valeurs)
  - **=**, **!=**, **<**, **<=**, **>**, **>=** (comparaison générale)
  - **is**, **<<**, **>>** (comparaison de nœuds)

# Opérateurs de comparaison de valeurs

- Les opérateurs **eq**, **ne**, **lt**, **le**, **gt**, **ge** permettent de comparer des valeurs atomiques. La comparaison de valeurs avec ces opérateurs nécessite de prendre en compte le type générique « **xs:untypedAtomic** » (valeur atomique non typée et non validée par XML Schema) ainsi que la comparaison d'une séquence avec un item. Sans cela le processeur retourne une erreur.
- Exemples :
- 1861 **ne** 1860
- doc("arbres.xml")//arbre/annee **eq** 1860
- doc("arbres.xml")//arbre[@id="90"]/annee **eq** 1860
- xs:int(doc("arbres.xml")//arbre[@id="90"]/annee) **eq** 1860
- doc("books1.xml")//book/title **eq** "Data on the Web"
- doc("books1.xml")//book[3]/title **eq** "Data on the Web"
- doc("books1.xml")//book/title **eq** doc("books1.xml")//book[3]/title
- //a **eq** //b (ici on compare les valeurs contenues dans a et b)

# Opérateurs de comparaison de valeurs

```
1 doc("arbres.xml")//arbre[@id="90"]/annee eq 1860
```

```
2
```

Cc [...] \* Trouver...

✖ Types xs:untypedAtomic and xs:integer are not comparable.

t.xq

```
1 xs:int(doc("arbres.xml")//arbre[@id="90"]/annee) eq 1860
```

```
2
```

Cc [...] \* Trouver...

✓ OK



true

t.xq

```
1 doc("arbres.xml")//arbre/annee eq 1860
```

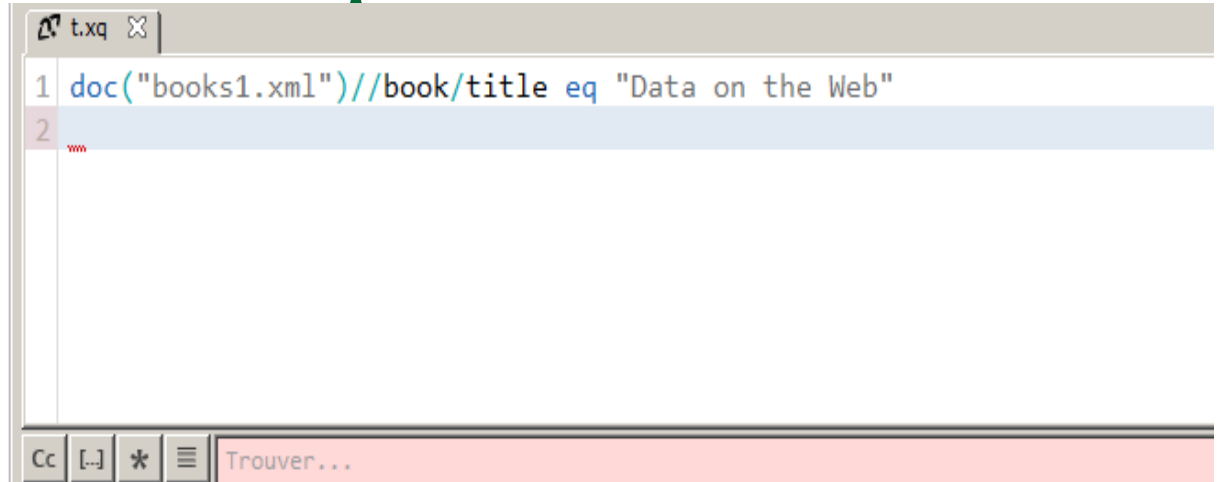
```
2
```

Cc [...] \* Trouver...

✖ Item expected, sequence found: ("1935", "1854", "1862", "1906", "1784", "1860", ...)



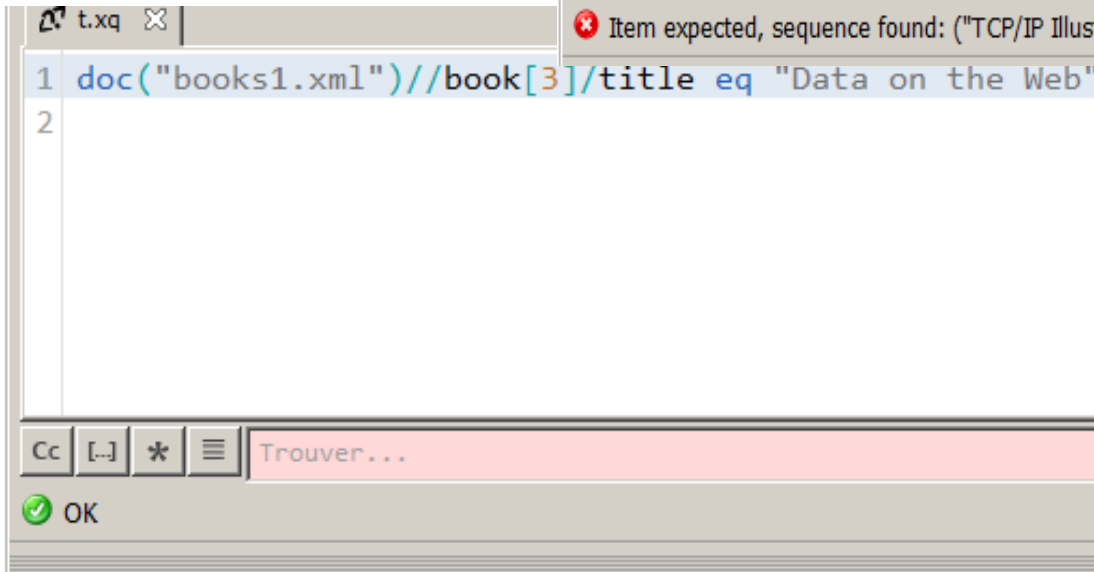
# Opérateurs de comparaison de valeurs



A screenshot of a t.xq editor window. The title bar shows 't.xq' and a close button. The editor contains two lines of code: line 1 is `doc("books1.xml")//book/title eq "Data on the Web"` and line 2 is empty. A red squiggly line under the `eq` operator indicates a syntax error. Below the editor is a toolbar with buttons for 'Cc', '[..]', '\*', and a menu icon, followed by a search bar containing the text 'Trouver...'. A status bar at the bottom shows a red error icon and the message: 'Item expected, sequence found: ("TCP/IP Illustrated", "Advanced Programming in the UNIX Environment", ...)'.

```
1 doc("books1.xml")//book/title eq "Data on the Web"
2
```

Item expected, sequence found: ("TCP/IP Illustrated", "Advanced Programming in the UNIX Environment", ...).



A screenshot of a t.xq editor window. The title bar shows 't.xq' and a close button. The editor contains two lines of code: line 1 is `doc("books1.xml")//book[3]/title eq "Data on the Web"` and line 2 is empty. Below the editor is a toolbar with buttons for 'Cc', '[..]', '\*', and a menu icon, followed by a search bar containing the text 'Trouver...'. A status bar at the bottom shows a green checkmark icon and the text 'OK'.

```
1 doc("books1.xml")//book[3]/title eq "Data on the Web"
2
```

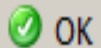


true

# Opérateurs de comparaison de valeurs

```
1 doc("books1.xml")//book[3]/title eq doc("books1.xml")//book[3]/title
```

```
2
```



OK



true

# Opérateurs de comparaison de valeurs

```
1 doc("books1.xml")//book/title eq doc("books1.xml")//book[3]/title
```

```
2
```

\*\*\*

❌ Item expected, sequence found: ("TCP/IP Illustrated", "Advanced Programming in the UNIX Environment",



# Opérateurs de comparaison de valeurs

1 //a eq //b		
Item expected, sequence found: ("5", "5").		
a-b.xml		
element		
a	b	a
5	5	5

# Opérateurs de comparaison de valeurs

The screenshot displays a software interface with a top toolbar containing icons for file operations (save, home, search) and a search bar labeled "Trouver...". Below the toolbar, the text "true" is displayed. At the bottom, a table with two columns, "a" and "b", shows the values "5" and "5" respectively, indicating a successful comparison.

a	b
5	5

## Opérateurs de comparaison générale

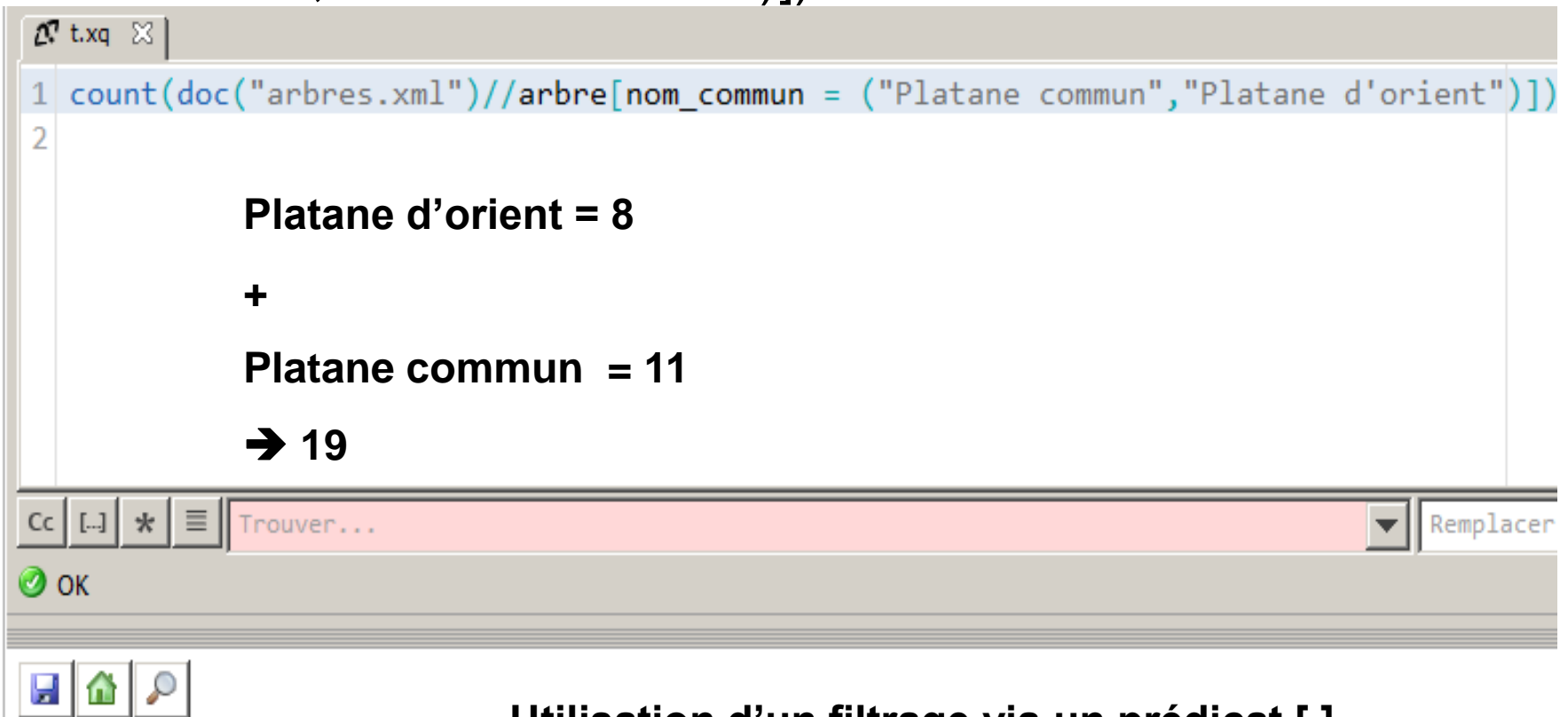
- Les opérateurs `=`, `!=`, `<`, `<=`, `>`, `>=` permettent d'effectuer une comparaison générale (valeurs atomiques, séquences, `xs:UntypedAtomic`, valeurs de nœuds).
  - Les requêtes XQuery suivantes vont retourner « true »
  - `doc("arbres.xml")//arbre/annee = 1860`
  - `doc("books1.xml")//book/title = "Data on the Web"`
  - `doc("arbres.xml")//arbre[@id="90"]/annee = 1860`  
(conversion automatique de l'opérande gauche en entier)
  - `doc("books1.xml")//book[3]/title = "Data on the Web"`
  - `//a = //b`
- 
- Tester aussi en remplaçant `=` par **eq**.

## Opérateurs de comparaison générale

- séquence1 = séquence 2 s'il existe un élément dans s1 égal à un élément dans s2.
- Exemples:
  - ("Platane commun","Platane") = ("Platane commun","Platane d'orient")
  - doc("arbres.xml")//nom\_commun = ("Platane commun","Platane d'orient")
- Résultat sur BaseX → true pour ces deux requêtes.

# Opérateurs de comparaison générale

- Exemple :
- `count(doc("arbres.xml")//arbre[nom_commun = ("Platane commun","Platane d'orient")])`



Utilisation d'un filtrage via un prédicat [ ]



# Opérateurs de comparaison générale

- $(39, (1, 2), \text{"table"}, \text{<table/>}) = (39, 1, 2, \text{"table"}, \text{<table/>})$
- Le résultat de cette comparaison est **true** (voir plus haut slide intitulé "Séquences").
- Dans cet exemple, l'opérateur de comparaison de valeurs **eq** retourne une erreur, car il n'est pas possible de comparer, via cet opérateur, une séquence avec une autre séquence.

## Opérateurs de comparaison de nœuds

- L'opérateur de comparaison **is** permet de comparer l'identité des nœuds et non leur valeur.
- $n1 \text{ **is** } n2 \Leftrightarrow n1 \text{ est identique à } n2$ .
- Pour illustrer ce type de comparaison, comparons un nœud à lui-même, en utilisant l'opérateur **is**.
- `doc("arbres.xml")//arbre[2] is doc("arbres.xml")//arbre[2]`
- `> true`
- Recopions notre document XML et renommons-le « arbres-.xml ».
- `doc("arbres-.xml")//arbre[2] is doc("arbres.xml")//arbre[2]`
- `> false`
- Les nœuds sont différents, car ils proviennent de deux documents différents.

## Opérateurs de comparaison de nœuds

- Voyons à présent un autre exemple de comparaison des deux nœuds « **a** » provenant du document « a-b.xml » suivant :
- `<element>`
- `<a>5</a>`
- `<b>5</b>`
- `<a>5</a>`
- `</element>`
- `doc("a-b.xml")//a[1] is doc("a-b.xml")//a[2]`
- Les deux nœuds sont différents, car chacun a sa propre identité. Et pourtant ils possèdent la même valeur : **5**
- `> false`

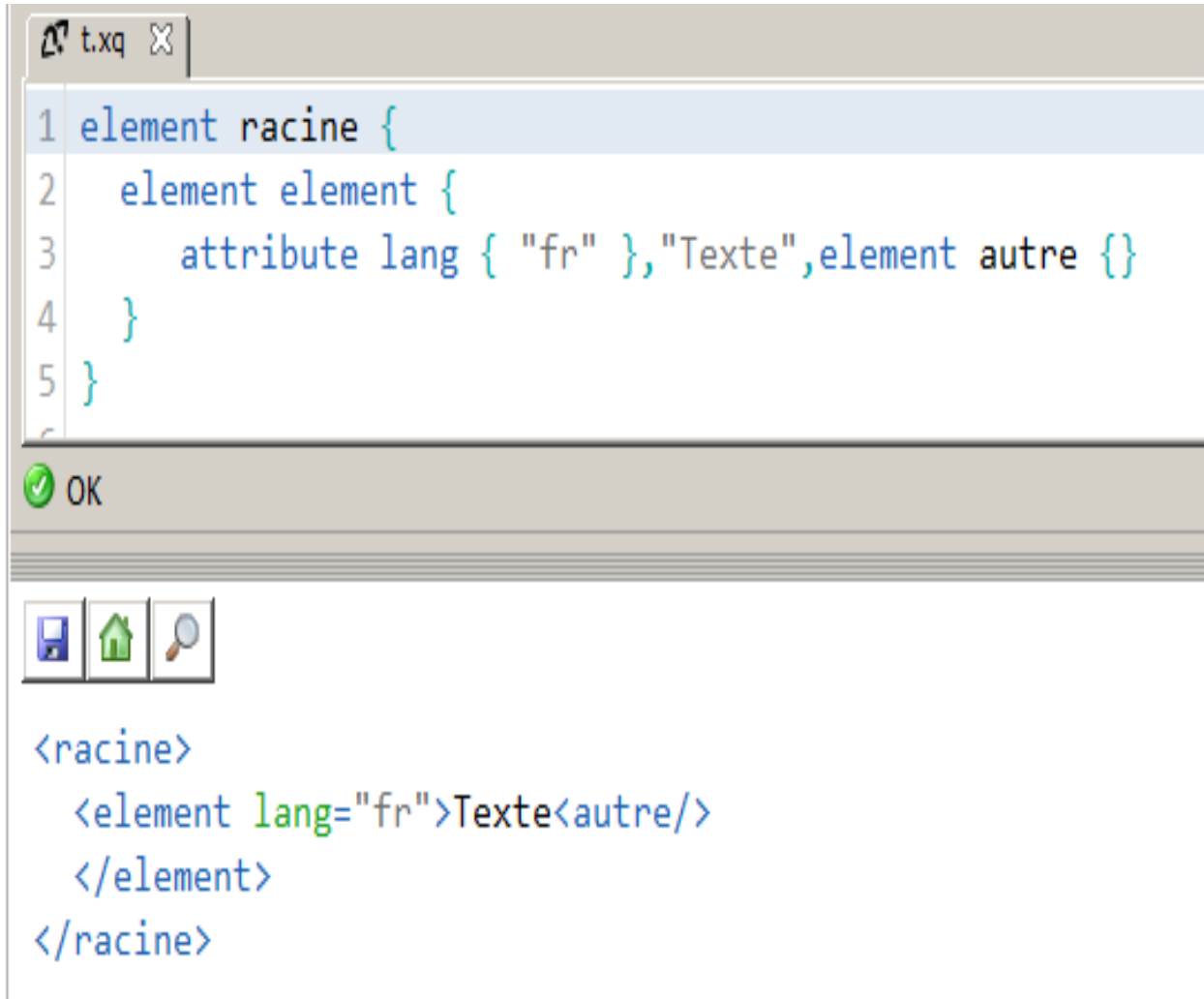
# Opérateurs de comparaison de nœuds

- Il est possible de comparer l'ordre d'apparition des nœuds, en utilisant les opérateurs **<<** et **>>**.
- Autrement dit, ces opérateurs permettent de comparer la **position** de deux nœuds.
- $n1 \text{ << } n2 \text{ (} n1 \text{ >> } n2) \Leftrightarrow n1 \text{ apparaît avant (après) } n2.$
- Exemple :
- `doc("a-b.xml")//a[1] << doc("a-b.xml")//a[2]`
- **> true**

# Construction de nœuds XML avec « element » et « attribute »

- Exemple :
- **element** racine {
- **element** element {
- **attribute** lang { "fr" } , "Texte", element autre { }
- }
- }
- Remarque : Les { } représentent un nœud vide.

# Construction de nœuds XML avec « element » et « attribute »



The screenshot shows a software interface with a code editor and a preview area. The code editor, titled 't.xq', contains the following XQuery code:

```
1 element racine {  
2   element element {  
3     attribute lang { "fr" }, "Texte", element autre {}  
4   }  
5 }
```

Below the code editor is a status bar with a green checkmark icon and the text 'OK'. At the bottom of the interface is a toolbar with three icons: a floppy disk (save), a house (home), and a magnifying glass (search). Below the toolbar, the resulting XML structure is displayed:

```
<racine>  
  <element lang="fr">Texte<autre/>  
  </element>  
</racine>
```

# Expressions complexes

- Les expressions **XQuery** complexes sont :
  - 1) Des expressions de chemin **XPath**.
  - 2) Des expressions **FLWOR**.

# Expressions de chemin XPath

- Une expression de chemin XPath est une requête XQuery.
- Un chemin localise et retourne un ensemble de nœuds.
- Exemples :
- **//HHH**
- **//\*[@\*="Cantal"]**



# Expressions de chemin XPath

Context: db:open("xpath-3")

Éditeur

t.xq

1 //HHH

OK

1 : 6

Résultat

xpath-3.xml

AAA

BBB

CCC

BCC

GGG

III

CCC

BCD

BBB

JJJ

JJJ

DDD

HHH

CDB

ddd

gggg

EEE

DDD

FFF

DDD

EEE

FFF

EFG

BBB

DDD

CCC

DDD

EEE

HHH

HHH

HHH

TTT

HHH

HHH

<HHH att="1"/>

<HHH att="2"/>

<HHH fromage="Gouda"/>

<HHH>

<TTT/>

</HHH>

<HHH/>

<HHH fromage="Cantal"/>

# Expressions de chemin XPath

Context: db:open("xpath-3")

Éditeur

txq

```
1 //*[@*="Cantal"]
2
```

OK

2 : 1

xpath-3.xml

AAA  
BBB CCC BCC GGG III CCC  
BCD BBB JJJ JJJ DDD  
HHH HHH CDB ddd EEE  
HHH HHH TTT gggg DDD  
DDD  
EEE FFF EFG BBB  
HHH HHH DDD CCC  
DDD EEE

Résultat

```
<HHH fromage="Cantal"/>
```

xpath-3.xml

```
graph TD
    AAA[AAA] --> BBB1[BBB]
    AAA --> CCC1[CCC]
    AAA --> BBB2[BBB]
    AAA --> BCC[BCC]
    AAA --> DDD1[DDD]
    AAA --> GGG[GGG]
    AAA --> III[III]
    AAA --> BBB3[BBB]
    AAA --> CCC2[CCC]
    BBB1 --> HHH1[HHH]
    BBB1 --> HHH2[HHH]
    BBB1 --> HHH3[HHH]
    BBB1 --> HHH4[HHH]
    BBB1 --> HHH5[HHH]
    CCC1 --> DDD2[DDD]
    CCC1 --> DDD3[DDD]
    DDD2 --> TTT[TTT]
    DDD2 --> HHH6[HHH]
    HHH6 --> gggg[gggg]
    HHH6 --> DDD4[DDD]
    DDD3 --> EEE1[EEE]
    DDD3 --> FFF1[FFF]
    CCC2 --> EEE2[EEE]
    CCC2 --> DDD5[DDD]
    DDD5 --> DDD6[DDD]
    DDD5 --> FFF2[FFF]
```

## Expressions de chemin XPath et fonction doc()

- Il est possible d'appliquer une expression XPath sur la fonction **doc("URI")**. Voir aussi plus haut les exemples illustrant les "Opérateurs".
- Exemple :
- **doc("xpath-3.xml")//HHH**

# Expressions de chemin XPath et fonction doc()

Context: db:open("xpath-3") Éditeur

t.xq\*

1 doc("xpath-3.xml")//HHH

OK 1 : 24

xpath-3.xml

Résultat

```
<HHH att="1"/>
<HHH att="2"/>
<HHH fromage="Gouda"/>
<HHH>
  <TTT/>
</HHH>
<HHH/>
<HHH fromage="Cantal"/>
```

# Littéraux XML et expressions de chemin XPath

- Exemples :
- `<resultat>`
- `<xq>{doc("xpath-3.xml")//HHH[@*]}</xq>`
- `</resultat>`
- `#-----#`
- `declare variable $m := (1,2);`
- `<e>`
- `<o>{$m[1],//CDB}</o>{$m[2],$m[1]}`
- `$m[2] {$m[2]}`
- `</e>`

# Littéraux XML et expressions de chemin XPath

```
t.xq
1 <resultat>
2 <xq>{doc("xpath-3.xml")//HHH[@*]}</xq>
3 </resultat>
4
```

Résultat

```
<resultat>
  <xq>
    <HHH att="1"/>
    <HHH att="2"/>
    <HHH fromage="Gouda"/>
    <HHH fromage="Cantal"/>
  </xq>
</resultat>
```

# Littéraux XML et expressions de chemin XPath

```
1 declare variable $m := (1,2);
2 <e>
3     <o>{$m[1],//CDB}</o>{$m[2],$m[1]}
4     $m[2] {$m[2]}
5 </e>
6
```

✓ OK

6 : 1



Résultat

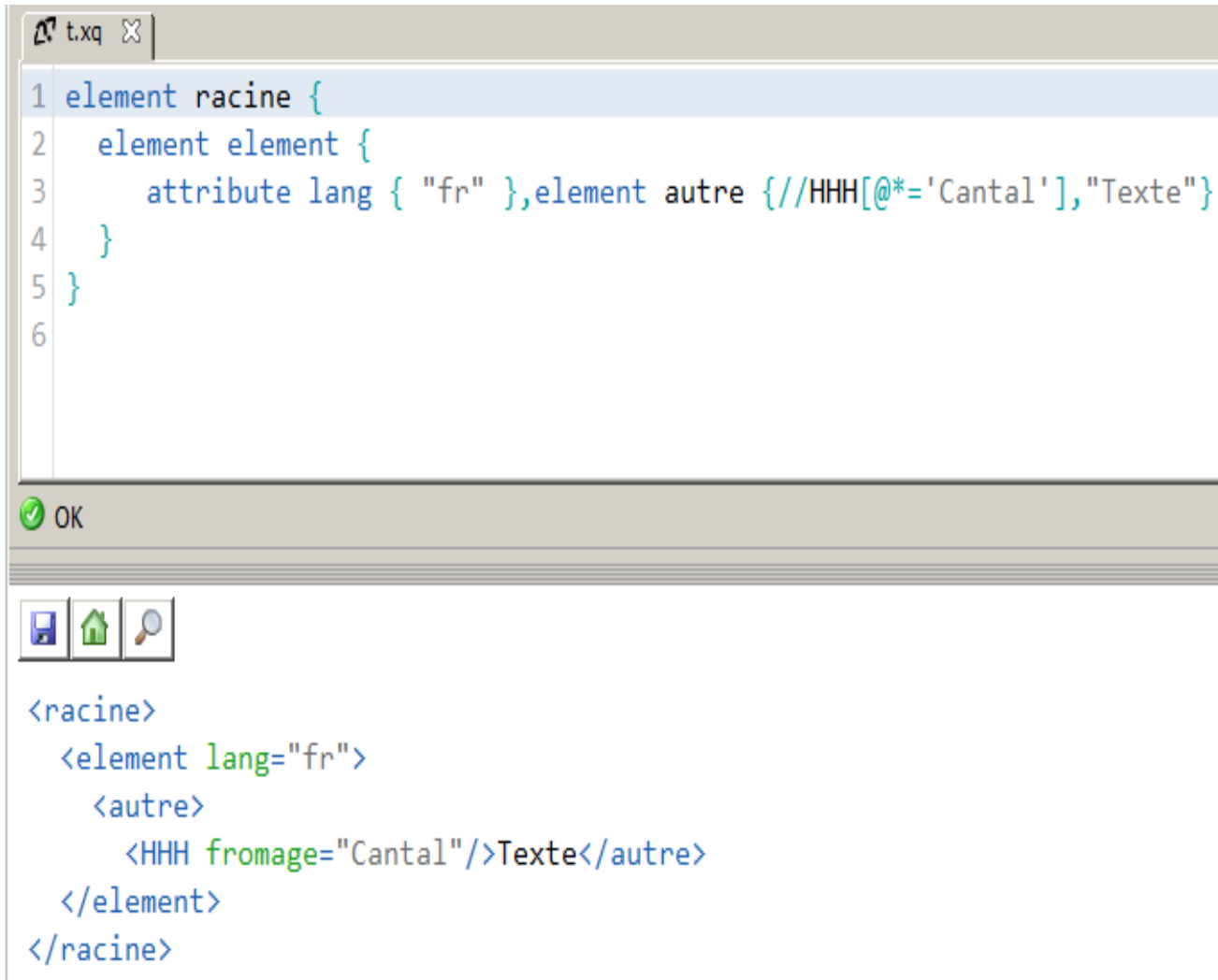
```
<e>
  <o>1<CDB/>
  </o>2 1
  $m[2] 2</e>
```

# Expressions de chemin XPath et construction de nœuds XML avec « element » et « attribute »

- Exemple :
- **element** racine {
- **element** element {
- **attribute** lang { "fr" },
- **element** autre { //HHH[@\*='Cantal'], "Texte" }
- }
- }



# Expressions de chemin XPath et construction de nœuds XML avec « element » et « attribute »



The screenshot shows a software interface with a text editor at the top and a preview area at the bottom. The text editor, titled 't.xq', contains the following XPath expression:

```
1 element racine {  
2   element element {  
3     attribute lang { "fr" }, element autre { //HHH[@*='Cantal'], "Texte" }  
4   }  
5 }  
6
```

Below the editor is a status bar with a green checkmark and the text 'OK'. At the bottom, there is a toolbar with icons for file operations and a preview area showing the resulting XML structure:

```
<racine>  
  <element lang="fr">  
    <autre>  
      <HHH fromage="Cantal"/>Texte</autre>  
    </element>  
</racine>
```

# Conditions - if-then-else

```
<?xml version="1.0"?>
<arbres>
  <arbre id="6">
    <genre>Maclura</genre>
    <espece>Pomifera</espece>
    <famille>Moraceae</famille>
    <nom_commun>Oranger des osages</nom_commun>
    <annee>1935</annee>
    <hauteur>13.0</hauteur>
    <lieu>
      <geopoint lat="48.857140829" lon="2.29533455314" />
      <nom>Parc du Champ de Mars</nom>
    </lieu>
  </arbre>
</arbres>
```

## Conditions - if-then-else

- declare variable \$arbre := //arbre;
- **if** (\$arbre/annee < 1950)
- **then** concat(\$arbre/nom\_commun, " est un arbre ancien")
- **else** concat(\$arbre/nom\_commun, " est un arbre récent")
- **> Oranger des osages est un arbre ancien**

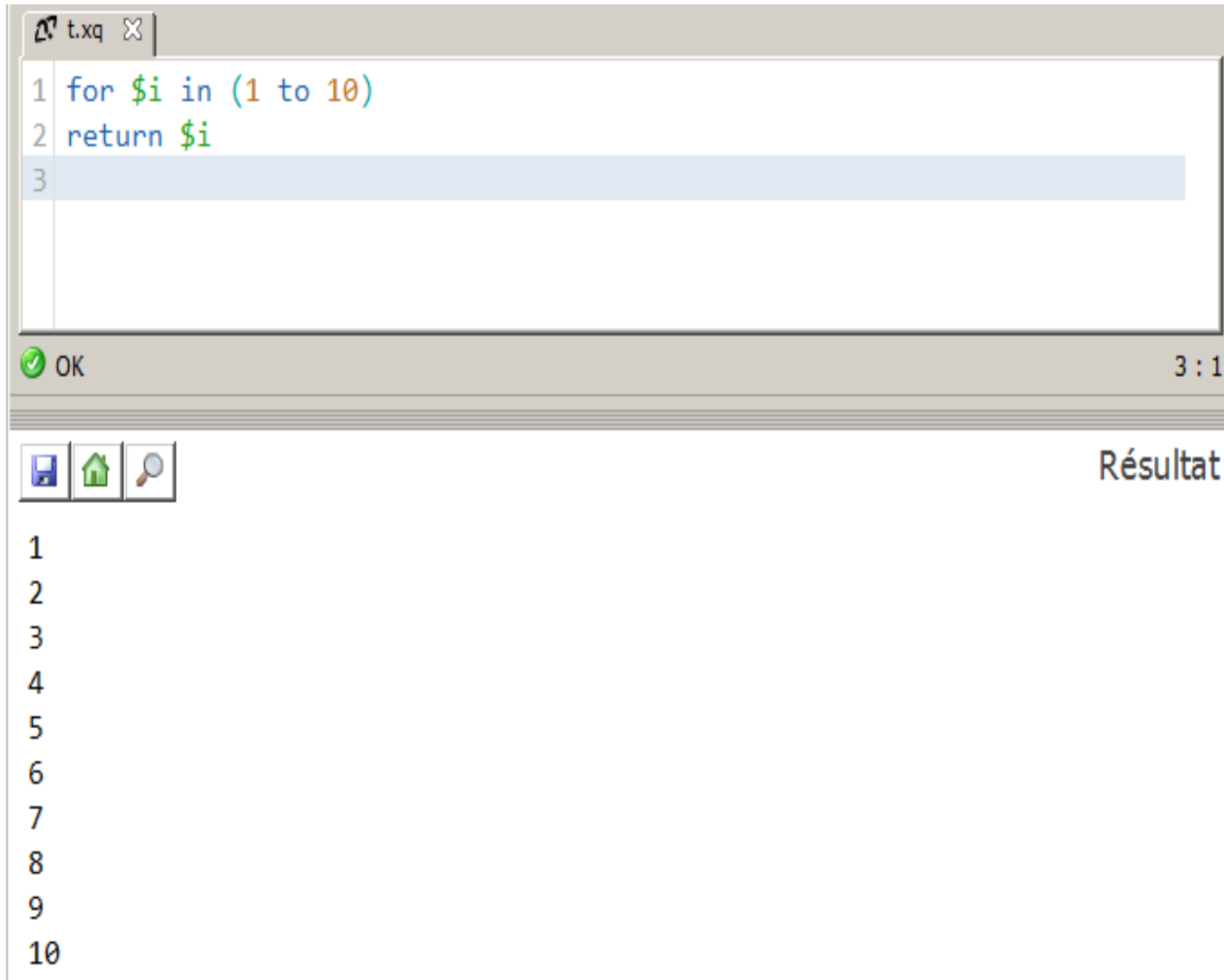
# Expressions FLWOR

- Une expression FLWOR prononcée "flower" :
- 1) **itère** sur des séquences (for) ;
- 2) **définit** des variables (let) ;
- 4) applique des **filtres** (where) ;
- 3) **trie** les résultats (order by) ;
- 5) **retourne** un résultat (return).

## Boucles - clause « for »

- La clause (ou instruction) « **for** » fait un parcours sur une collection. Le résultat sera retourné via la clause « **return** ».
  - Par exemple, une énumération d'entiers (inf to sup) grâce à l'opérateur « **to** », comme dans l'exemple suivant :
- 
- **for** \$i in (1 **to** 10)
  - **return** \$i

# Boucles - clause « for »



The image shows a screenshot of a TQL (Terminology Query Language) environment. At the top, a window titled 't.xq' contains the following code:

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

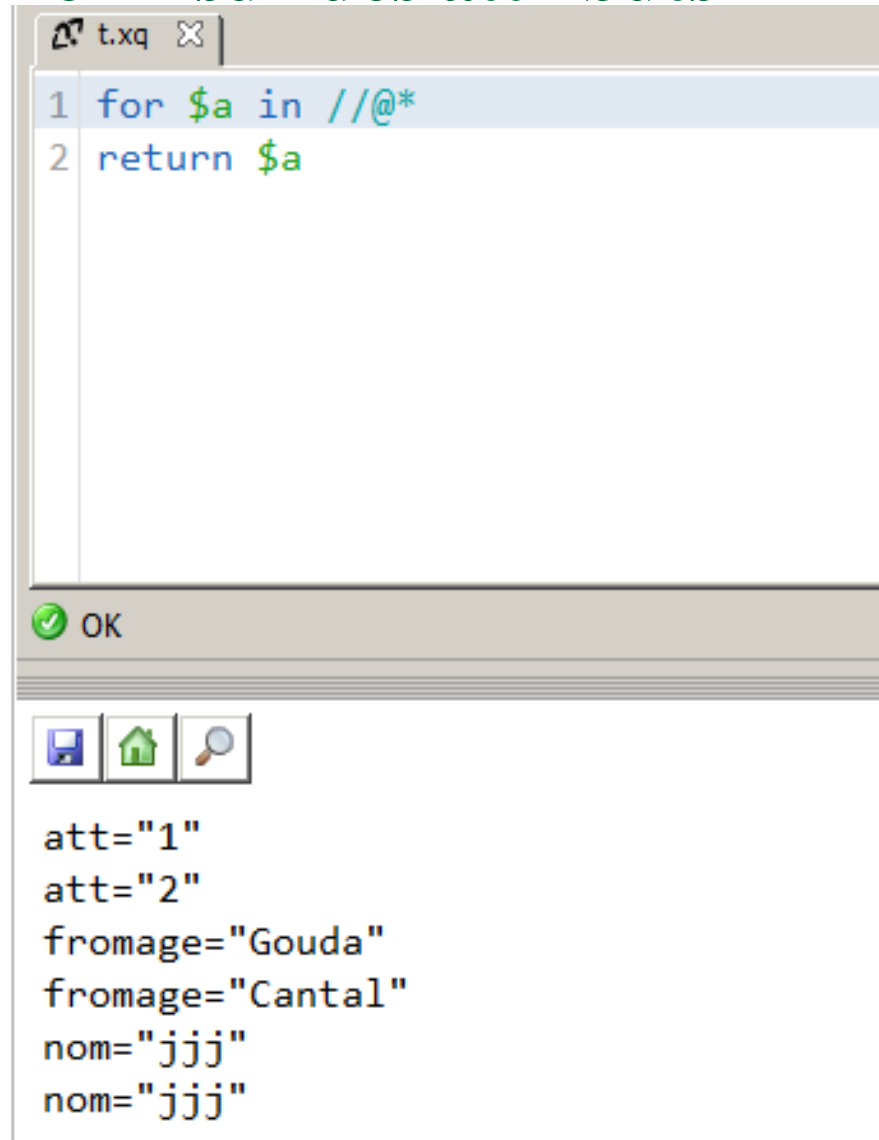
Below the code editor, a status bar shows a green checkmark, the text 'OK', and the number '3 : 1'. The bottom section of the window displays the results of the query, labeled 'Résultat' on the right. The results are a list of numbers from 1 to 10, each on a new line.

Résultat

```
1
2
3
4
5
6
7
8
9
10
```

# Boucles - clause « for » sur des attributs

```
for $a in //@*  
return $a
```

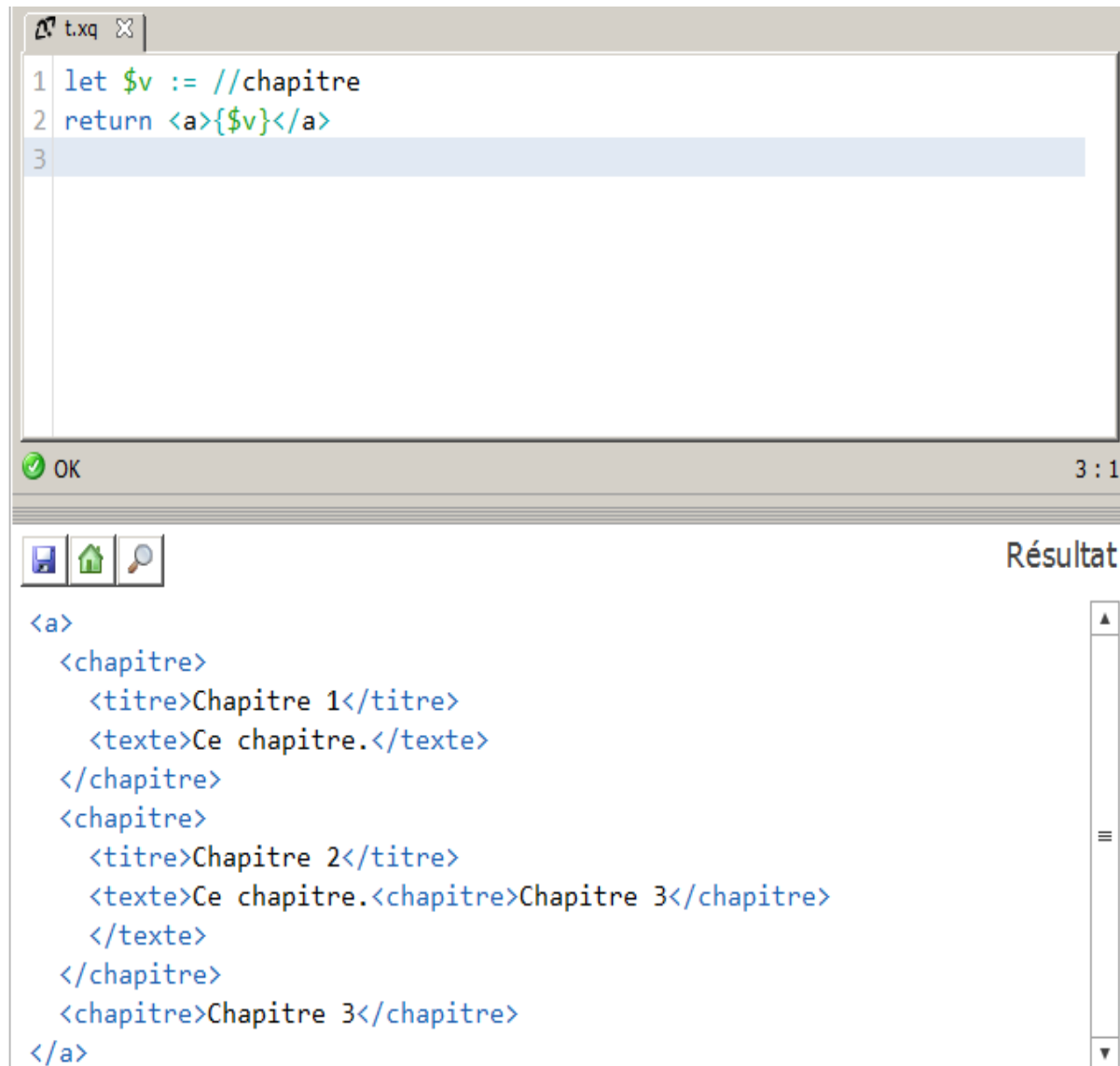


## Clause « let » - variables

- La clause « **let** » permet de déclarer (définir) une variable.
- Pour afficher le contenu d'une variable déclarée avec « let », on utilise la clause « **return** ». Cette dernière indique ce qu'il faut produire en sortie.
- Les clauses « **let** » et « **return** » sont utilisées dans les expressions « **FLWOR** ».
- Exemple (tester avec « xpath-1.xml »):
- **let** \$v := //chapitre
- **return** <a>{\$v}</a>
- Remarque : Dans une expression « **FLWOR** », la clause « **let** » permet de définir une variable **locale** (blocs).



# Clause « let » - variables



The screenshot shows a TiddlyWiki editor window with a tab labeled 't.xq'. The editor contains the following code:

```
1 let $v := //chapitre
2 return <a>{$v}</a>
3
```

Below the editor, there is a status bar with a green checkmark, the text 'OK', and the page number '3 : 1'. To the right of the editor is a 'Résultat' (Result) pane. It contains a toolbar with icons for save, home, and search. The rendered output is as follows:

```
<a>
  <chapitre>
    <titre>Chapitre 1</titre>
    <text>Ce chapitre.</text>
  </chapitre>
  <chapitre>
    <titre>Chapitre 2</titre>
    <text>Ce chapitre.<chapitre>Chapitre 3</chapitre>
  </text>
  </chapitre>
  <chapitre>Chapitre 3</chapitre>
</a>
```

The 'Résultat' pane also features a vertical scrollbar on its right side.

# Clause « let » - variables avec « element » et « attribute »

- Exemple :
- **let** \$a := "livre"
- **let** \$b := "chapitres"
- **let** \$c := "texte."
- **return element** res {
- **element** ele {
- **attribute** lang { "xpath-1.xml" },
- "Un",\$a, "contient des",\$b,"qui contiennent du",\$c
- }
- }

# Clause « let » - variables avec « element » et « attribute »

t.xq

```
1 let $a := "livre"
2 let $b := "chapitres"
3 let $c := "texte."
4 return element res {
5   element ele {
6     attribute lang { "xpath-1.xml" },
7     "Un",$a, "contient des",$b,"qui contiennent du",$c
8   }
9 }
```



OK



```
<res>
  <ele lang="xpath-1.xml">Un livre contient des chapitres qui contiennent du texte.</ele>
</res>
```

# Clause « let » - portée des variables

```
declare variable $msg :=  
("msg1","msg2");  
<exp>  
<output1>  
{  
let $x:=6  
return  
<a>{$msg[2],"et",$x}</a>  
}  
</output1>  
<a>  
{  
  <b>{$msg[1],"et",$x}</b>  
}  
</a>  
</exp>
```

```
1 declare variable $msg := ("msg1","msg2");  
2 <exp>  
3   <output1>  
4     {  
5       let $x :=6  
6       return <a>{$msg[2],"et",$x} </a>  
7     }  
8   </output1>  
9  
10  <a>  
11  {  
12    <b>{$msg[1],"et",$x}</b>  
13  }  
14 </a>  
15 </exp>
```

Cc [...] \* ≡ Trouver... Remplac

✖ Undeclared variable: \$x.

# Clause « let » - portée des variables

```
<exp>
  <output1>
    <a>msg2 et 5</a>
  </output1>
  <a>
    <b>msg1 et 6</b>
  </a>
</exp>
```

**declare variable \$msg :=**  
**("msg1","msg2");**

**<exp>**  
**<output1>**  
**{**  
**let \$x:=5**  
**return**  
**<a>{\$msg[2],"et",\$x}</a>**  
**}**  
**</output1>**  
**<a>**  
**{**  
**let \$x:=6**  
**return**  
**<b>{\$msg[1],"et",\$x}</b>**  
**}**  
**</a>**  
**</exp>**

```
t.xq X
1 declare variable $msg := ("msg1","msg2");
2 <exp>
3 <output1>
4 {
5 let $x:=5
6 return <a>{$msg[2],"et",$x}</a>
7 }
8 </output1>
9 <a>
10 {
11 let $x:=6
12 return <b>{$msg[1],"et",$x}</b>
13 }
14 </a>
15 </exp>
16
```

OK

# Clause « declare » - portée des variables

```
<exp>
  <output1>
    <a>msg2 et 6</a>
  </output1>
  <a>
    <b>msg1 et 6</b>
  </a>
</exp>
```

**declare variable \$msg :=**  
**("msg1","msg2");**  
**declare variable \$x:=6;**  
**<exp>**  
**<output1>**  
**{**  
**<a>{\$msg[2],"et",\$x}</a>**  
**}**  
**</output1>**  
**<a>**  
**{**  
**<b>{\$msg[1],"et",\$x}</b>**  
**}**  
**</a>**  
**</exp>**

```
t.xq
1 declare variable $msg := ("msg1","msg2");
2 declare variable $x:=6;
3 <exp>
4 <output1>
5 {
6 <a>{$msg[2],"et",$x}</a>
7 }
8 </output1>
9 <a>
10 {
11 <b>{$msg[1],"et",$x}</b>
12 }
13 </a>
14 </exp>
```

## « let » + « for » + « return »

- Il est possible de combiner les trois clauses : **let**, **for** et **return**.
- Exemple:
- **let** \$genres := //genre/text()
- **for** \$genre in \$genres (:On peut boucler directement sur une expression XPath:)
- **return** \$genre

## « let » + « for » + « return »

```
1 let $genres := //genre/text()  
2 for $genre in $genres  
3 return $genre
```

✓ OK

1 : 1



Résultat

Maclura  
Calocedrus  
Pterocarya  
Celtis  
Quercus  
Platanus  
Platanus  
Alnus  
Aesculus  
Ginkgo  
Fraxinus  
Ailanthus  
Taxodium  
Diospyros





## « element » + « let » + « for » + « return »

- Combinaison de **element**, **let**, **for** et **return**.
- Exemple :
- **element** resultat {
- **let** \$noms := //nom\_commun
- **for** \$nom in \$noms
- **return element** nom{\$nom/text()}
- }

# « element » + « let » + « for » + « return »

```
1 element resultat{  
2   let $noms := //nom_commun  
3   for $nom in $noms  
4     return element nom{$nom/text()}  
5 }
```

✓ OK



```
<resultat>  
  <nom>Oranger des osages</nom>  
  <nom>Cedre a encens</nom>  
  <nom>Perocarya du caucase</nom>  
  <nom>Micocoulier de provence</nom>  
  <nom>Chene rouvre</nom>  
  <nom>Platane commun</nom>  
  <nom>Platane commun</nom>  
  <nom>Aulne glutineux</nom>  
  <nom>Marronnier d'inde</nom>  
  <nom>Arbre aux quarante ecus</nom>  
  <nom>Frene commun</nom>  
  <nom>Ailanthé</nom>
```

« **element** » + « **let** » + « **for** » + « **let** » + « **return** »

- Il est possible d'insérer une ou plusieurs affectations avant et après la clause « **for** », comme dans l'exemple suivant :

- **element** resultat {
- **let** \$noms := //nom\_commun
- **for** \$nom in \$noms
- **let** \$res:= starts-with(\$nom,"O")
- **return** element nom{\$nom/text()," : ",\$res}
- }

# « element » + « let » + « for » + « let » + « return »

```
1 element resultat {  
2   let $noms := //nom_commun  
3   for $nom in $noms  
4     let $res := starts-with($nom,"0")  
5     return element nom{$nom/text()," : ",$res}  
6 }  
7
```

✓ OK



```
<nom>Platane d'orient : false</nom>  
<nom>Arbre aux quarante ecus : false</nom>  
<nom>Noisetier de byzance : false</nom>  
<nom>If commun : false</nom>  
<nom>Hetre pourpre : false</nom>  
<nom>Platane commun : false</nom>  
<nom>Marronnier d'inde : false</nom>  
<nom>Arbre aux quarante ecus : false</nom>  
<nom>Arbre aux quarante ecus : false</nom>  
<nom>Noisetier de byzance : false</nom>  
<nom>Catalpa commun : false</nom>  
<nom>Platane d'orient : false</nom>
```

## « let » + « for » + « return » + « if-then-else »

- **let** \$arbre := //arbre
- **for** \$a in \$arbre
- **return if** (\$a/annee < 1950)
- **then** concat(\$a/nom\_commun, " est un arbre ancien")
- **else** concat(\$a/nom\_commun, " est un arbre récent")

# Boucles imbriquées

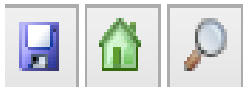
- Exemple :
- **element** resultat {
- **for** \$i in 1 to 5
- **for** \$j in ("a", "b", "c")
- **return** concat(\$j, \$i)
- }

# Boucles imbriquées

```
1 element resultat {  
2   for $i in 1 to 5  
3     for $j in ("a", "b", "c")  
4       return concat($j, $i)  
5   }  
6
```



OK



1 Résultat, 65 b

<resultat>a1 b1 c1 a2 b2 c2 a3 b3 c3 a4 b4 c4 a5 b5 c5</resultat>

## Clause « where »

- La clause « **where** » est une condition optionnelle dans une boucle « for », afin de filtrer ses itérations, comme dans l'exemple suivant :
- `let $arbres := //arbre`
- `for $arbre in $arbres`
- `where $arbre/hauteur > 30`
- `return $arbre`



# Clause « where »

```
1 let $arbres:=//arbre
2 for $arbre in $arbres
3 where $arbre/hauteur > 30
4 return $arbre
```



OK

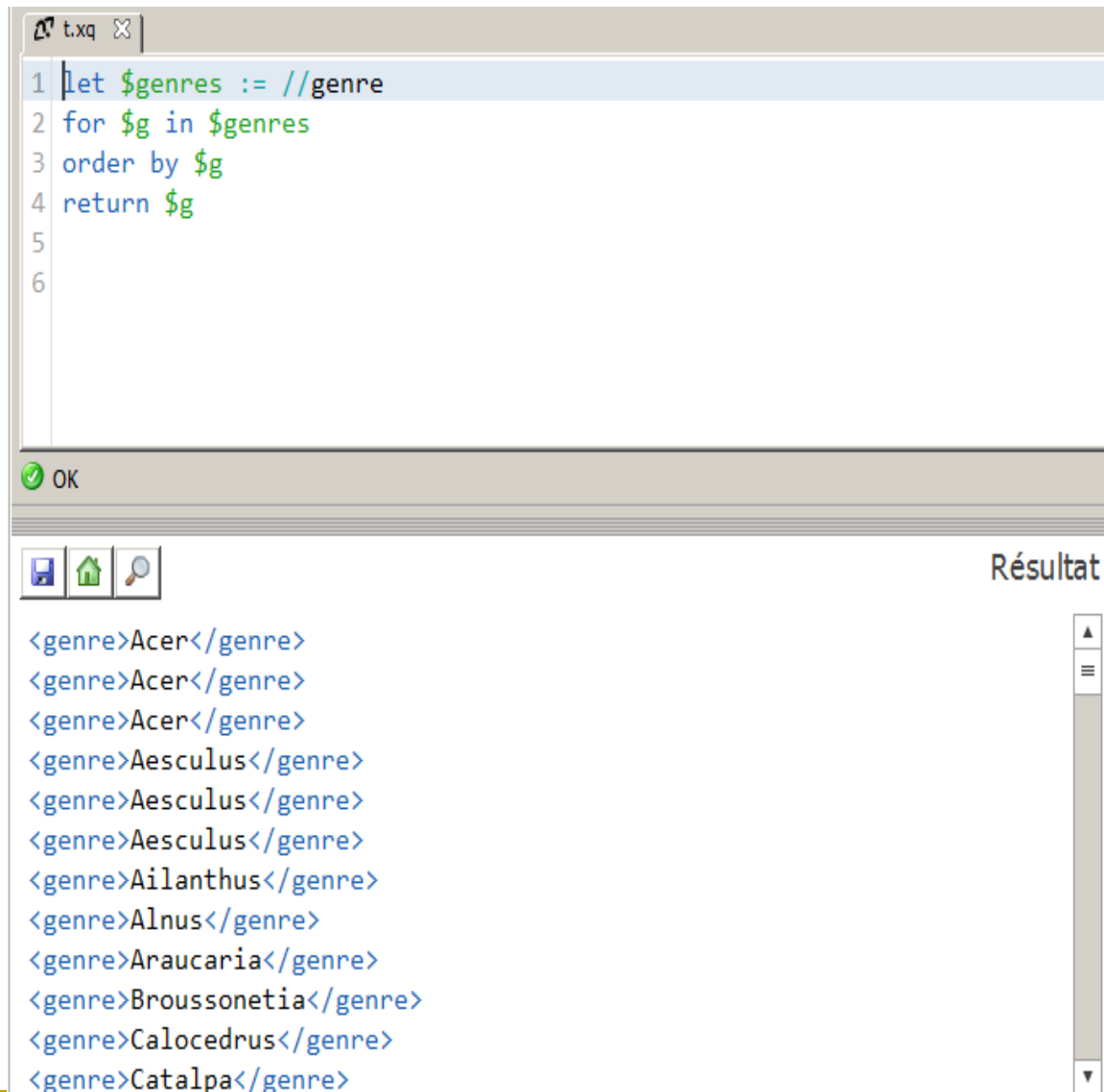


```
<arbre id="21">
  <genre>Platanus</genre>
  <espece>X acerifolia</espece>
  <famille>Platanaceae</famille>
  <nom_commun>Platane commun</nom_commun>
  <annee>1860</annee>
  <hauteur>45.0</hauteur>
  <circonference>405.0</circonference>
  <lieu>
    <geopoint lat="48.8325900983" lon="2.41116455985"/>
    <nom>Bois de Vincennes</nom>
    <nom_precis>Ile de Bercy</nom_precis>
    <adresse arrondissement="12">Ile de Bercy</adresse>
```

## Clause « order by »

- La clause optionnelle « **order by** » permet de trier les éléments à traiter dans la boucle.
- Pour avoir un ordre croissant, on associe à « order by » le mot clé « **ascending** » (tri par défaut).
- Pour avoir un ordre décroissant, on associe à « order by » le mot clé « **descending** ».
- Exemple :
- `let $genres := //genre`
- `for $g in $genres`
- `order by $g (: order by $g descending :)`
- `return $g`

# Clause « order by »



The screenshot shows a TQL editor window titled 't.xq' with the following code:

```
1 let $genres := //genre
2 for $g in $genres
3 order by $g
4 return $g
5
6
```

Below the editor is a status bar with a green checkmark and the text 'OK'. The results pane, titled 'Résultat', displays the output of the query as a list of XML elements:

```
<genre>Acer</genre>
<genre>Acer</genre>
<genre>Acer</genre>
<genre>Aesculus</genre>
<genre>Aesculus</genre>
<genre>Aesculus</genre>
<genre>Ailanthus</genre>
<genre>Alnus</genre>
<genre>Araucaria</genre>
<genre>Broussonetia</genre>
<genre>Calocedrus</genre>
<genre>Catalpa</genre>
```

The results pane includes a vertical scrollbar on the right side.

## Exercice 1

- À partir des fichiers « books1.xml » et « books2.xml », générer le résultat suivant à l'aide d'une requête XQuery :



Résultat

```
<a>
  <fi>S.</fi>
  <first>Serge</first>
</a>
<a>
  <fi>P.</fi>
  <first>Peter</first>
</a>
<a>
  <fi>D.</fi>
  <first>Dan</first>
</a>
```

## Exercice 2

- À partir du fichier « books2.xml », produire, à l'aide d'une requête Xquery, le résultat suivant :



```
<title publisher="O'Reilly"/>
```

Résultat

## Exercice 3

- À partir du fichier « gaz\_rare.xml », produire le résultat suivant à l'aide d'une requête XQuery :

```
<classification_atomique>
  <!--commentaire-->
  <famille type="gaz rare">
    <atome>
      <nom>hélium</nom>
      <symbole>He<autre>autre</autre>
    </symbole>
    <numero>2</numero>
    <masse>masse = 10</masse>
  </atome>
  <atome>
    <nom>néon</nom>
    <symbole>Ne</symbole>
    <numero>10</numero>
    <masse>20</masse>
  </atome>
  <atome>
    <nom>argon</nom>
    <symbole>Ar</symbole>
    <numero>18</numero>
    <masse>40</masse>
  </atome>
</famille>
</classification_atomique>
```

## Exercice 4

- À partir du fichier « gaz\_rare.xml » extraire, via une requête XQuery, la masse strictement inférieure à "20". Produire le résultat suivant :



Résultat

```
<masse>la masse est 4</masse>
```

- Extraire via une autre requête l'atome contenant cette masse.

## Exercice 5

- Modifier la requête XQuery suivante pour afficher à partir du fichier « arbres.xml » les genres d'arbre sans répétition :
- for \$genre in //genre
- return \$genre



## Exercice 6

- À partir du fichier « arbres.xml », produire, à l'aide d'une requête XQuery, le résultat suivant :

```
<resultat>
  <nom>Oranger des osages ID 6</nom>
  <nom>Cedre a encens ID 11</nom>
  <nom>Perocarya du caucase ID 14</nom>
  <nom>Micocoulier de provence ID 16</nom>
  <nom>Chene rouvre ID 19</nom>
  <nom>Platane commun ID 21</nom>
  <nom>Platane commun ID 26</nom>
  <nom>Aulne glutineux ID 28</nom>
  <nom>Marronnier d'inde ID 30</nom>
  <nom>Arbre aux quarante ecus ID 46</nom>
  <nom>Frene commun ID 52</nom>
  <nom>Ailanthé ID 53</nom>
  <nom>Cypres chauve ID 56</nom>
  <nom>Kaki ID 58</nom>
  <nom>Sequoia geant ID 59</nom>
  <nom>Chicot du canada ID 61</nom>
  <nom>Hetre pleureur ID 63</nom>
  <nom>Perocarya du caucase ID 65</nom>
  <nom>Sequoia geant ID 67</nom>
  <nom>Pin noir ID 69</nom>
  <nom>Platane d'orient ID 73</nom>
  <nom>Zelkova du japon ID 83</nom>
  <nom>Noyer noir ID 85</nom>
  <nom>Cypres chauve ID 87</nom>
  <nom>Pin napoleon ID 95</nom>
  <nom>Pin noir ID 97</nom>
```

Remarque : Il y a 97 éléments  
<nom>...</nom> dans le résultat.

## Exercice 7

- Écrire une requête XQuery permettant de compter, à partir du fichier « arbres.xml », le nombre d'arbres ayant une hauteur strictement supérieure à 30 mètres.
- Écrire une requête XQuery permettant d'extraire, à partir du fichier « arbres.xml », tous les arbres ayant une hauteur strictement supérieure à 30 mètres et dont le genre commence par la lettre « G » ou « T ».
- Écrire une requête XQuery permettant d'extraire, à partir du fichier « arbres.xml », tous les arbres ayant une hauteur strictement supérieure à 30 mètres, dont le genre commence par la lettre « G » ou « T » et générer l'élément <position\_XML> contenant la position de chaque arbre dans le fichier XML. Le tout doit être placé dans l'élément racine <res>.
- Écrire une requête XQuery permettant d'extraire, à partir du fichier « arbres.xml », tous les arbres ayant une hauteur strictement supérieure à 30 mètres et se situant dans le 12<sup>ème</sup> ou le 16<sup>ème</sup> arrondissement.

---

## Lien de téléchargement de BaseX

- <https://basex.org/download/>