

# Données du Web - TD2 1

Basil Dalié - Yannis Naidja

Octobre 2019

# Chapitre 1

## XPath : Films

### 1.1 Les titres des films

```
1 /descendant::TITRE/text()
```

### 1.2 Les titres des films parus en 1990

```
1 /descendant::ANNEE[text() =  
  ↪ '1990']/parent::FILM/child::TITRE/text()
```

### 1.3 Le resume d'Alien

```
1 /descendant::TITRE[text() =  
  ↪ 'Alien']/parent::FILM/child::RESUME/text()
```

### 1.4 Quel est le dernier film du document ?

```
1 /descendant::FILM[last()]
```

### 1.5 Quel est le dernier film du document paru en 1990 ?

```
1 /descendant::ANNEE[text() = '1990']/parent::FILM
```

### 1.6 Les titres des films qui ont un résumé

```
1 /descendant::RESUME[text() !=  
  ↪ '']/parent::FILM/child::TITRE/text()
```

## 1.7 Les titres des films dont l'élément résumé n'est pas présent.

```
1 /descendant::FILM[not(child::RESUME)]/child::TITRE/text()
```

## 1.8 Donnez les noms des acteurs qui ont joué dans Vertigo.

```
1 /descendant::ARTISTE[@id = /descendant::FILM[TITRE =  
  → 'Vertigo']/ROLES/ROLE/@idref]/ACTNOM/text()
```

## 1.9 Qui a mis en scène Vertigo ?

```
1 /descendant::ARTISTE[@id = /descendant::FILM[TITRE =  
  → 'Vertigo']/MES/@idref]/ACTNOM/text()
```

## 1.10 Donnez tout les films du directeur de Vertigo.

```
1 /descendant::MES[@idref = /descendant::FILM[TITRE =  
  → 'Vertigo']/MES/@idref]/preceding-sibling::TITRE/text()
```

## 1.11 Donnez les titres des films qui contiennent la lettre "V" (utiliser la fonctioncontains()).

```
1 //TITRE[contains(text(), 'V')]/text()
```

## 1.12 Les titres des films où l'acteur Bruce Willis a joué.

```
1 descendant::FILM/ROLES/ROLE[@idref =  
  → /descendant::ARTISTE[ACTNOM/text() = 'Willis' and  
  → ACTPNOM/text() = 'Bruce']/number(@id)]
```

## 1.13 Quel rôle joue Harvey Keitel dans Reservoir dogs ?

```
1 descendant::FILM[TITRE = 'Reservoir dogs']/ROLES/ROLE[@idref =  
  → /FILMS/ARTISTE[ACTNOM = 'Keitel' and ACTPNOM =  
  → 'Harvey']/@id]/text()
```

### 1.14 Qui a joué avec Harvey Keitel dans Reservoir dogs ?

```
1 /descendant::ARTISTE[@id = /descendant::FILM[TITRE/text() =  
  ↳ "Reservoir dogs"]/ROLES/ROLE/@idref and ACTNOM/text() !=  
  ↳ 'Keitel' and ACTPNOM/text() != 'Harvey']
```

### 1.15 Donnez les nœuds qui ont exactement trois descendants (utiliser la fonctioncount()).

```
1 /descendant::*[count(*)=3]
```

### 1.16 Donnez les nœuds dont le nom contient la chaîne "TI" (utiliser la fonctionname()).

```
1 /descendant::*[contains(name(), 'TI')]
```

### 1.17 Quel est le titre du film qui précède immédiatement Shining (dans l'ordre du document) ?

```
1 /descendant::FILM[TITRE =  
  ↳ 'Shining']/preceding-sibling::FILM[1]/TITRE/text()
```

## Chapitre 2

# XPath : Recettes

### 2.1 Le nom complet de toutes les recettes

```
1 /recettes/recette/@nom
```

### 2.2 Les ingrédients de la recette dont le nom court est "Chiffonnade" ;

```
1 /recettes/recette[@nomCourt =  
  ↳ "Chiffonnade"]/materiel/ingredient/text()
```

### 2.3 Le nom complet des recettes utilisant du "persil" ;

```
1 /recettes/recette/materiel/ingredient[contains(text(),  
  ↳ "persil")]/parent::materiel/parent::recette/@nom
```

### 2.4 (Sans utiliser l'axechild) Le nom complet des recettes utilisant du "persil"

```
1 /descendant::ingredient[contains(text(),  
  ↳ "persil")]/ancestor::recette/@nom
```

### 2.5 Le nom complet des recettes ayant plus de deux ingrédients, et contenant des oeuf

```
1 /recettes/recette[count(materiel/ingredient) > 2 and  
  ↳ materiel/ingredient[contains(text(), "oeuf")]]/@nom
```

## 2.6 (Sans utiliser la fonctioncount()) Le nom complet des recettes ayant plus de deux ingrédients, et contenant l'ingrédient "huile"

```
1 /recettes/recette/materiel/ingredient[last() > 2 and  
  → contains(text(),  
  → "huile")] /parent::materiel/parent::recette/@nom
```

## 2.7 La dernière recette du document

```
1 /recettes/recette[last()]/@nom
```

## Chapitre 3

# XPath : Trains

Exprimer en XPath les interrogations suivantes.

### 3.1 Le numéro des trains qui possèdent une voiture-bar

```
1 /gare/train/voiture/bar/parent::voiture/parent::train/@numero)
```

### 3.2 Le nom des usages ayant effectué au moins une réservation ;

```
1 /gare/usager[@id = /gare/train/voiture/resa/@id]/@nom
```

(Extra) Est-il possible d'exprimer en XPath les requêtes suivantes ?

### 3.3 Le numéro des trains dont au moins 2 places sont réservées :

```
1 /gare/train/voiture[count(resa) >= 2]/parent::train/@numero
```

### 3.4 Le nom des personnes ayant réservé exactement deux fois.

Impossible, car en XPath on ne peut pas compter le nombre d'occurrence d'un attribut

### 3.5 Les usagers n'ayant effectué aucune réservation

```
1 /gare/usager[not(@id = /gare/train/voiture/resa/@id)]
```



## Chapitre 4

# XPath : Tweets

Reprenez votre DTD pour les Tweets, et créez un document XML valide contenant au moins 3 utilisateurs et 5 tweets. Attention : pour le bon déroulement de l'exercice, il sera peut être nécessaire d'apporter des légères modifications à votre DTD afin qu'il soit possible d'interroger vos données !

Donner les requêtes XPath correspondants aux expressions suivantes et évaluer ces expressions dans le document XML créé pour les Tweets.

### 4.1 Les noms des auteurs des tweets.

```
1  /tweeter/users/user[@id =  
    ↳ /tweeter/tweets/tweet/@author_ref]/concat(first_name, ' ',  
    ↳ last_name)
```

### 4.2 Les tweets de l'utilisateur dont l'id est "u41".

```
1  /tweeter/users/user[@id =  
    ↳ /tweeter/tweets/tweet/@author_ref]/concat(first_name, ' ',  
    ↳ last_name)
```

### 4.3 Les tweets contenant l'hashtag "#I;3XML".

```
1  /tweeter/tweets/tweet/body/hashtag[contains(text(),  
    ↳ 'I<3XML')]/ancestor::tweet
```

### 4.4 Le tweet le plus recent.

```
1  /tweeter/tweets/tweet/header/date[text() =  
    ↳ max(/tweeter/tweets/tweet/header/date)]/ancestor::tweet
```

## 4.5 Les tweet sans hashtags.

```
1 /tweeter/tweets/tweet/body[count(hashtag) = 0]/ancestor::tweet
```

## 4.6 Les retweets du tweet dont l'id est "t42".

```
1 /tweeter/tweets/tweet[@id = /tweeter/tweets/tweet[@id =  
  ↳ "T42"]/header/retweets/retweet/@ref]
```

## 4.7 Les utilisateurs ayant répondu au tweet dont l'id est "t42".

```
1 /tweeter/users/user[@id =  
  ↳ /tweeter/tweets/tweet[@id=/tweeter/tweets/tweet[@id =  
  ↳ 'T42']/header/answers/answer/@ref]/@author_ref]
```

## Chapitre 5

# Propriétés des requêtes XPath

**5.1 Reformuler les requêtes suivantes en utilisant exclusivement les axes child, descendant, descendant-or-self, following et following-sibling**

### 5.1.1 Requete 1

```
1 //b[parent::a]
```

devient

```
1 //a/b
```

### 5.1.2 Requete 2

```
1 //a/preceding-sibling::c
```

devient

```
1 //c[following-sibling::a]
```

### 5.1.3 Requete 3

```
1 //c[preceding::d]
```

devient

```
1 //d/following::c
```

#### 5.1.4 Requete 4

```
1 //b/a/preceding-sibling::c/preceding::d
    devient
1 //d[following::c[following-sibling::a[child::b]]]
```

#### 5.1.5 Requete 5

```
1 /a/b/.../*/.../preceding::d
    devient
1 //d[following::*[child::*]]
```

#### 5.1.6 Requete 6

```
1 //a/ancestor::b/parent::c/child::d/parent::e
```

La requête est syntaxiquement correcte mais ne retourne de résultat pour aucun document XML. Voir explication dans la solution au troisième exercice.

### 5.2 Reformuler les requêtes `//a/following : :b` et `//a/preceding : :b` en utilisant les axes descendant-or-self, ancestor, following-sibling et preceding-sibling.

#### 5.2.1 Requete 1

```
1 //a/following::b
    devient
1 //a/ancestor-or-self::node()/following-sibling::node()/descendant-or-self::b
```

#### 5.2.2 Requete 2

```
1 //a/preceding::b
    devient
1 //a/ancestor-or-self::node()/preceding-sibling::node()/descendant-or-self::b
```

**5.3** Pour chaque requête définie aux points 1 et 2, proposer un document XML pour lequel la réponse à la requête n'est pas vide, sinon expliquer pourquoi un tel document n'existe pas.

**5.3.1** Requete 1

```
1 <r>
2   <a>
3     <b>
4     </b>
5   </a>
6 </r>
```

**5.3.2** Requete 2

```
1 <r>
2   <c>
3   </c>
4   <a>
5   </a>
6 </r>
```

**5.3.3** Requete 3

```
1 <r>
2   <a>
3     <d>
4     </d>
5   </a>
6   <c>
7   </c>
8 </r>
```

**5.3.4** Requete 4

```
1 <r>
2   <a>
3     <d>
4     </d>
5   </a>
6   <c>
7   </c>
```

```

8      <a>
9          <b>
10             </b>
11         </a>
12     </r>

```

### 5.3.5 Requete 5

```

1      <r>
2          <d>
3              </d>
4          <a>
5              <b>
6                  </b>
7              </a>
8          </r>

```

### 5.3.6 Requete 6

Il n'y a aucun document XML correspondant car la fin de la requête est :

```

1  c/child::d/parent::e

```

Or un noeud de type c ne peut pas avoir d'enfant de type d qui ait un parent e car un noeud n'a qu'un seul parent.

## 5.4 Donner un document XML pour lequel la requête `//r[a[1] = a[2]]` n'est pas vide sans que les éléments comparés soient strictement identiques.

```

1  <r>
2      <a>
3          abcdef
4      </a>
5      <a>
6          <b>
7              abc
8          </b>
9          <b>
10             def
11         </b>
12     </a>
13 </r>

```

Les deux noeuds de type `a` sont égaux en XPath car ils partagent la sous-chaine `abc`, pourtant il n'y a pas isomorphisme entre les deux sous-arbres et les noeuds textes ne sont pas identiques.

## 5.5 Est il vrai que, dans le cadre du langage XPath, si $X = Y$ et $Y = Z$ alors $X = Z$ ?

Non, en XPath, l'égalité n'est pas transitive.

Voici un contre exemple :

```
1 <root>
2   <x>
3     abc
4   </x>
5   <y>
6     <y1>
7       abc
8     </y1>
9     <y2>
10      def
11    </y2>
12  </y>
13  <z>
14    def
15  </z>
16 </root>
```

Dans ce document XML, les égalités  $x = y$  et  $y = z$  sont vraie en XPath mais l'égalité  $x = z$  est fausse