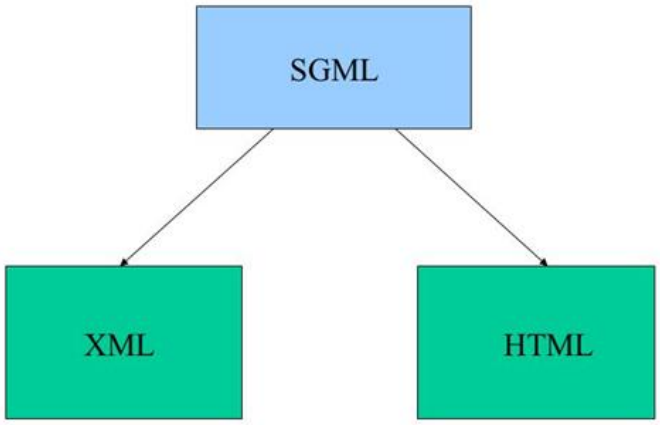
XML : eXtensible Markup Language

Introduction :
XML (eXtensible Markup Language) est une spécification proposée par le W3C (World Wide Web Consortium) en 1998 et qui s'est imposé comme un standard incontournable .Le langage XML est un format général de documents orienté texte. C'est un sous-ensemble de SGML (Standard Generalized Markup Language)



Les avantages de XML

- XML utilise un langage humain et non informatique. XML est lisible et compréhensible
- XML est extensible : XML permet de définir ses propres balises et ses propres attributs.
- Un document XML peut être validé par des règles strictes, contenues par des DTD ou des Schémas, décrivant sa structure et la hiérarchisation de ses données.
- Séparation stricte entre contenu et présentation : le traitement de la mise en forme est rigoureusement séparé de la structure du document XML.
- XML est un format **standardisé** ouvert ne nécessitant aucune licence, intégralement basé texte et qui peut être associé à n'importe quel jeu de caractères.
- Très utilisé : pour le stockage de document, pour l'échange de données entre applications, comme fichier de configuration,…
- ….

Remarque : Plusieurs technologies et de langages qui se sont développés autour de XML. Ceux-ci enrichissent les outils pour la manipulation des documents XML. La liste ci-dessous énumère les principaux langages qui font partie de l'environnement XML.

- XLink et XPointer (liens entre documents)
- XPath (langage de sélection)
- XQuery (langage de requête)
- Schémas XML (modèles de documents)
- XSLT (transformation de documents)

1- STRUCTURE D'UN FICHIER XML DE BASE :

Pour qu'un document XML soir correct, il doit d'abord être ***bien formé*** et, ensuite, être ***valide***. La première contrainte est de nature *syntaxique* (concerne les régles d'écriture) et la seconde contrainte est de nature structurelle (le document doit respecter le modèle d'organisation). Un document XML est généralement contenu dans un fichier texte dont l'extension est .xml

Le document se découpe en deux parties consécutives qui sont le ***prologue*** et le ***corps***. Les commentaires et les instructions de traitement sont ensuite librement insérés avant, après et à l'intérieur du prologue et du corps. La structure globale d'un document XML est la suivante.

```
<?xml ... ?>          ⎤ Prologue
 ...                  ⎦
<root-element>        ⎤
 ...                  | Corps
</root-element>       ⎦
```

1.1- Le prologue : Le prologue contient deux déclarations facultatives mais fortement conseillées ainsi que des commentaires et des instructions de traitement (<? et ?>). La ligne d'introduction d'un document XML est : **<?xml version="1.0" encoding="iso-8859-1" standalone="no"?>**

Cette ligne permet donc d'indiquer la version XML utilisée, le jeu de caractères utilisé et l'autonomie du document :
- **La version XML** : soit 1.0 ou 1.1, sachant que la très grande majorité des documents sont en version 1.0 et que la version 1.1 est assez décriée
- **Encoding :** Le jeu de caractères employé (UTF-8 est l'encoding pris par défaut)

Jeux de caractères du Standard Unicode *liste non exhaustive*	
Norme	**Correspondance**
UTF-8	Jeu de caractères universel sur 8 bits
UTF-16	Jeu de caractères universel sur 16 bits

Jeux de caractères du Standard ISO *liste non exhaustive*	
Norme	**Correspondance**
ISO-8859-1	Latin 1 – Langues d'Europe de l'ouest et d'Amérique latine
ISO-8859-2	Latin 2 – Langues d'Europe centrale et Slaves
ISO-8859-3	Latin 3 – Langues Espéranto, Galicienne, Maltaise et Turc
ISO-8859-4	Latin 4 – Langues Estonienne, Lettonne et Lithuanienne
ISO-8859-5	Langue Cyrilliques
ISO-8859-6	Langue Arabe
ISO-8859-7	Langue Grecque
ISO-8859-8	Langue Hébraïque
ISO-8859-9	Latin 5 – Langue Turc
ISO-8859-10	Latin 6 – Langues Groenlandaises et Lapones

- **standalone** : désigne l'indépendance du document, au sens où il n'existe aucun élément externe qui puisse altérer la forme finale du document XML fourni à l'application par le parseur (références d'entités, valeurs par défaut…). Ce champ prend les valeurs yes ou no. Dans la pratique, il n'a pas grand intérêt et vous pouvez l'ignorer. Si l'attribut *standalone* est omis, c'est la valeur no qui est prise par défaut.

Exemple : Si il n'y a pas de DTD ou si elle est interne, le document est autonome et la valeur de l'attribut standalone peut être définie à *yes*. Si la DTD référencée est externe la valeur de cet attribut doit être définie à no.

Remarques :
- Les attributs *encoding* et *standalone* sont facultatifs.
- Le prologue n'est pas obligatoire et, dans ce cas, le comportement par défaut (version 1.0 et encoding UTF-8) est utilisé.

1.2- L'élément racine

L'élément racine est lui aussi obligatoire. Cet élément est une balise unique dans le document. Elle encadrera le contenu de votre document XML. N'oubliez pas que XML est sensible à la casse, par conséquent <tag> et <Tag> n'ont pas la même signification.

Pour notre exemple, nous nommerons l'élément racine librairie :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<librairie>
...
</librairie>
```

1.3- Le contenu : le document XML est organisé sous format **éléments**, dont l'un est l'élément racine. Un élément est formé d'une balise ouvrante, d'un contenu et de la balise fermante correspondante

Tous les éléments doivent avoir une balise d'ouverture et une balise de fermeture sauf l'élément vide. On peut avoir des éléments imbriqués (attention aux fermetures) , des éléments ayant des attributs et d'autres cas qu'on va traiter au fur et à mesure dans ce cours.

Exemple 1 :
- <LIVRE> . . . </LIVRE>
- <fichier chemin="cours.pdf"/>

Exemple 2 : document bien formé

```
<?xml version="1.0" encoding="iso-8859-1"?>
<stagiaires>

<stagiaire>

<numi>1025</numi>

<nom>khalidi</nom>

<prenom>ahlam</prenom>

<groupe>tdm102</groupe>

</stagiaire>
```

<stagiaire>

<numi>5204</numi>

<nom>IdAli</nom>

<prenom>Khalid</prenom>

<groupe>TRI102</groupe>

</stagiaire>

</stagiaires>

Test : Ouvrez un éditeur de texte et composez ces quelques lignes. Enregistrez le fichier avec l'extension **.xml** et ouvrez le dans MSIE. Vous remarquez que le fichier apparaît sous forme d'arborescence.

Remarque : Pour visualisé un document XML, il vaut faut un "parser" XML (parseur en français). Un parseur est un analyseur syntaxique. La plupart des navigateurs possèdent un parseur XML.

Remarque : Certains caractères ayant une signification particulière dans la grammaire du XML restent interdits : < , > , & .Pour ces caractères comme pour les caractères pouvant poser des problèmes à l'affichage utilisez les caractères de masquages (d'échappement) soit sous forme d'entités nommées, soit sous forme d'entités codées.

liste non exhaustive –

Caractère	Entité nommée	Entité codée
<	&lt;	&#60;
>	&gt;	&#62;
&	&amp;	&#38;

1.4- Commentaires, Instructions de traitement et section CDATA

1.4.1 Les commentaires :

Les commentaires sont introduits dans un document XML comme dans un document HTML : <!-- commentaires -->

1.4.2 Instructions de traitement :

Elles permettent de fournir des informations supplémentaires sur le document aux analyseurs syntaxiques. Une instruction de traitement commence par **<?** et se termine par **?>**. La plus utilisée de ces instructions est celle constituant le prologue d'un document XML : <?xml version="1.0"?>

L'inclusion d'une référence à une feuille de styles utilise aussi ce type d'instruction.

1.4.3 Section CDATA

Cette section permet d'inclure des données textuelles, des exemples de code…, dans un document XML sans qu'il soit nécessaire de substituer les caractères spéciaux par des caractères de masquage. La mise en oeuvre d'une section CDATA s'écrit ainsi :

<![CDATA[

données textuelles

]]>

Remarque : Les sections CDATA ne peuvent pas être imbriquées

Exercice 1 : Création d'un livre en XML
On souhaite écrire un livre en utilisant le formalisme XML. Le livre est structuré en sections (au moins 2), en chapitres (au moins 2) et en paragraphes (au moins 2). Le livre doit contenir la liste des auteurs (avec nom et prénom).Tous les éléments doivent posséder un titre, sauf le paragraphe qui contient du texte.
- Proposez une structuration XML de ce document (avec 2 auteurs, 2 sections, 2 chapitres par section et 2 paragraphes par chapitre).

Exercice 2: Utilisation des attributs
On souhaite compléter la structure du document XML de l'exercice précédent par les attributs nom et prenom pour les auteurs et titre pour le livre, les sections et les chapitres.
Analysez la structure du nouveau document. Y a-t-il des simplifications possibles ?
Vérifiez, à l'aide de l'éditeur, que votre document est bien formé.

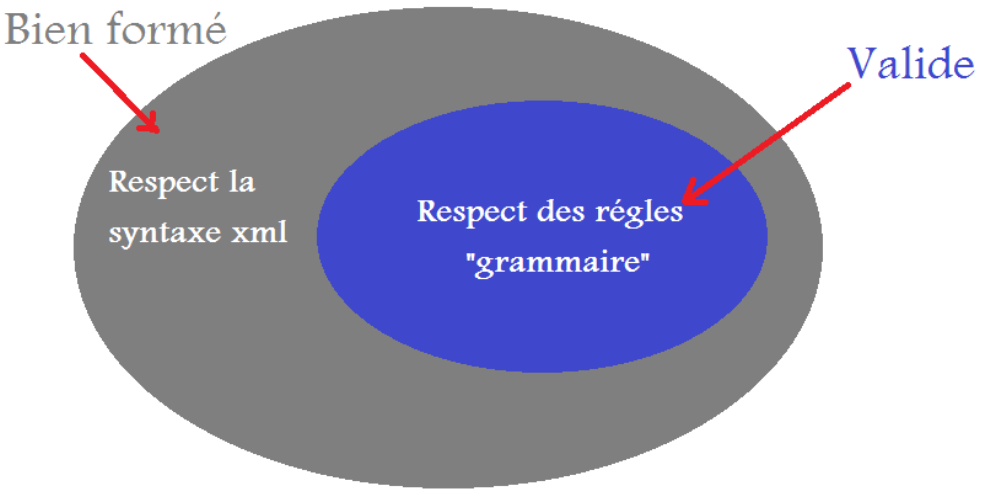
2- La validation des documents XML

Un document XML doit être bien formé, c'est-à-dire, on peut le parser pour en extraire l'arbre correspondant. Mais un document XML n'est en général pas un arbre arbitraire: il encode une donnée structurée d'un langage particulier.

Exemple : un arbre HTML commence obligatoirement par une racine <html> qui a deux fils (et uniquement deux fils) <head> et <body> : Il s'agit de la grammaire (ou du schéma) du langage XHTML.

Un schéma décrit donc

1- Un vocabulaire: ensemble de noms de balises et d'attributs
2- Une structure: comment les choses sont agencées les unes aux autres (hiérarchisation des données)



Pour décrire la structure d'un document XML, on peut utiliser les DTDs (Document Type Definitions) ou les Schémas xml (XSD)

2.1- DTD : Document Type Definition

Une DTD (Document Type Definition) est une forme de grammaire relativement ancienne. La syntaxe d'une DTD n'est pas à base de balise mais à base d'instructions, selon la syntaxe **<!INSTRUCTION…>** . Une DTD peut être interne ou externe au document XML.

- Déclaration d'une DTD externe

Dans chaque document XML, une référence vers la DTD externe est insérée par l'instruction d'en-tête DOCTYPE.

Exemples :

Fichier local	url
<?xml version="1.0"?> **<!DOCTYPE livres SYSTEM "reg.dtd">** <**livres**> … </livres>	<?xml version="1.0"?> **<!DOCTYPE livres SYSTEM "http://www..../.../reg.dtd" ">** <**livres**> … </livres>

- Déclaration d'une DTD interne

Syntaxe : **<!DOCTYPE root-element [declarations]>**

Exemple : document xml avec dtd interne

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (de,a,type,message)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT de (#PCDATA)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT message (#PCDATA)> ]>
<note>
<de>Formatrice</de>
<a>stagiaire</a>
<type>Rappel</type>
<message>N'oublie pas de réviser ton cours</message>
</note>
```

Structure d'une DTD

Une DTD permet essentiellement de spécifier les noms, les contenus et les attributs des éléments qui composent le fichier XML visé.

- Définition des éléments :

Un élément (balise en xml) est exprimé selon la syntaxe suivante : **<!ELEMENT nomElément contenu >**

Avec :

- nomElément est le nom de l'élément;
- contenu indique le type de données ou les éléments qui doivent ou peuvent être imbriqués. Le contenu peut être un :

Contenu textuel	#PCDATA (Parsed Character Data)	<!ELEMENT PRENOM (#PCDATA)>
Vide	Empty	< !ELEMENT BR EMPTY>
Aléatoire	ANY : l'élément peut contenir n'importe quel élément présent dans la DTD.	< !ELEMENT adresse ANY>
Séquence d'éléments	suite de noms d'éléments séparés par des virgules. L'ordre doit être respecté	<!ELEMENT NOMComplet (NOM,PRENOM+)>

Choix	suite de noms d'éléments séparés par des barres verticales "|"	<!ELEMENT NOMComplet (NOM|PRENOM+)>
Mélange	Contenu variable	<!ELEMENT tes (NOM|#PCDATA)>

Remarque : indicateurs d'occurrences

Les contenus peuvent avoir l'un des indicateurs d'occurrences suivants :

- "*****" : Une occurrence quelconque (0 ou plus).
- "**+**" : Une occurrence quelconque (1 ou plus).
- "**?**" : 0 ou 1 occurrence.

Exercice 1 : Proposer un document xml qui est valide selon le DTD suivant :

```
<!ELEMENT Personne ( (M | Mme | Mlle), Prenom+, Nom ) >
<!ELEMENT M EMPTY>
<!ELEMENT Mme EMPTY>
<!ELEMENT Mlle EMPTY>
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT nom (#PCDATA) >
```

Exercice 2 : Proposer un document xml qui est valide selon le DTD suivant :

```
<!ELEMENT Personne ( (M | Mme | Mlle), Prenom+, Nom ,DateNaissance) >
<!ELEMENT M EMPTY>
<!ELEMENT Mme EMPTY>
<!ELEMENT Mlle EMPTY>
<!ELEMENT prenom (#PCDATA) >
<!ELEMENT nom (#PCDATA) >
< !ELEMENT DateNaissance(jour,mois,annee)>
<!ELEMENT jour (#PCDATA) >
<!ELEMENT mois (#PCDATA) >
<!ELEMENT annee (#PCDATA) >
```

Exercice 3 : Proposer un DTD pour que le document xml suivant soit valide :

```
<?xml  version="1.0" encoding="UTF-8" ?>
<Modules>
<cours>
<titre>Prog des applications Web coté client</titre>
<mh>  180h </mh>
<coefficient>3</coefficient>
</cours>
<cours>
<titre>PS</titre>
<mh>120h</mh>
</cours>
</Modules>
```

- Définition des attributs :

Les éléments peuvent avoir des attributs, alors la DTD doit inclure la déclaration de ces attributs via l'instruction ATTLIST selon la syntaxe suivante :

<!ATTLIST nomElement nom TYPE OBLIGATION VALEUR_PAR_DEFAUT>

Avec :

TYPE	○ CDATA : du texte (Character Data) ○ ID : un identifiant unique (caractères et nombres) ○ IDREF : une référence vers un ID ; ○ IDREFS : une liste de références vers des ID (séparés par un blanc) ; ○ NMTOKEN : un mot (contient : des lettres, des chiffres, un point [.] , un tiret [-], un trait de soulignement [_] et un deux-points [:]) ; ○ NMTOKENS : une liste de mots (séparation par un blanc) ; ○ Une énumération de valeurs : chaque valeur est séparée par le caractère |.
OBLIGATION	○ #REQUIRED : obligatoire ○ #IMPLIED : optionnel ○ #FIXED *valeur* : valeur imposée par défaut et non modifiable par une valeur effective.
VALEUR_PAR_DEFAUT	○ la valeur par défaut

Exemples :	Explications
<!ATTLIST chapitre titre CDATA #REQUIRED auteur CDATA #IMPLIED>	L'élément chapitre possède un attribut titre obligatoire et un attribut auteur optionnel.
<!ATTLIST crayon couleur (rouge|vert|bleu) "bleu">	L'élément crayon possède un attribut couleur dont les valeurs font partie de l'ensemble rouge, vert, bleu.
<!ATTLIST produit code ID>	chaque produit doit avoir un code **unique**

Exemple complet (DTD et XML) :

```
<!DOCTYPE magasin [
<!ELEMENT magasin (service+)>
<!ELEMENT service (produit*)>
<!ATTLIST service code ID #REQUIRED>
<!ELEMENT produit (#PCDATA)>
<!ATTLIST produit code ID #REQUIRED> ]>
<magasin>
<service code="A001">
<produit code="DE205"> Soupe </produit>
<produit code="TM206"> Condiment </produit>
</service>
<service code="A003">
<produit code="OU152"> Lessive </produit>
<produit code="AH070"> Essuie-tout </produit>
</service>
</magasin>
```

Exercice : Proposer un DTD au document xml suivant

```
< ?xml version= "1.0"?>
<magasin>
<service code="A001" nom="s1"/>
<service code="A003" nom="s3"/>
<produit code="DE205" codeService="A001"> clé usb </produit>
<produit code="OU152" codeService="A003"> anti-virus </produit>
<produit code="TM206" codeService="A001"> disque dur </produit>
<produit code="AH070" codeService="A003"> outils bureautiques </produit>
</magasin>
```

- Définition des entités :

L'entité associe un nom à une valeur. Ce nom est employé dans le document XML comme une forme d'alias ou de raccourci vers la valeur suivant la syntaxe **&nom;**. La valeur d'une entité peut être interne ou externe.

- Dans la forme interne la syntaxe pour déclarer une entité est simplement la suivante :

<!ENTITY nom "VALEUR">

Exemple :

Dans le DTD	Dans xml
<!ENTITY intic "ista ntic">	<PARA>Les formateurs de la **&intic;** vous souhaiter bon courage dans les exams</PARA>

- Dans la forme externe, on se retrouve avec le même principe qu'avec l'instruction DOCTYPE en tête du document XML assurant le lien vers une DTD. Les mots-clés SYSTEM et PUBLIC servent donc à réaliser un lien vers une valeur présente dans un fichier.

Exemple :

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE book [
<!--Entités externes -->
<!ENTITY chapter1 SYSTEM "chapter1.xml">
<!ENTITY chapter2 SYSTEM "chapter2.xml"> ]>
<book>
<!--Inclusion du fichier chapter1.xml --> &chapter1;
<!--Inclusion du fichier chapter2.xml --> &chapter2;
</book>
```

Les entités ne s'appliquent pas uniquement au document XML. Elles peuvent également servir à la réalisation de la DTD pour limiter les répétitions de blocs de définition (par exemple, un attribut présent dans plusieurs éléments). Cette forme d'entité est appelée **entité paramétrique** et doit être déclarée suivant la syntaxe : **<!ENTITY % nom "VALEUR">**

L'instruction %nom; sert à utiliser une entité paramétrique dans la DTD.

Exemple :

```
<!ENTITY % type_defaut "CDATA  #REQUIRED ">
<!ATTLIST chapitre  titre    %type_defaut; >
```

TD : XML - DTD

Exercice 1 : Rédiger une DTD pour une bibliographie .Cette bibliographie :
contient des livres et des articles

- les informations nécessaires pour un livre sont :
- ✓ son titre général
- ✓ les noms des auteurs
- ✓ ses tomes et pour chaque tome, leur nombre de pages
- ✓ des informations générales sur son édition comme par exemple le nom de l'éditeur, le lieu d'édition, le lieu d'impression, son numéro ISBN
- les informations nécessaires pour un article sont :
- ✓ son titre
- ✓ les noms des auteurs ;
- ✓ ses références de publication : nom du journal, numéro des pages, année de publication et numéro du journal

on réservera aussi un champ optionnel pour un avis personnel.

Exercice 2: Modifier la DTD précédente... (de l'exercice 1)

- en ajoutant un attribut optionnel soustitre à l'élément titre ;

- en faisant de l'élément tome un élément vide et en lui ajoutant un attribut requis nb_pages et un attribut optionnel soustitre ;
- en faisant de l'élément nom_journal un attribut de l'élément journal et en lui donnant comme valeur par défaut Feuille de Chou ;
- en faisant de l'élément annee un attribut de type énuméré, prenant comme valeurs possibles 2002, 2003, 2004, "avant_2002" et "inconnue » et proposant comme valeur par défaut inconnue.

Exercice 3 : **Utilisation d'une DTD**

On a la DTD carnet.dtd suivante :

```
<!ELEMENT carnet (personne+)>
<!ELEMENT personne EMPTY>
<!ATTLIST personne
nom CDATA #REQUIRED
prenom CDATA #IMPLIED
telephone CDATA #REQUIRED>
```

Créez un document XML qui soit valide par rapport à cette DTD.

Exercice 4 : **Création d'une DTD**

Créez une DTD livre.dtd à partir du document livre2.xml créé dans la partie précédente (Partie1 du cours xml)

Rappel: le document « livre2.xml » est le suivant:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<livre titre="Mon livre">
<auteurs> <auteur nom="Brillant" prenom="Alexandre"/> <auteur nom="Briand" prenom="Aristide"/> </auteurs>
 <sections> <section titre="Section 1">
<chapitre titre="Chapitre 1">
<paragraphe>Premier paragraphe</paragraphe> <paragraphe>Deuxième paragraphe</paragraphe> </chapitre> <chapitre titre="Chapitre 2">
<paragraphe>Premier paragraphe</paragraphe> <paragraphe>Deuxième paragraphe</paragraphe> </chapitre>
</section>
<section titre="Section 2">
```

<chapitre titre="Chapitre 1">
<paragraphe>Premier paragraphe</paragraphe> <paragraphe>Deuxième paragraphe</paragraphe>
</chapitre>
<chapitre titre="Chapitre 2">
<paragraphe>Premier paragraphe</paragraphe> <paragraphe>Deuxième paragraphe</paragraphe>
</chapitre>
</section>
</sections>
</livre>

Exercice 5 : **Utilisation des entités paramétriques**

Modifiez la DTD créée dans l'exercice 4 pour faire en sorte que la définition de l'attribut titre soit unique à l'aide d'une entité paramétrique.

Les limites du DTD :

- les DTD ne sont pas au format XML : Cela signifie qu'il est nécessaire d'utiliser un outil spécial pour "parser" un tel fichier, différent de celui utilisé pour l'édition du fichier XML.
- les DTD ne supportent pas les "espaces de nom" : En pratique, cela implique qu'il n'est pas possible, dans un fichier XML défini par une DTD, d'importer des définitions de balises définies par ailleurs
- le "typage" des données est extrêmement limité : que du #PCDATA

2.2- Les Schémas XML (XSD)

Les schémas XML permettent comme les DTD de définir des modèles de documents. Il est ensuite possible de vérifier qu'un document donné respecte un schéma.

Avantage des schémas xml :

- Syntaxe XML
- **Un grand nombre de types de données de base** : booléens, entiers, intervalles de temps, etc.
- Il est possible de créer de **nouveaux types** : par ajout de contraintes sur un type existant, ….
- **La notion d'héritage** : Les éléments peuvent hériter du contenu et des attributs d'un autre élément
- Une grande **facilité** de conception modulaire de schémas

Exemple : DTD – XSD

Fichier xml :

```
<?xml version="1.0" encoding="utf-8"?>
<cours>
 <titre> Exemple 1 d'utilisation des schémas xml </titre>
<auteur>ELGARRAI Zineb</auteur>
<module>
<nom>Programmation Web Coté Client</nom>
<mh>180h</mh>
<date>2011-05-20</date>
  </module>
<module>
<nom>Programmation Web Coté serveur</nom>
<mh>180h</mh>
<date>2011-03-21</date>
</module>
</cours>
```

DTD :

```
<!ELEMENT cours (titre,auteur,module+)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT module (nom,mh,date)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT mh (#PCDATA)>
<!ELEMENT date (#PCDATA)>
```

XSD:

```
<xsd:schema xmlns:xsd=http://www.w3.org/2000/10/XMLSchema …>
<xsd:element name="cours">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="titre" type="xsd:string"/>
<xsd:element name="auteur" type="xsd:string"/>
<xsd:element name="module" minOccurs="1" maxOccurs="unbounded">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="nom" type="xsd:string"/>
<xsd:element name="mh" type="xsd:string" />
<xsd:element name="date" type="xsd:date"/>
```

```
</xsd:sequence>
</xsd:complexType>
</xsd:element>
</xsd:sequence>
<xsd:attribute name="code" type="xsd:string"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

2.2.1- Syntaxe XSD et référencement

Un schéma xml est un fichier xml avec extension « .xsd ». Comme tout document xml, un schéma xml commence par le prologue, puis l'élément racine.

Syntaxe :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--Déclarations d'éléments, d'attributs et définitions de types -->
...
</xsd:schema>
```

Avec :

- xmlns: xsd = "http://www.w3.org/2001/XMLSchema"

Indique que les éléments et types de données utilisés dans le schéma proviennent de l'espace de noms "http://www.w3.org/2001/XMLSchema". Il spécifie également que les éléments et les types de données provenant de l'espace de noms "http://www.w3.org/2001/XMLSchema" doivent être préfixés par « xsd: »

Référencement :

Le référencement d'un schéma XML se fait au niveau de l'élément racine du fichier XML grâce à l'utilisation de 2 attributs:

```
<?xml version="1.0" encoding="UTF-8"?>
<racine xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="chemin_vers_fichier.xsd">
….
</racine>
```

2.2.2- XSD : déclaration des éléments

Syntaxe : <xsd:element name="nom_element" type="type_element">

Exemples : <xsd:element name="date-naissance" type="xsd:date">
<xsd:element name="nombre-enfants" type="xsd:integer">

Remarque : Valeur par défaut /Valeur fixe

on peut ajouter l'attribut **default** pour spécifier la valeur par défaut , ou l'attribut **fixed** pour spécifier une valeur fixe égale à "valeur"

Les éléments peuvent être de type simple ou complexe

- **Eléments de types Simples**

Un élément simple est un élément XML qui ne peut contenir que du **texte**. Il ne peut contenir aucun autre élément ou attribut.

Remarque : Le texte peut être de plusieurs types différents. Il peut s'agir de l'un des types prédéfinis (booléen, chaîne, date, etc.), ou il peut s'agir d'un type personnalisé qu'on peut définir nous-même (restrictions ,union ou list)

Remarque : les types simples prédéfinis:

Types Numériques	**xsd:boolean**, **xsd:int** (32 bits), **xsd:short** (16 bits), **xsd:byte** (8 bits), **xsd:long** (64 bits), **xsd:integer (**entier sans limite de précision), **xsd:positiveInteger** (entier strictement positif sans limite de précision), **xsd:negativeInteger** (entier strictement négatif sans limite de précision), **xsd:float** (32 bits conforme à la norme IEEE), **xsd:double** (64 bits conforme à la norme IEEE), **xsd:decimal** (Nombre décimal sans limite de précision)
Types Chaines de caractères	– **xsd:string, xsd:normalizedString (** pas de : tabulation / de saut de ligne / de retour chariot) , **xsd:Name** (Nom XML), **xsd:language** (Code de langue comme fr, en-GB), **xsd:anyURI** (comme http://www.istanticsafi.ma/~cours/).
Types Date & Heures	**xsd:time** format hh:mm:ss[.sss][TZ]. La partie .sss des secondes est optionnelle. Tous les autres champs sont obligatoires. L'heure peut être suivie d'un décalage horaire TZ qui est soit Z pour le temps universel soit un décalage commençant par + ou - comme -07:00. – **xsd:date** : Date au format YYYY-MM-DD. Tous les champs sont obligatoires. – **xsd:dateTime** : Date et heure au format YYYY-MM-DDThh:mm:ss, Par exemple 2008-01-16T14:07:23. Tous les champs sont obligatoires.
Types Hérités des DTD	**xsd:ID (**identifiant un élément), **xsd:IDREF** (référence à un élément par son identifiant), **xsd:IDREFS , xsd:ENTITY**(permet de faire référence à une entité le plus souvent non XML et déclaré dans des fichiers DTD), **xsd:ENTITIES** : liste d'entités externes non XML séparés par des espaces

- **Eléments de types Complexe :**
  Un élément est dit de type complexe s'il contient autres éléments (enfants) ou/et a des attributs.

2.2.3- XSD : déclaration des attributs

L'attribut, ne contenant que du texte, ainsi il est de type simple. L'attribut peut être global et donc réutilisable au sein de plusieurs définitions de type complexe.

Syntaxe : <xsd:attribute name="nom_attribut" type="type_attribut"/>

Exemples :
<**xsd:attribute** name="langue" type="xsd:string"/>
<**xsd:attribute** name="categorie" type="xsd:string"/>

Remarque 1 : Attributs et Options
Les attributs sont facultatifs par défaut. Pour spécifier que l'attribut est obligatoire, utilisons l'attribut "**use**":
L'attribut **use** de xsd:attribute peut prendre la valeur **optional** ; **required** ou **prohibited**

Remarque 2: Attributs et Valeurs par défaut
Les attributs peuvent avoir une valeur par défaut (**default**) OU une valeur fixe spécifiée (**fixed**). Une valeur par défaut est automatiquement attribuée à l'attribut lorsqu'aucune autre valeur n'est spécifiée. Une valeur fixe est également automatiquement attribuée à l'attribut et nous ne pouvons pas spécifier une autre valeur.

2.2.4 XSD : Eléments de type complexe

Un élément complexe est un élément XML qui contient d'autres éléments et / ou attributs.

Il existe quatre types d'éléments complexes:

- éléments vides
- éléments qui ne contiennent que d'autres éléments
- éléments contenant uniquement du texte
- éléments contenant à la fois d'autres éléments et du texte

Remarque: Chacun de ces éléments peut également contenir des attributs!

Exemples :

Eléments vides	`<module code="M205"/>`

Eléments qui ne contiennent que d'autres éléments	`<formateur>`
	`<nom>ELGARRAI</nom>`
	`<prenom>Zineb</prénom>`
	`</ formateur >`

Eléments contenant uniquement du texte	`<examen type="pratique">`
	`Programmation web`
	`</examen>`

Eléments contenant à la fois d'autres éléments et du texte	`<description>cours fait à <dateSeance>2011-05-20</ dateSeance>` `</description>`

Pour définir un **type complexe**, on peut utiliser la balise complexType:

```
<xsd:complexType ...>
...
</xsd:complexType>
```

Le type peut être global et donc associé à un nom (**Méthode 2**) ou bien être local à un élément (**Méthode 1**)

Exemple :

Méthode 1 **(Type anonyme)**

```
<xs:element name="formateur">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="prenom" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

Méthode 2

```
<xs:element name="employee" type=" infosFormateur"/>
<xs:complexType name="infosFormateur">
  <xs:sequence>
    <xs:element name="nom" type="xs:string"/>
    <xs:element name="prenom" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Il est bien évident que la deuxième méthode favorise la réutilisation.
Remarque : On peut également baser un élément complexe sur un élément complexe existant et ajouter des éléments (voir la notion d'extension).

On peut contrôler COMMENT les éléments doivent être utilisés dans les documents avec des indicateurs :

Indicateurs d'ordre	sequence, choice, all

Indicateurs d'occurrences	maxOccurs, minOccurs
Indicateurs de groupes	xsd:group , xsd:attributeGroup.
…	

a. Indicateurs d'ordre :

- Opérateurs de séquences : xsd:sequence définit un nouveau type formé d'une suite des éléments en séquence **ordonnée**. C'est l'équivalent de l'opérateur *' , ' des DTD*.

DTD	XSD
<!ELEMENT livre (titre, auteur, isbn)> **< !ELEMENT titre (#PCDATA)>** **< !ELEMENT auteur (#PCDATA)>** **< !ELEMENT isbn (#PCDATA)>**	<xsd:element name="livre"> <xsd:complexType> **<xsd:sequence>** <xsd:element name="titre" type="xsd:string"/> <xsd:element name="auteur" type="xsd:string"/> <xsd:element name="isbn" type="xsd:string"/> </xsd:sequence> </xsd:complexType> </xsd:element>

- Opérateurs de choix : **xsd:choice** définit un nouveau type formé d'une suite des éléments énumères de choix. C'est l'équivalent de l'opérateur ' | ' des DTD.

DTD	XSD
<!ELEMENT publication (livre | article | rapport)> **< !ELEMENT livre (#PCDATA)>** **< !ELEMENT article (#PCDATA)>** **< !ELEMENT rapport (#PCDATA)>**	<xsd:element name="publication"> <xsd:complexType> **<xsd:choice>** <xsd:element name="livre" type="xsd:string"/> <xsd:element name="article" type="xsd:string"/> <xsd:element name="rapport" type="xsd:string"/> </xsd:choice> </xsd:complexType> </xsd:element>

- Opérateurs d'ensemble : Il définit un nouveau type dont chacun des éléments doit apparaître une fois dans un ordre quelconque.

b. Indicateurs d'occurrence :

Les attributs **minOccurs** et **maxOccurs** permettent de préciser le nombre minimal ou maximal d'occurrences d'un élément ou d'un groupe. Ils sont l'équivalent des opérateurs ?, * et + des DTD.

L'attribut minOccurs prend un entier comme valeur. L'attribut maxOccurs prend un entier ou la chaîne unbounded comme valeur pour indiquer qu'il n'y a pas de nombre maximal. La valeur par défaut de ces deux attributs est la valeur 1.

Exemple : soit en DTD : <!ELEMENT elem (elem1, elem2?, elem3*) > . Donner son equivalent en xsd

c. Indicateurs de groups :

Il est possible de nommer des groupes d'éléments et des groupes d'attributs afin de pouvoir les réutiliser. Ce mécanisme aide à structurer un schéma complexe et vise à obtenir une meilleure modularité dans l'écriture des schémas. Les groupes d'éléments et d'attributs sont respectivement définis par les éléments **xsd:group** et **xsd:attributeGroup**.

- xsd:group : permet de définir un groupe d'éléments dont le nom est donné par l'attribut **name**. Le contenu de l'élément xsd:group est un fragment de type nécessairement inclus dans un élément xsd:sequence, xsd:choice ou xsd:all. L'utilisation d'un groupe est équivalente à l'insertion de son contenu. L'intérêt d'un groupe est de pouvoir l'utiliser à plusieurs reprises et de factoriser ainsi les parties communes à plusieurs types.

Exemple : Donner un document xml valide selon ce schéma

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:group name="persongroup">
  <xs:sequence>
    <xs:element name="nom" type="xs:string"/>
    <xs:element name="prenom" type="xs:string"/>
    <xs:element name="date" type="xs:date"/>
  </xs:sequence>
</xs:group>

<xs:element name="formateur" type="infosFormateur"/>

<xs:complexType name="infosFormateur">
  <xs:sequence>
    <xs:group ref="persongroup"/>
    <xs:element name="matricule" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
</xs:schema>
```

- xsd:attributeGroup : permet de définir un groupe d'attributs dont le nom est donné par l'attribut **name**. Le contenu de l'élément xsd:attributeGroup est constitué de déclarations d'attributs introduites par l'élément xsd:attribute.

Exemple :

```
<xs:attributeGroup name="personattrgroup">
  <xs:attribute name="nom" type="xs:string"/>
  <xs:attribute name="prenom" type="xs:string"/>
  <xs:attribute name="date" type="xs:date"/>
</xs:attributeGroup>

<xs:element name="formateur">
  <xs:complexType>
    <xs:attributeGroup ref="personattrgroup"/>
  </xs:complexType>
</xs:element>
```

Exercices :

Exercice 1: Soit un document XML contenant un nombre indéterminé d'éléments sous la forme :
<contact titre="..." techno="...">
<nom>...</nom>
<prenom>...</prenom>
<telephone> ...</telephone>

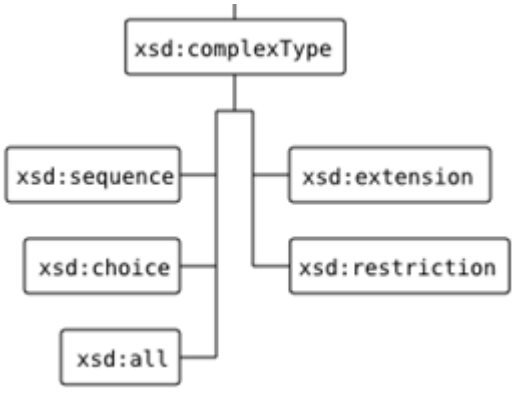
<email>...</email>
<email>...</email>
...
</contact>
L'élément telephone et l'attribut techno sont en option . Les textes seront des chaînes simples xs:string. Vous utiliserez les types complexes numerosType et contactType pour construire un schéma Nommé annuaire.xsd.

Exercice 2 : soit le document xml suivant :

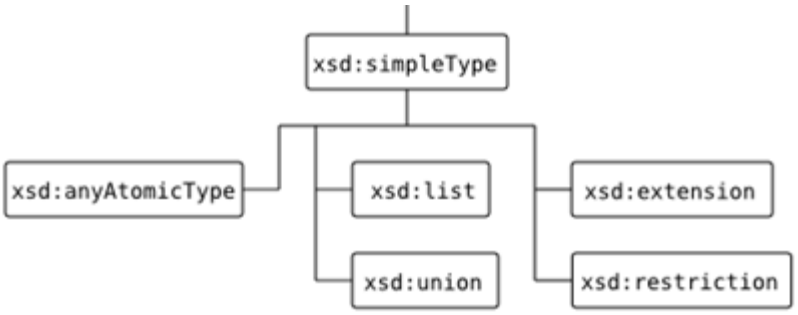
```
<?xml version="1.0" ?>
<encyclopedie>
<personne datenaissance="1942-01-08"
sexe="H">
 <nom>HAWKING</nom>
<prenom>Stephen</prenom>
<publication>Une brève histoire du
temps</publication>
</personne>
<personne datenaissance="1932-07-13"
sexe="H">
<nom>REEVES</nom>
<prenom>Hubert</prenom>
<publication>L'Univers expliqué à mes petits-
enfants</publication>
   </personne>
   <personne datenaissance="1879-03-14"
sexe="H">
<nom>EINSTEN</nom>
<prenom>Albert</prenom>
<publication>Des ondes gravitationnelles
</publication>
 <publication>Sur la théorie quantique du
rayonnement
</publication> </personne>
 <personne datenaissance="1867-11-07"
sexe="F">
<nom>CURIE</nom>
<prenom>Marie</prenom>
<publication>traité de radioactivité
</publication>
</personne>
</encyclopedie>
```

donner un schéma xsd qui permet de valider le document xml ci-dessus

Résumé Types Complexes
Un type complexe peut être construit explicitement ou être dérivé d'un autre type par **extension** ou **restriction**.



2.2.5 XSD : Eléments de type simple
Les types simples définissent uniquement des contenus textuels (pas d'attributs ni d'éléments enfants). Ils peuvent être utilisés pour les éléments ou les attributs.



Nous avons déjà définit les types simples de base (prédéfinis) : xsd:string, xsd:decimal, xsd:integer, xsd:boolean, xsd:date,….
Exemple :

Xml	Xsd
<module>PWCC</module>	<xsd:element name="module" type="xs:string"/>

On peut définir un nouveau type simple à l'aide de la syntaxe suivante :

```
<xsd:simpleType  ….>

(annotation?,(extension|restriction|list|union))

</xsd:simpleType>
```

- Opérateurs d'union

L'opérateur **xsd:union** définit un nouveau type simple dont les valeurs sont celles des types listés dans l'attribut memberTypes.
Exemple : <xsd:simpleType name="IntegerType">
<xsd:union memberTypes="xsd:nonPositiveInteger xsd:nonNegativeInteger">
</xsd:simpleType>

- Opérateurs de liste

L'opérateur **xsd:list** définit un nouveau type simple dont les valeurs sont les listes de valeurs du type simple donné par l'attribut itemType. Il s'agit uniquement de listes de valeurs séparées par des espaces. Ces listes sont souvent utilisées comme valeurs d'attributs.
Exemples : <!-- Type pour les listes d'entiers -->
<xsd:simpleType name="IntList"> <xsd:list itemType="xsd:integer"/> </xsd:simpleType>
<!-- Type pour les listes de 5 entiers -->
<xsd:simpleType name="IntList5">
<xsd:restriction base="IntList"> <**xsd:length** value="5"/> </xsd:restriction> </xsd:simpleType>

2.2.6- Restriction (Restriction de types)

La restriction permet d'obtenir un type dérivé à partir d'un type de base (héritage). L'idée générale de la restriction est de définir un nouveau type dont les contenus au sens large sont des contenus du type de base. La restriction d'un type est introduite par l'élément **xsd:restriction** dont l'attribut **base** donne le nom du type de base.

1- Restriction par longueur :

Pour limiter la longueur d'une valeur dans un élément, nous utiliserions les contraintes de longueur, maxLength et minLength.

Exemples :

Exemple 1	Exemple 2
<xsd:element name="password"> <xsd:simpleType> <xsd:**restriction base**="xsd:string"> <**xsd:Length** value="5"/> </xsd:restriction> </xsd:simpleType> </xsd:element>	<xsd:element name="password"> <xsd:simpleType> <xsd:**restriction base**="xsd:string"> <**xsd:minLength** value="5"/> <**xsd:maxLength** value="8"/> </xsd:restriction> </xsd:simpleType> </xsd:element>

2- Restriction par intervalle

Exemple 1	**Exemple 2**
<xsd:element name="annee"> <xsd:simpleType> <**xsd:restriction base**="xsd:integer"> <xsd:**minInclusive** value="1970"/> <xsd:**maxInclusive** value="2050"/> </xsd:restriction> </xsd:simpleType> </xsd:element>	<xsd:attribute name="date"> <xsd:simpleType> <**xsd:restriction base**="xsd:date"> <!-- Date après le 1er janvier 2010 exclus --> <xsd:**minExclusive** value="2010-01-01"/> </xsd:restriction> </xsd:simpleType> </xsd:attribute>

3- Restriction par énumération

```
<xsd:element name="language" type="Langage"/>
 <xsd:simpleType name="Langage">
  <xsd:restriction base="xsd:language">
   <xsd:enumeration value="ar"/>
   <xsd:enumeration value="an"/>
   <xsd:enumeration value="fr"/>
 </xsd:restriction>
</xsd:simpleType>
```

4- Restriction par motif

Exemple 1	Exemple 2
<xs:element name="sexe"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:**pattern** value="homme|femme"/> </xs:restriction> </xs:simpleType> </xs:element>	<xs:element name="lettre"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:pattern value="([a-z][A-Z])+"/> </xs:restriction> </xs:simpleType> </xs:element>

5- Traitement des caractères d'espacement (xsd:whiteSpace)

Cette facette est particulière. Elle ne restreint pas les valeurs valides mais elle modifie le traitement des caractères d'espacement à l'analyse lexicale. Cette facette peut prendre les trois valeurs preserve, replace et collapse qui correspondent à trois modes de fonctionnement de l'analyseur lexical.

```
<xsd:element name="address">
  <xsd:simpleType>
    <xsd:restriction base="xs:string">
      <xsd:whiteSpace value="preserve"/>
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>
```

De façon générale :
- value="preserve" : le processeur XML ne supprimera aucun caractère d'espace blanc.
- value="replace" : le processeur XML remplacera tous les caractères d'espaces blancs (sauts de ligne, tabulations, espaces et retours chariot) par des espaces.
- value="collapse" : tous les caractères d'espacement seront remplacés par des espaces, les espaces de début et de fin sont supprimés et les espaces multiples sont réduits à un seul espace.

Résumé sur les restrictions (Facettes)

Contrainte	Description
enumeration	Définit une liste de valeurs acceptables
fractionDigits	Spécifie le nombre maximal de décimales autorisé. Doit être égal ou supérieur à zéro
length	Spécifie le nombre exact de caractères ou d'éléments de liste autorisés. Doit être égal ou supérieur à zéro
maxExclusive	Spécifie les limites supérieures des valeurs numériques (la valeur doit être inférieure à cette valeur)
maxInclusive	Spécifie les limites supérieures des valeurs numériques (la valeur doit être inférieure ou égale à cette valeur)
maxLength	Spécifie le nombre maximal de caractères ou d'éléments de liste autorisés. Doit être égal ou supérieur à zéro
minExclusive	Spécifie les limites inférieures des valeurs numériques (la valeur doit être supérieure à cette valeur)
minInclusive	Spécifie les limites inférieures des valeurs numériques (la valeur doit être supérieure ou égale à cette valeur)
minLength	Spécifie le nombre minimum de caractères ou d'éléments de liste autorisés. Doit être égal ou supérieur à zéro
pattern	Définit la séquence exacte de caractères acceptable
totalDigits	Spécifie le nombre exact de chiffres autorisés. Doit être supérieur à zéro
whiteSpace	Spécifie comment les espaces blancs (sauts de ligne, tabulations, espaces et retours chariot) sont gérés

Remarque : on peut, aussi, faire des **restrictions des types complexes**. L'idée reste la même avec

2.2.6 Les extensions (Extension de types)

L'extension est une façon, comme les restrictions, d'obtenir un type dérivé à partir d'un type de base (Héritage). L'idée générale de l'extension est de rajouter du contenu et des attributs. L'extension s'applique aux types simples et aux types complexes mais elle donne toujours un type complexe.

L'extension d'un type est introduite par l'élément **xsd:extension** dont l'attribut ***base*** donne le nom du type de base. Celui-ci peut être un type prédéfini ou un type défini dans le schéma. Le contenu de l'élément xsd:extension explicite le contenu et les attributs à ajouter au type de base.
L'élément xsd:extension est enfant d'un élément xsd:simpleContent ou xsd:complexContent, lui-même enfant de l'élément xsd:complexType.

1- Extension des types simples

L'extension permet uniquement d'ajouter des attributs pour donner **un type complexe à contenu simple**. Lors d'une extension d'un type simple, l'élément xsd:extension est toujours enfant d'un élément xsd:**simpleContent**. Les déclarations des attributs qui sont ajoutés sont placées dans le contenu de l'élément xsd:extension.

Exemple : Définition d'un type Prix qui étend le type prédéfini xsd:decimal en lui ajoutant un attribut devise de type xsd:string

Xsd	xml
<xsd:complexType name="Prix"> <xsd:**simpleContent**> <**xsd:extension** base="xsd:decimal"> <!-- Attribut ajouté --> <xsd:attribute name="devise" type="xsd:string"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType> <xsd:element name="price" type="Prix"/>	<price Prix= "euro"> 15 </price>

2- Extension des types complexes

2.1- **Types complexes à contenu simple**

Il est possible d'étendre un type complexe à contenu simple pour lui ajouter de nouveaux attributs. On obtient alors un nouveau type complexe à contenu simple qui possède les attributs du type de base et ceux déclarés par l'extension. L'extension d'un tel type est similaire à l'extension d'un type simple. L'élément xsd:extension est encore enfant d'un élément xsd:simpleContent. Les déclarations des attributs qui sont ajoutés sont placées dans le contenu de l'élément xsd:extension.

Exemple :

Xsd	xml
<xsd:complexType name="Prix"> <xsd:simpleContent> <xsd:extension base="xsd:decimal"> <xsd:attribute name="devise" type="xsd:string"/> </xsd:extension></xsd:simpleContent> </xsd:complexType> <xsd:complexType name="PrixPays"> <xsd:**simpleContent**> <**xsd:extension** base="**Prix** "> <!-- Attribut ajouté --> <xsd:attribute name="pays" type="xsd:string"/> </xsd:extension> </xsd:simpleContent> </xsd:complexType> <xsd:element name="price" type="PrixPays"/>	<price Prix= "euro" pays="france" > 15 </price>

2.2- **Types complexes à contenu complexe**

L'extension d'un type complexe à contenu complexe consiste à ajouter du contenu et/ou des attributs. Le contenu est ajouté après le contenu du type de base. L'ajout d'attribut est semblable au cas des types

complexes à contenu simple.

Exemple :

```
 <xs:element name="formateur" type="nom_prenom"/>
   <xs:complexType name="nom_prenom1">
    <xs:sequence>
      <xs:element name="nom" type="xs:string"/>
      <xs:element name="prenom" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
<xs:complexType name="nom_prenom">
<xs:complexContent>
<xs:extension base="nom_prenom1">
<xs:attribute name="age" type="xs:float"/>
</xs:extension>
</xs:complexContent>
</xs:complexType>
```

Annexes

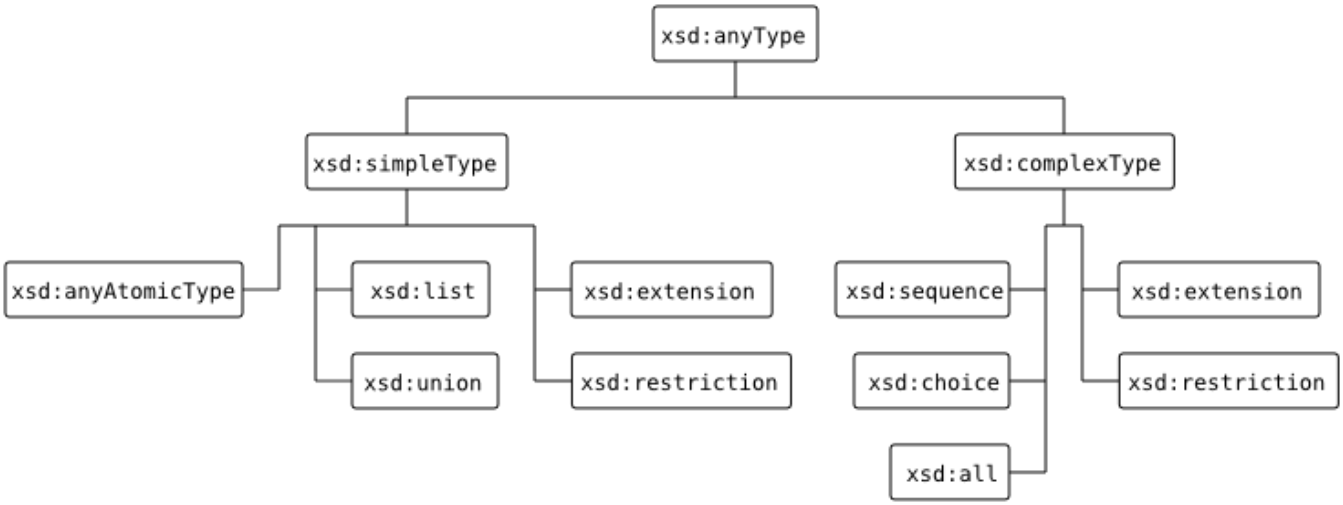
1- Référence à un élément global

Un élément global, c'est-à-dire dont la déclaration par `xsd:element` est enfant direct de l'élément `xsd:schema`, peut être utilisé par des définitions de type. Ces définitions de type se contentent de faire référence à l'élément global de la façon suivante. Ce mécanisme permet une certaine modularité dans l'écriture des schémas puisqu'il évite des répétitions.

```
<!-- Déclaration globale de l'élément title -->
<xsd:element name="title" type="Title"/>
...
<!-- Définition d'un type global ou local -->
<xsd:complexType ... >
  ...
  <!-- Utilisation de l'élément title -->
  <xsd:element ref="title"/>
  ...
</xsd:complexType>
```

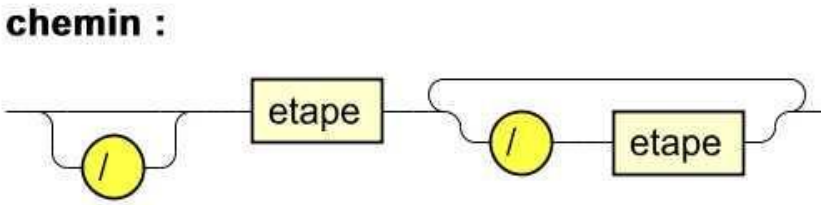
Les deux attributs `name` et `ref` ne peuvent pas être présents simultanément dans l'élément `xsd:element`. Par contre, l'un des deux doit toujours être présent soit pour donner le nom de l'élément défini soit pour référencer un élément déjà défini.

2- Hiérarchie de construction des types



3- Le langage de requête XPath (XML Path Language)

XPath peut être utilisé pour parcourir les éléments et les attributs d'un document XML. Ainsi, Une expression XPath (non xml) est une requête qui sert principalement à extraire telle ou telle partie de l'arborescence XML.



XPath contient plus de 200 fonctions intégrées. Aujourd'hui, les expressions XPath peuvent également être utilisées en JavaScript, Java, schéma XML, PHP, Python, C et C ++, et de nombreux autres langages. Aussi, XPath est un élément majeur de la norme XSLT

Remarque :

- XPath 1.0 est devenu une recommandation W3C le 16 novembre 1999.
- XPath 2.0 est devenu une recommandation du W3C le 23 janvier 2007.
- XPath 3.0 est devenu une recommandation W3C le 8 avril 2014.

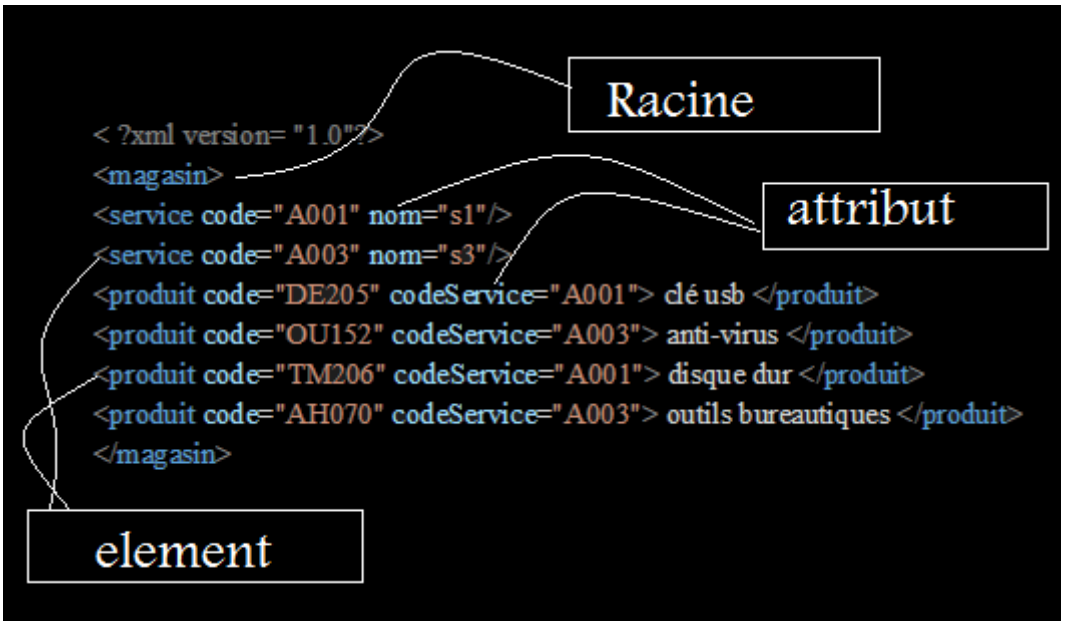
XPath utilise des expressions de chemin pour sélectionner des nœuds ou des ensembles de nœuds dans un document XML. Ces expressions de chemin ressemblent beaucoup aux expressions de chemin que vous utilisez avec les systèmes de fichiers informatiques traditionnels:

Remarque :
- chemin absolu : /étape1/étape2/étape3/...
- chemin relatif : étape1/étape2/étape3/...

3.1- Terminologie XPath

a- Noeuds

Les documents XML sont traités comme des arbres de nœuds. L'élément le plus haut de l'arbre est appelé l'élément racine. Dans XPath, il existe sept types de nœuds: nœuds élément, attribut, texte, espace de noms, instruction de traitement, commentaire et document.



b- les relations entre les nœuds

- **Parent** : Chaque élément et attribut a un parent.
- **Children :** Les nœuds d'élément peuvent avoir zéro, un ou plusieurs enfants.
- **Siblings** : Noeuds ayant le même parent.
- **Ancestors** (ancêtres) : Parent d'un nœud, parent d'un parent, etc.
- **Descendants** (descendance) : Les enfants d'un nœud, les enfants des enfants, etc.

3.2- La syntaxe XPath

XPath utilise des expressions de chemin pour sélectionner des nœuds ou des ensembles de nœuds dans un document XML. Le nœud est sélectionné en suivant un chemin ou des étapes.

a- **Sélection des nœuds :** Le nœud est sélectionné en suivant un chemin ou des étapes. Les expressions de chemin les plus utiles sont répertoriées ci-dessous:

Expression	Description
nodename	Sélectionne tous les nœuds avec le nom "nodename"
/	Sélectionne à partir du nœud racine
//	Sélectionne les nœuds du document du nœud actuel qui correspondent à la sélection, peu importe où ils se trouvent
.	Sélectionne le nœud actuel
..	Sélectionne le parent du nœud actuel
@	Sélectionne les attributs

Exemple :

Doc xml	Expressions XPath & Résultat	
<?xml version="1.0" encoding="UTF-8"?> <stagiaires> <stagiaire numI="1509"> <note module="108">15.25</note> <nom>ELAMARI</nom> <prenom> aya </prenom> </stagiaire> <stagiaire numI="1809"> <note module="108">17.25</note> <nom>khedami</nom> <prenom> ibrahim </prenom> </stagiaire> </stagiaires>	/stagiaires/stagiaire	Sélectionne tous les éléments 'stagiaire' qui sont des enfants de 'stagiaires'
	//stagiaire	Sélectionne tous les éléments 'stagiaire' peu importent où ils se trouvent dans le document
	// @ module	Sélectionne tous les attributs nommés module

b- Prédicats

Les prédicats sont utilisés pour rechercher un nœud spécifique ou un nœud contenant une valeur spécifique (exprimer une condition). Les prédicats sont toujours intégrés entre **crochets.**

Exemples :

Expression XPath	Resultat
/stagiaires/stagiaire[1]	Sélectionne le premier élément stagiaire qui est l'enfant de l'élément de stagiaires.
/stagiaires/stagiaire[last()]	Sélectionne le dernier élément stagiaire qui est l'enfant de l'élément de stagiaires.
/stagiaires/stagiaire[last()-1]	Sélectionne l'avant dernier élément stagiaire qui est l'enfant de l'élément de stagiaires.
/stagiaires/stagiaire[position()<5]	Sélectionne les quatre premiers éléments stagiaire qui est l'enfant de l'élément de stagiaires.
//stagiaire[@numI]	Sélectionne tous les éléments 'stagiaire' qui ont un attribut nommé numI
//stagiaire[@numI='1509']	Sélectionne tous les éléments 'stagiaire' qui ont un attribut numI avec la valeur 1509
//stagiaire[note>10]	Sélectionne tous les éléments 'stagiaire' qui ont un élément de note avec une valeur supérieure à 10.
//stagiaire[note>10]/nom	Sélectionne tous les éléments 'nom' des éléments 'stagiaire' qui ont un élément de note avec une valeur supérieure à 10.

c- Sélection de nœuds inconnus

Les caractères génériques XPath peuvent être utilisés pour sélectionner des nœuds XML inconnus.

caractères génériques	Description
*	Correspond à n'importe quel nœud d'élément

@*	Correspond à n'importe quel nœud d'attribut
node ()	Correspond à n'importe quel nœud de toute sorte

Exemple :
- //* : selectionner tous les éléments du document
- //stagiaire[@*] : Sélectionne tous les éléments 'stagiaire' qui ont au moins un attribut de toute nature

d- Sélection de plusieurs chemins

En utilisant le | Dans une expression XPath, vous pouvez sélectionner plusieurs chemins.
Exemples :
- //note | //nom : sélectionner tous les éléments 'note' et 'nom' dans le document.

Remarque : Les opérateurs XPath
Une expression XPath renvoie soit un ensemble de nœuds, une chaîne, un booléen ou un nombre. Voici une liste des opérateurs qui peuvent être utilisés dans les expressions XPath: | , +, - , *, div, =, !=, < , <=, >, >=, or, and, mod

Liste des exemples :

Expression	Description	Exemple
/AAA	Sélectionne l'élément racine AAA	**<AAA>** **<BBB/>** **<CCC/>** **<BBB/>** **<DDD>** **<BBB/>** **</DDD>** **<CCC/>** </AAA>
AAA/CCC	Sélectionne tous les éléments CCC qui sont enfants de l'élément racine AAA	**<AAA>** **<BBB/>** <CCC/> **<BBB/>** **<DDD>** **<BBB/>** **</DDD>** <CCC/> **</AAA>**
AAA/DDD/BBB	Sélectionne tous les éléments BBB qui sont enfants de DDD, qui sont enfants de l'élément racine AAA	**<AAA>** **<BBB/>** **<CCC/>** **<BBB/>** **<DDD>** <BBB/> **</DDD>** **<CCC/>** **</AAA>**

/AAA/BBB[1]	Sélectionne le premier élément BBB, fils de l'élément racine AAA	**<AAA> <BBB/> <BBB/> <BBB/> </AAA>**
AAA/BBB[last()]	Sélectionne le dernier élément BBB, fils de l'élément racine AAA	**<AAA> <BBB/> <BBB/> <BBB/> <BBB/> </AAA>**
//@id	Sélectionne tous les attributs id	**<AAA> <BBB id = "b1"/> <BBB id = "b2"/> <BBB name = "bbb"/> <BBB/> </AAA>**
//BBB[@id]	Sélectionne tous BBB qui ont un attribut id	**<AAA> <BBB id = "b1"/> <BBB id = "b2"/> <BBB name = "bbb"/> <BBB/> </AAA>**
//BBB[@*]	Sélectionne tous BBB qui ont un attribut	**<AAA> <BBB id = "b1"/> <BBB id = "b2"/> <BBB name = "bbb"/> <BBB/> </AAA>**
//BBB[@name]	Sélectionne tous BBB qui ont un attribut name	**<AAA> <BBB id = "b1"/> <BBB id = "b2"/> <BBB name = "bbb"/> <BBB/> </AAA>**
//BBB[not(@*)]	Sélectionne tous BBB qui n'ont pas d'attribut	**<AAA> <BBB id = "b1"/> <BBB id = "b2"/> <BBB name = "bbb"/> <BBB/> </AAA>**
//BBB[normalize-space(@name)='bbb']	Sélectionne tous les éléments BBB ayant un attribut name dont la valeur est bbb. Les espaces de début de de fin sont supprimés avant la comparaison	**<AAA> <BBB id = "b1"/> <BBB name = " bbb "/> <BBB name = "bbb"/> </AAA>**
//BBB[@id='b1']	Sélectionne tous les éléments BBB ayant un attribut id dont la valeur est b1	**<AAA> <BBB id = "b1"/> <BBB name = " bbb "/> <BBB name = "bbb"/> </AAA>**
//*[count(BBB)=2]	Sélectionne les éléments ayant deux enfants BBB	**<AAA> <CCC> <BBB/>**

		<BBB/> <BBB/> </CCC> <DDD> <BBB/> <BBB/> </DDD> <EEE> <CCC/> <DDD/> </EEE> </AAA>
//*[count(*)=3]	Sélectionne les éléments ayant trois enfants	<AAA> <CCC> <BBB/> <BBB/> <BBB/> </CCC> <DDD> <BBB/> <BBB/> </DDD> <EEE> <CCC/> <DDD/> </EEE> </AAA>
//*[name()='BBB']		<AAA> <BCC> <BBB/> <BBB/> <BBB/> </BCC> <DDB> <BBB/> <BBB/> </DDB> <BEC> <CCC/> <DBD/> </BEC> </AAA>
//*[starts-with(name(),'B')]		<AAA> <BCC> <BBB/> <BBB/> <BBB/> </BCC> <DDB> <BBB/> <BBB/> </DDB>

		<BEC> <CCC/> <DBD/> </BEC> </AAA>
//*[contains(name(),'C')]		<AAA> <BCC> <BBB/> <BBB/> <BBB/> </BCC> <DDB> <BBB/> <BBB/> </DDB> <BEC> <CCC/> <DBD/> </BEC> </AAA>
//*[string-length(name()) = 3]		<AAA> <Q/> <SSSS/> <BB/> <CCC/> <DDDDDDDD/> <EEEE/> </AAA>
//*[string-length(name()) < 3]		<AAA> <Q/> <SSSS/> <BB/> <CCC/> <DDDDDDDD/> <EEEE/> </AAA>
/AAA/EEE | //BBB	Sélectionne tous les éléments BBB et EEE qui sont enfants de l'élément racine AAA	<AAA> <BBB/> <CCC/> <DDD> <CCC/> </DDD> <EEE/> </AAA>
//DDD/parent::*		<AAA> <BBB> <DDD> <CCC> <DDD/> <EEE/> </CCC> </DDD> </BBB> <CCC> <DDD> <EEE>

		<DDD> <FFF/> </DDD> </EEE> </DDD> </CCC> </AAA>
/AAA/BBB/DDD/CCC/ EEE/ancestor::*	Séléctionne tous les éléments donnés dans ce chemin absolu	<AAA> <BBB> <DDD> <CCC> <DDD/> <EEE/> </CCC> </DDD> </BBB> <CCC> <DDD> <EEE> <DDD> <FFF/> </DDD> </EEE> </DDD> </CCC> </AAA>
//FFF/ancestor::*	Séléctionne tous les ancêtres de l'élément FFF	<AAA> <BBB> <DDD> <CCC> <DDD/> <EEE/> </CCC> </DDD> </BBB> <CCC> <DDD> <EEE> <DDD> <FFF/> </DDD> </EEE> </DDD> </CCC> </AAA>

Remarque :

- La fonction count() compte le nombre d'éléments sélectionnés.

- La fonction name() retourne le nom de l'élément,

- la fonction start-with retourne vrai si la chaîne du premier argument commence par celle du deuxième
- la fonction contains retourne vrai si la chaîne du premier argument contient celle du deuxième

- La fonction string-length retourne le nombre de caractères dans une chaîne. Vous devez utiliser < comme substitutif de < et &gt comme substitutif de >

- L'axe "parent" contient le parent du noeud contextuel s'il en a un

- l'axe ancêtre (ancestor) contient les ancêtres du noeud contextuel; cela comprend son parent et les parents des parents etc... Aussi, cet axe contient toujours le noeud racine, sauf si le noeud contextuel est lui-même la racine.

Exemple d'application : (Annexe)

Pour tester les compétences acquises dans cette partie de cours, on va procéder de la façon suivante :

1- L'utilisation d'un objet XMLHttpRequest pour charger des documents XML .
2- Il existe différentes manières de gérer XPath dans différents navigateurs.

- Chrome, Firefox, Edge, Opera et Safari utilisent la méthode evaluate () pour sélectionner les nœuds:

xmlDoc.evaluate (xpath, xmlDoc, null, XPathResult.ANY_TYPE, null);

- Internet Explorer utilise la méthode selectNodes () pour sélectionner le nœud:

xmlDoc.selectNodes (xpath);

Code complet :
Fichier xml : lststg

```
<?xml version="1.0" encoding="UTF-8"?>
<stagiaires>
  <stagiaire numI="1111">
    <note module="108">15.25</note>
    <nom>ELAMARI</nom>
    <prenom> aya </prenom>
  </stagiaire>
  <stagiaire numI="2222">
    <note module="108">17.25</note>
    <nom>khedami</nom>
    <prenom> ibrahim </prenom>
  </stagiaire>
  <stagiaire numI="3333">
    <note module="108">16.25</note>
    <nom>ibrahimi</nom>
    <prenom> laila </prenom>
  </stagiaire>
  <stagiaire numI="4444">
    <note module="108">10.25</note>
    <nom>adili</nom>
    <prenom> ali </prenom>
  </stagiaire>
  </stagiaires>
```

page html :

```
<!DOCTYPE html>
<html>
<body>
    <p id="demo"></p>
    <script>
var xhttp = new XMLHttpRequest();
xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
        showResult(xhttp.responseXML);
```

```
    }
};
xhttp.open("GET", "lststg.xml", true);
xhttp.send();

function showResult(xml) {
    var txt = "les notes : " + "<br>";
    path = "//stagiaire[note>12]/nom"
    if (xml.evaluate) {
        var nodes = xml.evaluate(path, xml, null, XPathResult.ANY_TYPE, null);
        var result = nodes.iterateNext();
        while (result) {
            txt += result.childNodes[0].nodeValue + "<br>";
            result = nodes.iterateNext();
        }
    // Code For Internet Explorer
    } else if (window.ActiveXObject || xhttp.responseType == "msxml-document") {
        xml.setProperty("SelectionLanguage", "XPath");
        nodes = xml.selectNodes(path);
        for (i = 0; i < nodes.length; i++) {
            txt += nodes[i].childNodes[0].nodeValue + "<br>";
        }
    }
    document.getElementById("demo").innerHTML = txt;
}
    </script>

</body>
</html>
```

Remarques :

1- **XLink** est utilisé pour créer des hyperliens dans des documents XML. En effet, Tout élément d'un document XML peut se comporter comme un lien. Avec XLink, les liens peuvent être définis en dehors des fichiers liés. XLink est une recommandation du W3C

2- **XPointer** permet aux liens de pointer vers des parties spécifiques d'un document XML. XPointer utilise des expressions **XPath** pour naviguer dans le document XML. XPointer est une recommandation du W3C

3- Il n'y a pas de support de navigateur pour de XLink et XPointer. Cependant, tous les principaux navigateurs prennent en charge XLinks en SVG.

4- **XQuery** est le langage pour interroger les données XML. XQuery pour XML est comme SQL pour les bases de données. Il est construit sur des expressions XPath. XQuery est pris en charge par toutes les principales bases de données. XQuery est une recommandation du W3C.

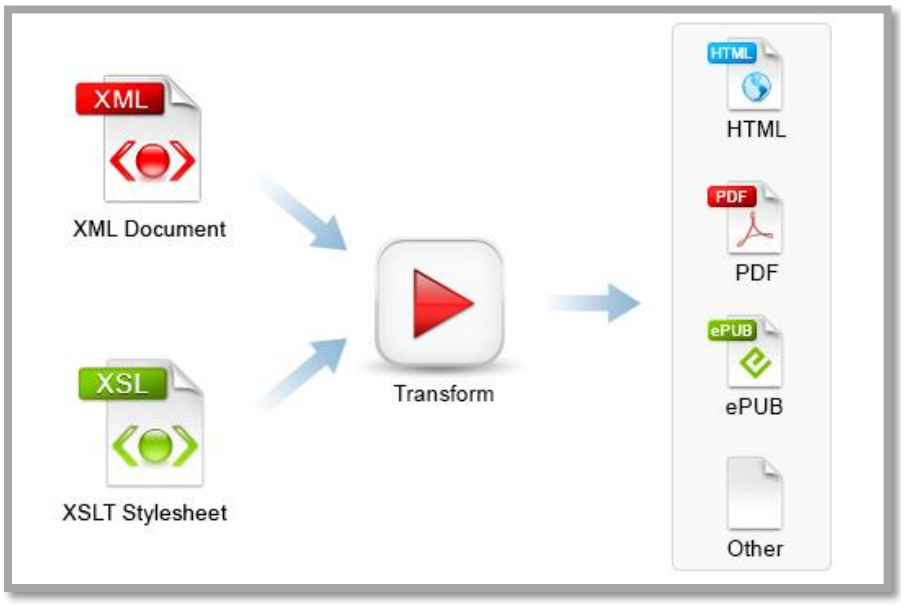
5- XQuery 1.0 et **XPath** 2.0 partagent le même modèle de données et prennent en charge les mêmes fonctions et opérateurs. Si vous avez déjà étudié XPath, vous n'aurez aucun problème à comprendre XQuery.

4- Le XSLT (*Extensible Stylesheet Language Transformations*) :

4.1 Définition :

XSLT est un standard du W3C depuis 1999 .C'est une technologie qui permet de transformer les informations d'un document XML vers un autre type de document (XML, XHTML/HTML, CSV…).

Le principe de fonctionnement : un document XSLT est associé à un document XML afin de créer un nouveau document d'une nature différente ou identique.



4.2 Structure de base :

Une feuille de style XSLT est un document xml (avec extension ".xsl") séparé qui contient des règles de transformation XSLT. Elle doit suivre la syntaxe globale ci-dessous :

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output …. />
 <!-- template -->
</xsl:stylesheet>
```

- Pour accéder aux éléments, attributs et fonctionnalités XSLT, nous devons déclarer l'espace de noms XSLT : "http://www.w3.org/1999/XSL/Transform" (pointe vers l'espace de noms XSLT W3C officiel)

- version = "1.0" représente la version xslt utilisée.

4.3 Référencement :

Le référencement d'un document XSLT se fait au niveau du document XML dont les informations seront utilisées au cours de la transformation. Ce référencement se fait via une ligne à placer sous le prologue et avant l'élément racine du document XML :

<?xml-stylesheet type="text/xsl" href= "mon_document.xsl" ?>

L'attribut type permet de définir le *type* du document que nous souhaitons référencer. Dans notre cas, il s'agit d'un document XSLT : "**text/xsl**". L'attribut href permet d'indiquer l'*URI* du document que l'on souhaite référencer.

Remarque : On peut associer une feuille de style XSL(T) à un (ou plusieurs) documents XML

4.4 XSLT :output

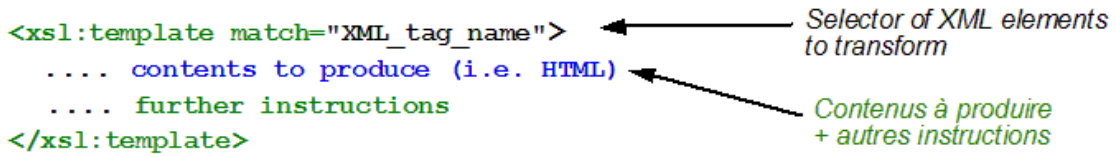
XSLT output, qui se place sous la racine stylesheet, désigne la nature du document en sortie

Syntaxe: **<xsl:output method**="xml" **encoding**="UTF-8" **.. />**

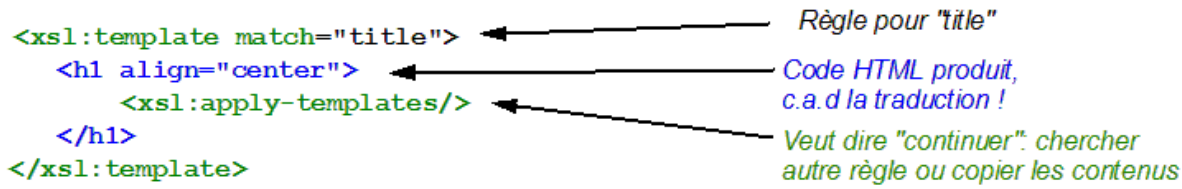
- **method:** préciser le *type* du document produit à l'issue des transformations **: xml, html , xhtml ou text.**
- **encoding:** préciser l'*encodage* du document produit à l'issue des transformations
- la balise **xsl :output** peut avoir d'autres attributs

4.5 XSLT :template

Un document xsl est un ensemble de templates (règles de traduction). Le **contenu d'un template** permet de définir les transformations à appliquer à l'ensemble des données sélectionnées par l'expression XPath qui lui est attachée. Ce contenu peut-être de différentes natures.



Exemple :



Le squelette d'un document XSLT effectuant une transformation vers un document XHTML/HTML serait donc, par exemple, sous cette forme :

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
<html>
<body>...</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

4.6 XSLT: fonctions

Ces fonctions vont nous permettre d'exploiter les données sélectionnées par nos templates afin de procéder à des transformations plus ou moins complexes.

Fonction	Syntaxe & Exemples
value-of : Extraire la *valeur* d'un élément XML ou la valeur de ses attributs.	<**xsl:value-of** select="expression XPath" /> Exemple : <xsl:value-of select="personne/@sexe" /> <xsl:value-of select="personne[1]/prenom" />
for-each : Créer une boucler sur un ensemble d'éléments	Syntaxe : <**xsl:for-each** select="expression XPath" > <!– contenu --> </**xsl:for-each**> Exemple: <xsl:for-each select="//personne"> <xsl:value-of select="prenom" /> </xsl:for-each>
sort : Trier en ordre croissant ou décroissant	Syntaxe **:<xsl:sort select**="expression XPath" /> Exemple : <xsl:for-each select="//personne"> <xsl:sort select="prenom"/> <xsl:value-of select="prenom" /> </xsl:for-each> Remarques: - généralement utilisée au sein d'une fonction <xsl:for-each /> - autres attributs optionnels: • **Order**: **ascending** (croissant), **descending** (décroissant) • **Case-order**: **upper-first** (les majuscules d'abord), **lower-first** (les minuscules d'abord) • **Data-type** : **text** (texte) et **number** (numérique) • **Lang** : code d'une langue (**fr**, **en**, **it**, etc.)
If : Conditionner une transformation	Syntaxe**: <xsl:if test**="condition" **></xsl:if>** Opérateurs logiques: AND, OR, NOT Opérateurs de comparaison: = , not (a = b), < pour inférieur, > pour supérieur Exemple: <**xsl:if test**="@sexe='H'" ><**xsl:value-of** select="nom" /></xsl:if>
Choose Traiter des cas (plusieurs conditions)	Syntaxe **:** **<xsl:choose>** <**xsl:when** test="test de comparaison"> <!-- suite de la transformation --> </**xsl:when**> <**xsl:when** test="test de comparaison"> <!-- suite de la transformation --> </**xsl:when**> <**xsl:otherwise**> <!-- suite de la transformation --> </**xsl:otherwise**> **</xsl:choose>** Exemple <xsl:choose> <xsl:when test="nom=ALAMI"> Bonjour </xsl:when>

	<xsl:when test ="nom=ALAOUI"> Bonsoir </xsl:when> <xsl:otherwise> Rien à dire </xsl:otherwise> </xsl:choose>
apply-templates permet de continuer la transformation des éléments enfants d'un template	Syntaxe**:** **<xsl:apply-templates select**="expression XPATH" /> Un simple apply-templates (sans attribut select) examine tous les nœuds enfants dans l'ordre. Si une règle qui correspond à un nœud est détectée, elle sera appliquée Exemple : <xsl:template match="/"> <xsl:apply-templates select="//personne" /> </xsl:template> <xsl:template match="nom"> <xsl:value-of select="."/> </xsl:template> <xsl:template match="prenom"> <xsl:value-of select="."/> </xsl:template>

Exemple : test sur l'éditeur en line de w3schools (https://www.w3schools.com/xml/tryxslt.asp?xmlfile=cdcatalog&xsltfile=cdcatalog)

XML Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="xsl1.xsl" ?>
<personne>
    <nom>ELGARRAI</nom>
<prenom>zineb</prenom>
</personne>
```

XSLT Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
    <xsl:output    method="html" />
    <xsl:template match="/">
        <html>
            <head> <title>Mon premier document XSLT</title> </head>
            <body>
                <p>Bonjour ! c'est le premier exemple</p>
            <p><xsl:value-of  select="personne[nom='ELGARRAI']/prenom" /> </p>
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Edit the XML or XSLT code above and Click Me »

Bonjour ! c'est le premier exemple

zineb

Exemple 2 :

XML Code:

```
<?xml-stylesheet type="text/xsl" href="xsl1.xsl" ?>
<personnes>
<personne>
     <nom>KHALIDI</nom>
     <prenom>ibrahim</prenom>
</personne>
<personne>
     <nom>ELGARRAI</nom>
     <prenom>Zineb</prenom>
</personne>
<personne>
     <nom>ALMAI</nom>
     <prenom>Ahmad</prenom>
</personne>
</personnes>
```

XSLT Code:

```
 <xsl:template match="/">
    <html>
        <head> <title>Exemple 2</title> </head>
       <body>
<xsl:apply-templates/>
        </body></html>
           </xsl:template>
  <xsl:template match="personne">
  <p><xsl:value-of select="nom"/> + <xsl:value-of
select="prenom"/></p>
  </xsl:template>
</xsl:stylesheet>
```

Edit the XML or XSLT code above and Click Me »

KHALIDI + ibrahim

ELGARRAI + Zineb

ALMAI + Ahmad

Exercice 1 : soit le fichier xml suivant

```xml
<?xml version="1.0"?>

<?xml-stylesheet type="text/xml" href="biblio.xsl"?>
<bibliotheque>
     <livre>
        <titre>N ou M</titre>
       <auteur>Agatha Christie</auteur>
       <ref>Policier-C-15</ref>
</livre>
<livre>
       <titre>Le chien des Baskerville</titre>
       <auteur>Sir Arthur Conan Doyle</auteur>
      <ref>Policier-D-3</ref>
</livre>
<livre>
      <titre>Dune</titre>
     <auteur>Franck Heckbert</auteur>
     <ref>Fiction-H-1</ref>
</livre>
</bibliotheque>
```

Question : Donnez la feuille XSLT qui transforme le document biblio.xml en la page HTML suivante affichée ainsi dans un navigateur :

Bibliotheque

N ou M Agatha Christie Policier-C-15
Le chien des Baskerville Sir Arthur Conan Doyle Policier-D-3
Dune Franck Heckbert Fiction-H-1

solution :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
   <html>
      <head> <title>Ma bibliotheque</title> </head>
     <body> <h1>Bibliotheque</h1>
       <xsl:for-each select="bibliotheque/livre">
         <SPAN style="font-style:italic; padding-right:3pt"><xsl:value-of
select="titre"/></SPAN>
         <SPAN style="color:red; padding-right:3pt"><xsl:value-of select="auteur"/></SPAN>
         <SPAN style="color:blue"><xsl:value-of select="ref"/></SPAN><br />
        </xsl:for-each></body></html>
         </xsl:template>
</xsl:stylesheet>
```

2- Si nous souhaitons classer la bibliothèque par titre en respectant l'ordre alphabétique. Donner la ligne à ajouter pour avoir le résultat :

solution :

nous utiliserons :

….

```
            <xsl:for-each select="bibliotheque/livre">
                <xsl:sort select="."  order="ascending" />
<SPAN style="font-style:italic; padding-right:3pt"> <xsl:value-of select="titre"/></SPAN>
```

….

Exercice 2 :

Ajoutons dans le document XML un attribut type qui correspond au type du livre .Exemple :

```
    <livre type="policier">
      <titre>N ou M</titre>
      <auteur>Agatha Christie</auteur>
      <ref>Policier-C-15</ref>
</livre>
```

Question : On décide de n'afficher que les livres dont le type est "policier" :

Solution :

…..

```
<xsl:for-each select="bibliotheque/livre[@type='policier']">
  <SPAN style="font-style:italic; padding-right:3pt"><xsl:value-of select="titre"/></SPAN>
```
…..

Exercice 3 : Modèle multiples

Pour afficher un élément XML comme livre dans notre exemple, il est possible d'utiliser xsl:applytemplates dans un modèle multiple (à la place du xsl:for-each).
Lorsqu'il y a plusieurs modèles il faut toujours qu'il y en ait un pour l'affichage de la racine du document (le /). Dans le premier modèle le xsl:apply-template indique que pour chaque élément livre enfant de bibliothèque il faut appliquer le deuxième modèle (celui pour lequel l'attribut match a pour valeur livre).

Solution :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
   <xsl:template match="/">
     <html> <head> <title>Ma bibliotheque</title> </head>
     <body> <H2>Bibliotheque</H2>
       <xsl:apply-templates select="bibliotheque/livre" />
    </body>
</html>
</xsl:template>
<xsl:template match="livre">
<SPAN style="font-style:italic; padding-right:3pt"><xsl:value-of select="titre"/></SPAN>
<SPAN style="color:red; padding-right:3pt"><xsl:value-of select="auteur"/> </SPAN>
<SPAN style="color:blue"><xsl:value-of select="ref"/></SPAN>
<br />
</xsl:template>
</xsl:stylesheet>
```