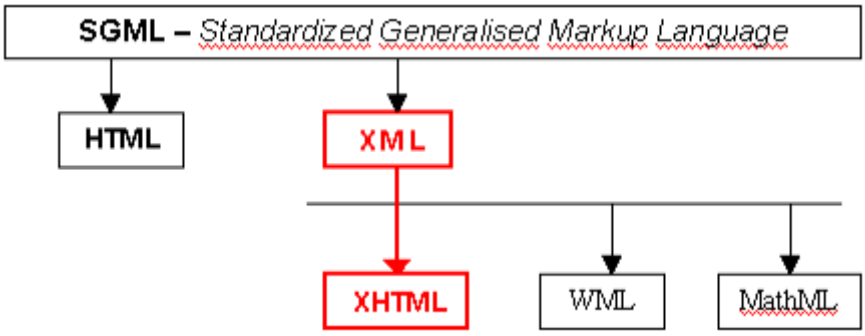
Programmation Web Côté Client : Manipulation des fichiers XML

I. Qu'est-ce que XML?

XML : eXtensible Mark-up Language, c'est une spécification proposée par W3C (Word Wide Web). C'est un langage de balisage pour la description des documents structurés dont le rôle fondamental pour l'échange de données.

Le langage XML est un format général de documents orienté texte. C'est un sous ensemble de SGML



II. INTERÊT de XML :

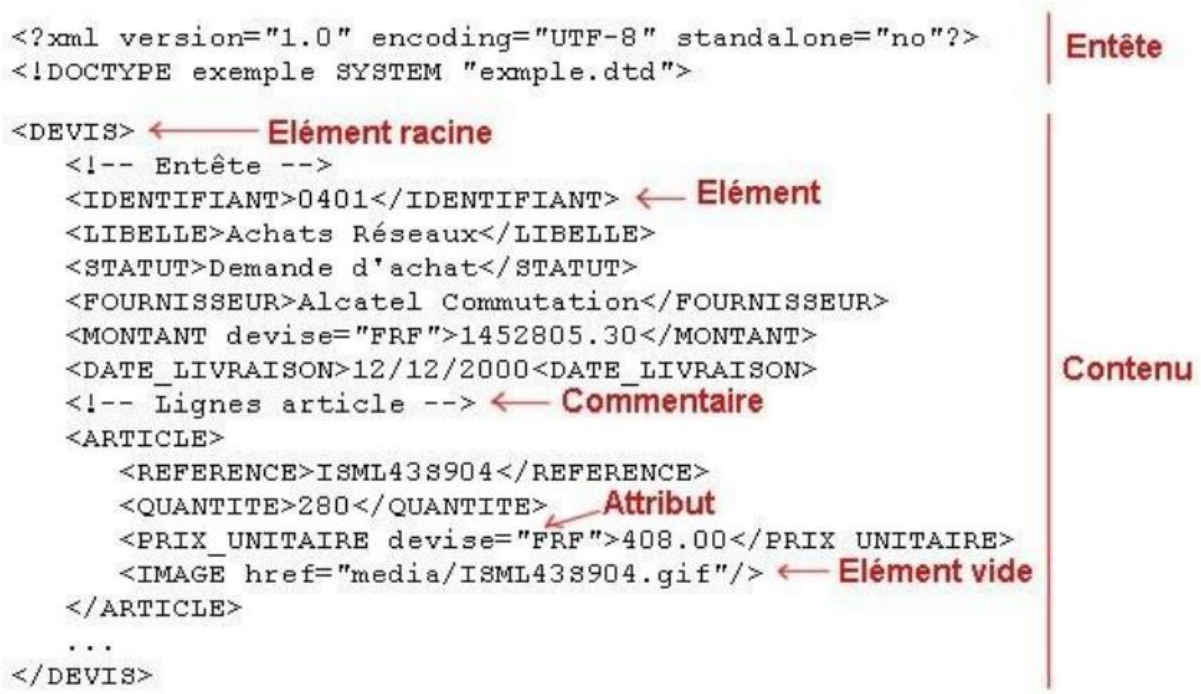
Richesse sémantique :

- Dédié au traitement des données
- Soutenant une grande variété d'applications

Facilité de mise en œuvre :

- Simple et lisible. (Langage humain)
- Portable et facilement utilisable sur Internet
- Assurant un développement aisé SPECIFICATIONS DU Langage
- Libre, indépendant des plateformes logicielles ou matérielles
- XML est extensible : ne contient pas un ensemble fixé de balises.
- XML est particulièrement adapté à l'échange de données et de documents.
- Un document XML peut être validé par des règles contenues par des DTD ou des schémas décrivant sa structure de ses données.

III. Structure d'un document XML




```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE exemple SYSTEM "exmple.dtd">

<DEVIS>
   <!-- Entête -->
   <IDENTIFIANT>0401</IDENTIFIANT>
   <LIBELLE>Achats Réseaux</LIBELLE>
   <STATUT>Demande d'achat</STATUT>
   <FOURNISSEUR>Alcatel Commutation</FOURNISSEUR>
   <MONTANT devise="FRF">1452805.30</MONTANT>
   <DATE_LIVRAISON>12/12/2000<DATE_LIVRAISON>
   <!-- Lignes article -->
   <ARTICLE>
      <REFERENCE>ISML43S904</REFERENCE>
      <QUANTITE>280</QUANTITE>
      <PRIX_UNITAIRE devise="FRF">408.00</PRIX_UNITAIRE>
      <IMAGE href="media/ISML43S904.gif"/>
   </ARTICLE>
   ...
</DEVIS>
```

Contenu d'un document XML :

- Un prologue ou en-tête (déclarations)
- Suivit d'un (SEUL) arbre d'éléments (balises)
- Des commentaires

Un document XML qui respecte **les règles syntaxiques** est dit **bien formé**. Il est utilisable sans DTD (La grammaire de notre document XML).

Un document XML bien formé qui respecte sa DTD est dit **valide**. Il est plus facile d'écrire des feuilles de style (XSLT) pour les documents valides !

Résumé

Le document XML doit être bien formé (syntaxe correcte selon les règles d'écritures), puis il doit être validé (structure correcte selon le modèle d'organisation).

Le document XML est généralement un fichier texte dont l'extension est .xml.

Un document respectant ces critères est dit "bien formé" :

- Un document doit commencer par une déclaration XML
- Toutes les balises avec un contenu doivent être fermées.
- Toutes les balises sans contenu doivent se terminer par les caractères />

- Le document doit contenir un et un seul élément racine.
- Les balises ne doivent pas se chevaucher.
- Les valeurs d'attributs doivent être entre guillemets.
- La casse doit être respectée pour toutes les occurrences de noms de balise.

III.1 LE PROLGUE :

La première déclaration (qui est optionnelle) permet de définir la version et le codage du document.

< ?xml version= « 1.0 » encoding= « iso-8859-1 » standalone= « no » ?>

(standalone= « yes » => pas besoin de DTD externe)

standalone= « no » //valeur par défaut il existe un fichier externe

- La version soit 1.0 ou 1.1, mais la version plus utilisée c'est 1.0

•L'encodage est basé sur la norme ISO 10646 (www.unicode.org).

•XML comprend automatiquement l'encodage UTF-8 et UTF-16 (UTF-8 est l'encodage par défaut). Cad chaque caractère est codé soit sur 8bits ou bien 16 bits.

•La référence à la DTD externe doit être placée au début du fichier :

```
<!DOCTYPE nom_d_élément SYSTEM "test.dtd">
```

•On peut enrichier la DTD externe avec des déclarations locales :

```
<!DOCTYPE nom_d_élément SYSTEM "test.dtd"
    [ déclarations ] >
```

•On peut se passer de référence à une DTD externe et définir toutes les balises dans le document XML :

```
<!DOCTYPE nom_d_élément [ déclarations ] >
```

III.2 commentaires et les instructions de traitement :

En XML les commentaires se notent : **< !- -commentaires- ->**

Les contraintes d'utilisation sont :

- Pas de doubles tirets dans le texte.
- Pas de commentaire dans un élément (l'exemple ci-dessous est incorrect) :

<stagiaire nom= ‘’ALAOUI’’ id= ‘’ 150 ‘’ **< !- -commentaires- -> />**

- Les commentaires sont ignorés (plus ou moins).

En XML les instructions de traitement se notent : **< ?..... ?>**

Elles permettent de fournir des informations supplémentaires sur le document aux analyseurs syntaxiques.

Exemples : prologue < ?xml version= ‘’1.0’ ?>

Feuille de style.

-

III.3 les balises (éléments)

•Forme générale :

<nom-élément> contenu </nom-élément>

•Les noms sont libres (contrairement à HTML). Ils obéissent à quelques règles:

- ✓ 1er caractère {alphabétique, «-», «_» },
- ✓ Les autres caractères {alphabétique, chiffre, «-», «_», «:» }.
- ✓ Pas de blanc,
- ✓ «xml» au début est interdit (maj./min.).

•La balise de fermeture est obligatoire.

• Le contenu d'un élément peut être :

- ✓ Vide (<toc></toc> ou<toc/>).
- ✓ Du texte (sauf «< » et « & ») basé sur l’encodage :
  < → <

  > → > ou bien directement >

  & → &
- ✓ Un ou plusieurs éléments complets : <toc>…</toc>.

- Une répétition de textes et d'éléments :

```
<article> Le langage <def>XML</def> contient <liste>
  <élément> du texte, </élément>
  <élément> des éléments, </élément>
 </liste></article>
```

- ✓ Les blancs comptent : <a>X est différent de<a> X
- ✓ Les deux systèmes de codage des ruptures de lignes sont pris en charge.

III.4 arbres d'élément

- Un document XML est un et un seul arbre d'éléments. C'est à dire :
  - ✓ Pas de chevauchement d'éléments. La notation suivante :

<list> …<item> ... ... </list>…..</item>

Est invalide. Il faut la corriger comme suit

<list> …<item> ... ... </item>…..</list>

Un document XML est composé d'un seul **élément Racine**. La notation suivante :

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

Est invalide. Il faut la corriger comme suit

```
<?xml version="1.0"?>
<groupe>
<etudiant></etudiant>
<etudiant></etudiant>
</groupe>
```

EXEMPLES :

Donner des exemples d'un document bien formé

Exemple1 :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<etudiant>
<nom>alaoui</nom>
<prenom>mohammed</prenom>
</etudiant>
```

Exemple2

```
<?xml version="1.0" encoding="iso-8859-1"?>
<etudiant>
<nom>alaoui</nom>
<prenom>mohammed</prenom>
<nom>alami</nom>
<prenom>khadija</prenom>
<nom>mhemmdi</nom>
<prenom>loubna</prenom>
</etudiant>
```

Exemple3

```
<?xml version="1.0" encoding="iso-8859-1"?>
<etudiant>
<!-- etudiant1 -->
<nom>alaoui</nom>
<prenom>mohammed</prenom>
<!-- etudiant2 -->
<nom>alami</nom>
<prenom>khadija</prenom>
<!-- etudiant3 -->
<nom>mhemmdi</nom>
<prenom>loubna</prenom>
</etudiant>
```

III.5 sections DATA

•Avec les sections littérales Il est possible de stopper l'interprétation des caractères spéciaux. La syntaxe est la suivante :
< ![CDATA[texte non soumi à l'analyse]]>
CDATA ne peuvent pas être imbriquées.

Exemple :

```
<nom>
<![CDATA[
groupe<TDI>&<TDM>
!]]>
</nom>
```

Exercice :

Création d'un annuaire téléphonique

Une société souhaite écrire un annuaire téléphonique en utilisant un document XML. L'annuaire est structuré en **personnes**. Chaque personne doit contenir **nom**, **prénom**, **email** et **téléphone personnel** & **professionnel**. Ainsi le **service** où il bosse , cet élément contient le **cadre** et le **matricule.**

Proposez un document xml bien formé.

```
<?xml version="1.0" encoding="iso-8859-1"?>
 <personnes>
 <personne>
<nom> magueri </nom>
 <prenom> salah eddine</prenom>
 <email> maguersalah@gmail.com </email>
<telephone>
<telephone-personel> 0654387388 </telephone-personel>
<telephone-professionel> 0635782511 </telephone-professionel>
</telephone>
 <service>
 <cadre> emloyer </cadre>
 <matricule> 2343 </matricule>
</service>
 </personne>
<personne>
 <nom> x</nom>
<prenom> y</prenom>
 <email> xy@gmail.com </email>
<telephone>
<telephone-personel> 0654557388 </telephone-personel>
<telephone-professionel> 0636652511 </telephone-professionel>
</telephone>
 <service>
<cadre> emloyer </cadre>
 <matricule> 2343 </matricule>
</service>
 </personne>
 </personnes>
```

III.6 les attributs :

•Un élément ouvrant peut être enrichi par des couples de la forme attribut1="valeur1" comme dans l'exemple

<produit nom="xx" prix="100">

•La valeur doit être entourée d'apostrophes si elle contient des guillemets, et inversement.

•Le nom des attributs suit les mêmes règles syntaxiques que les noms d'éléments.

xml:lang='langue' permet de définir la langue utilisée dans l'élément et tous les sous éléments.
La langue suit la norme ISO 3166 définie par la RFC 1766 (Request For Comment). Par exemple fr ou en-US ou fr-FR.

•xml:id='identificateur' permet d'associer une et une seule clef à un élément .
•xml:idref='identificateur' permet de faire référence à une clef.

IV. Validation d'un document XML : DTD (Document Type Definition)

IV.1 : Définition :

Un document XML avec une syntaxe correcte est dit bien formé C'est la garantie que n'importe quelle application peut lire sans problème un document XML.

- On peut vérifier également la conformité d'un document XML par rapport à une structure prédéfinie, on dit alors qu'un document est valide. (DTD/XSD)
- Une DTD :
- Fournit de l'information sur les données d'un document XML.
- Permet de déclarer de nouvelles balises et de spécifier des contraintes sur celles-ci.
- Permet à une application de savoir quel document XML produire et quoi lire.
- Permet de connaître ce qui est supporté (interopérabilité).
- C'est une grammaire qui décrit les éléments et les attributs acceptés dans un document XML respectant cette DTD.

Résumé des spécifications : Une DTD (grammaire) permet de déclarer :

- Un type d'élément,
- Une liste d'attribut d'un élément,
- Une entité.

- Chaque balise du langage doit faire l'objet d'une et d'une seule déclaration.

- Un document XML est dit "valide" s'il possède une DTD et si sa syntaxe est conforme aux règles de la DTD.

- Un document “valide” est obligatoirement “bien formé”.
- Bien formé →Respect la syntaxe XML.
- Valide →Respect des règles de grammaire

IV.2 : Structure d’une DTD :

```
<!ELEMENT DEVIS (IDENTIFIANT, STATUT, MONTANT, ARTICLE+)>
<!ELEMENT IDENTIFIANT (#PCDATA)>
<!ELEMENT STATUT (#PCDATA)>
<!ELEMENT MONTANT (#PCDATA )>
<!ATTLIST MONTANT devise CDATA #REQUIRED>
<!ELEMENT ARTICLE (REFERENCE, QUANTITE, PRIX_UNITAIRE, IMAGE+)>
<!ELEMENT REFERENCE (#PCDATA)>
<!ELEMENT QUANTITE (#PCDATA)>
<!ELEMENT PRIX_UNITAIRE (#PCDATA)>
<!ATTLIST PRIX_UNITAIRE devise CDATA #REQUIRED>
<!ELEMENT IMAGE EMPTY>
<!ATTLIST IMAGE href CDATA #REQUIRED>
```

Déclaration d'un élément dont le contenu est une suite d'éléments

Déclaration d'un élément dont le contenu est une suite de caractères

Déclaration d'un élément vide

Déclaration de liste d'attributs

IV.3 : Déclaration d’une DTD externe /interne :

Voir la Page 3.

DTD Externe→ <!DOCTYPE nom-elt-racine SYSTEM "test.dtd">

DTD Interne→ <!DOCTYPE nom-elt-racine [ ]>

IV.3.1 Définition des éléments :

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

nom-elt
Le nom d’élément

L’élément en XML est exprimé selon la syntaxe suivante : < !ELEMENT nom-elt contenu>

Contenu : type de données / les éléments

Le contenu peut être :

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)>

IV.3.2 les indicateurs d'occurrences :

elt?	elt est optionel
elt+	elt apparaît au moins une fois
elt*	elt apparaît entre 0 et n fois
elt1 | elt2	elt1 ou elt2
elt1, elt2	elt2 suit elt1
(elt1,elt2)+	elt1 suivi de elt2 apparaît au moins une fois
#PCDATA	données de type texte dans l'encodage courant *parsable character data*
ANY	n'importe quoi
EMPTY	vide

Elts ? : 0/1

Remarque : on peut tester si le fichier XML est validé par un XML Validator par exemple :

https://www.truugo.com/xml_validator/

https://xmlvalidation.com/

IV.3.3 Exercices :

Exercice1 : Soit le document XML suivant :

<etudiants>

<nom> ALAOUI</nom>

<prenom>Mohammed></prenom>

<matricule> 125</matricule>

<image/>

</etudiants>

1. Tester si le document XML est bien formé.
2. Proposer un DTD pour que le document XML soit valide.
3. Tester si le XML est valide.

Exercice2 : Soit le document XML suivant :

```
<personnes>
 <personne>
<nom> IDRISSI </nom>
 <prenom> Ahmed</prenom>
 <email> IA@gmail.com </email>
<telephone>
<telephone-personel> 0654387388 </telephone-personel>
<telephone-professionel> 0635782511 </telephone-professionel>
</telephone>
 <service>
 <cadre> cadre principal </cadre>
 <matricule> 2343 </matricule>
</service>
 </personne>
  </personnes>
```

1.Tester si le document XML est bien formé.

2.Proposer une DTD pour que le document XML soit valide

3.Tester si le XML est valide.

Exercice3 : Soit le DTD suivant :

```
<!DOCTYPE vehicule [
<!ELEMENT vehicule (voiture+)>
<!ELEMENT voiture (marque,couleur,matricule,carburant,boite-vitesse)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT couleur (#PCDATA)>
<!ELEMENT matricule (#PCDATA)>
```

<!ELEMENT carburant (diesel|essence)>

<!ELEMENT diesel EMPTY>

<!ELEMENT essence EMPTY>

<!ELEMENT boite-vitesse (mecanique|automatique)>

<!ELEMENT mecanique EMPTY>

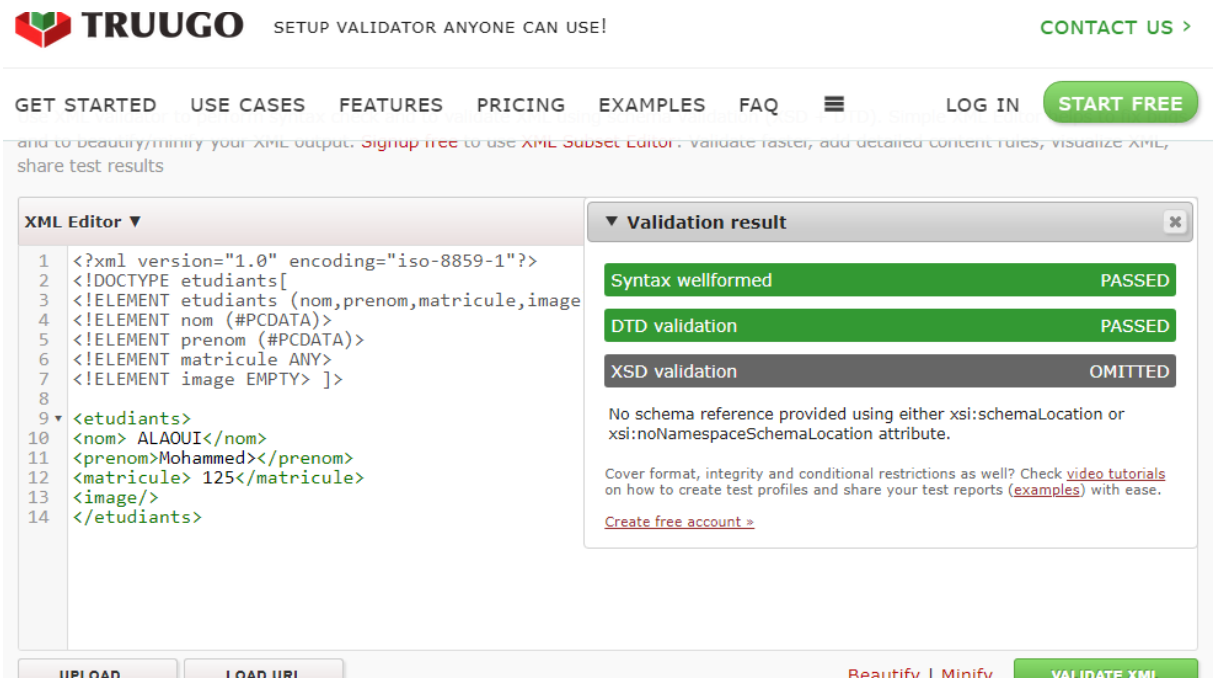
<!ELEMENT automatique EMPTY>

]>

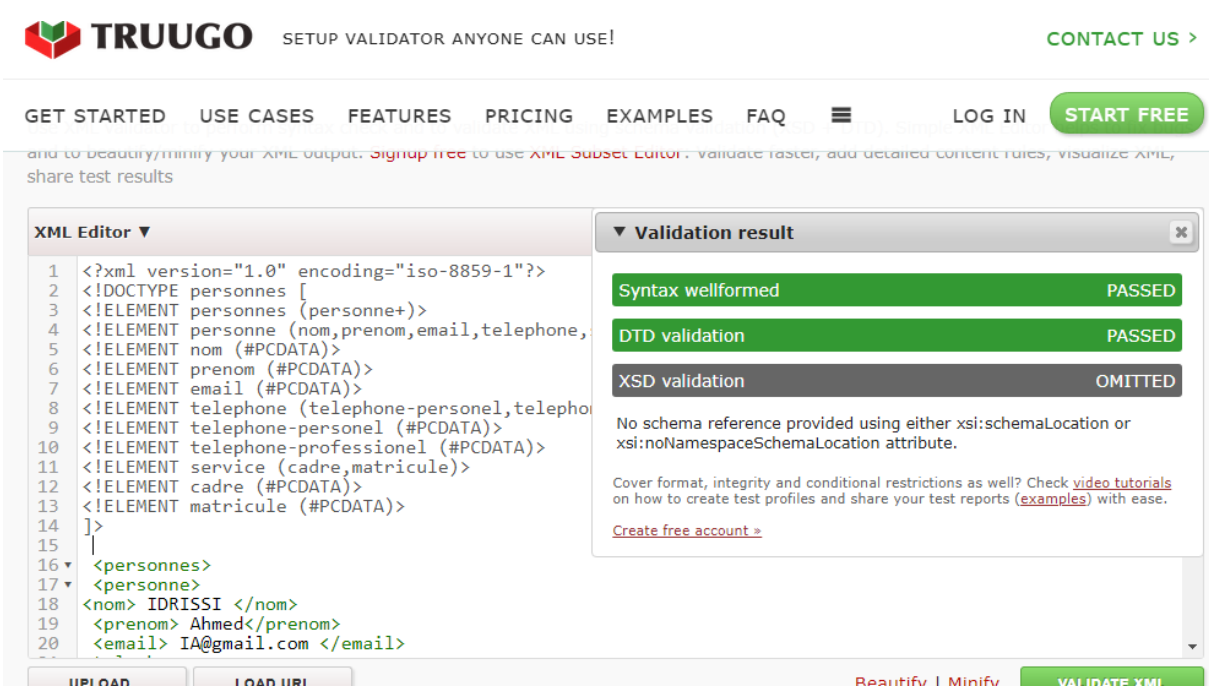
1. Proposer un document XML qui est valide selon le DTD ci-dessus.
2. Tester si le document XML est bien Formé
3. Tester si le XML est valide.

Correction :

Exercice1 :



Exercice 2 :



Exercice 3 :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE vehicule [
<!ELEMENT vehicule (voiture+)>
<!ELEMENT voiture (marque,couleur,matricule,carburant,boite-vitesse)>
<!ELEMENT marque (#PCDATA)>
<!ELEMENT couleur (#PCDATA)>
<!ELEMENT matricule (#PCDATA)>
<!ELEMENT carburant (diesel|essence)>
<!ELEMENT diesel EMPTY>
<!ELEMENT essence EMPTY>
<!ELEMENT boite-vitesse (mecanique|automatique)>
<!ELEMENT mecanique EMPTY>
<!ELEMENT automatique EMPTY>
]>
<vehicule>
```

```
<voiture>
<marque> skoda vision </marque>
<couleur> rouge</couleur>
<matricule> SV1452 </matricule>
<carburant>
<essence/>
</carburant>
<boite-vitesse>
<automatique/>
</boite-vitesse>
</voiture>
<voiture>
<marque> Ford Focus </marque>
<couleur> noir</couleur>
<matricule> FF9852 </matricule>
<carburant>
<diesel/>
</carburant>
<boite-vitesse>
<mecanique/>
</boite-vitesse>
</voiture>
</vehicule>
```

IV.4 ATTLIST (attribut) :

Définition d'attributs :

<!ATTLIST Nom-elt nom-att TYPE OBLIGATION VALEUR _PAR _DEFAUT>

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 (donc pas de blanc) ; ○ 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

IV.4. 1. Description

Nature des attributs : optionnels, obligatoires, valeur déterminée

- Optionnel sans valeur par défaut
- Optionnel avec valeur par défaut#IMPLIED
- Obligatoire #REQUIRED
- Fixe #FIXED

Exemple :

```
<!ATTLIST personneid ID #REQUIRED>
<!ATTLIST personne att1 CDATA #IMPLIED att2 CDATA #IMPLIED>
```

Types d'attributs :

- Données caractères **: CDATA**
- **Énumération** : (oui | non | peut-être)
- **ID** : identifiant pour l'élément (doit être unique dans le document)
- **IDREF, IDREFS** : référence à un ID de ce document (resp. Plusieurs références séparées par des espaces)
- **NMTOKEN, NMTOKENS** : accepte que les chaine (lettres/tiret/point...)
- **ENTITY, ENTITIES** : la valeur de l'attribut doit être le nom d'une entité déclarée dans la DTD (resp. Un ensemble d'entités séparées par des espaces)

IV.4. 2. Exemple

Exercice 1 :

Soit le DTD suivant :

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

```
<!DOCTYPE stocks [
<!ELEMENT stocks (article+)>
<!ELEMENT article (nom, prix,categorie?)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prix (#PCDATA)>
<!ELEMENT categorie (#PCDATA)>
<!ATTLIST prix devise (Euros|Dolars|DH) #IMPLIED>
<!ATTLIST nom nom-article ID #REQUIRED >]>
```

1. Proposer un document XML qui est valide selon le DTD ci-dessus.
2. Tester si le document XML est bien Formé
3. Tester si le XML est valide.

Exercice2 :

Soit le document XML suivant

```
<stagiaires>
<secteur nom="AGC" code="c01"/>
<secteur nom="NTIC" code="c02"/>
<filiere groupe="TDI" codeSecteur="NTIC">101</filiere>
<filiere groupe="TDM" codeSecteur="NTIC">201</filiere>
<filiere groupe="TSC" codeSecteur="AGC">102</filiere>
<filiere groupe="TSGE" codeSecteur="AGC">203</filiere>
</stagiaires>
```

1.Tester si le document XML est bien formé.

2.Proposer une DTD pour que le document XML soit valide

3.Tester si le XML est valide.

Correction

Exercice 1 :

<stocks>

<article>

<nom nom-article="A1">Article1</nom>

<prix devise="Euros">126</prix>

<categorie>C1</categorie>

</article>

<article>

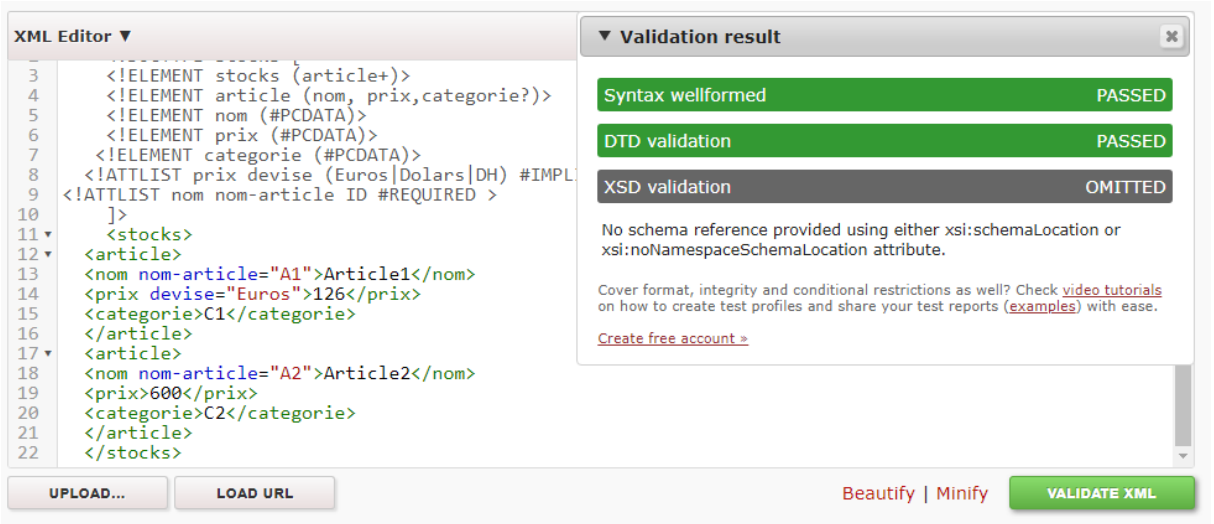
<nom nom-article="A2">Article2</nom>

<prix>600</prix>

<categorie>C2</categorie>

</article>

</stocks>



Exercice 2 :

<!DOCTYPE stagiaires[

<!ELEMENT stagiaires (secteur+,filiere+)>

<!ELEMENT filiere (#PCDATA)>

<!ELEMENT secteur EMPTY>

<!ATTLIST secteur nom ID #REQUIRED code CDATA #REQUIRED>

<!ATTLIST filiere groupe CDATA #REQUIRED codeSecteur IDREF #REQUIRED>

]>

IV.5. Entité

IV.5.1 Définition : permet d'associer un nom à un contenu ou une valeur (alias). Ce nom est employé dans le document XML comme un raccourci vers la valeur suivant la syntaxe **&nom ;.**

IV.5.2 types d'entité :

La valeur d'une entité peut être **interne** ou **externe.**

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

Syntaxe :

< !ENTITY NOM "valeur">

Exemple :

Dans le DTD →< !ENTITY msg "message">

Dans le XML→<message> veuillez lire le **&msg ;** suivant </message>

Exemple complet :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE liste-prof[
<!ELEMENT liste-prof (prof,adresse)*>
<!ELEMENT prof (nom,mail)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT mail (#PCDATA)>
<!ELEMENT adresse (#PCDATA)>
<!--entitÃ© interne-->
<!ENTITY adr "adresse du professeur est">
]>

<liste-prof>
<prof>
<nom>ppppp</nom>
<mail>bbbb@ggggg</mail>
</prof>
<!-- entitÃ© interne-->
<adresse> &adr; safi </adresse>
</liste-prof>
```

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

Exemple :

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

```
< !ENTITY e1 SYSTEM "e1.xml">
< !ENTITY e2 SYSTEM "e2.xml">
>]
< stagiaires>
&e1 ;
&e2 ;
< /stagiaires>
```

IV.5.3 entité paramétrée :

Les entités paramétriques sont les entités peuvent servir à la réalisation de la DTD pour limiter les répétitions de blocs de définition.

Syntaxe :

```
< !ENTITY %NOM  "valeur">
```

% NOM sert à utiliser une entité paramétrée dans le DTD

Exemple :

```
< !ENTITY  %valeur  "CDATA  # FIXED">
< !ATTLIST filiere  num  %valeur ;>
```

IV.6 les inconvénients du DTD

- Une DTD est non extensible (on travaille sur les instructions et pas des balises).
- Une DTD ne permet pas de typer les données (exp PCDATA)
- Une DTD ne peut prendre en compte qu'un seul espace de nom (namespace).

V. Les schémas XSD :

En réponse aux lacunes des DTD, une alternative a été proposée comme recommandation :il s'agit du **XML-schéma.** Cette nouvelle norme achève de faire d'XML un format pivot.

Les document XML-Schéma sont des documents :

- Respectant la syntaxe XML

- Permettant de décrire la structure d'un document XML d'une façon beaucoup plus complète que les DTD.
- Permet de spécifier la topologie des données que va contenir le document XML-Schéma.
- Permet de gérer une quarantaine de types de données simples
- Permet de gérer les types complexes, et les occurrences des données. (Nouveau-types/ héritage/module…)
- Pour la validation du Xml-Schema on peut utiliser les mêmes outils de validation que DTD (truugo/XMLvalidation.…)
  Autre proposition : https://www.freeformatter.com/xml-validator-xsd.html

V.1 Structure d'un XML-Schema

Le document XML :

```
<entree>
  <nom>Harry Cover</nom>
  <telephone>0102030405</telephone>
</entree>
```

Le document XML-Schema correspondant :

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema">
<xsd:element name="entree">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nom" type="xsd:string"
                   minoccurs="1" maxoccurs="1"/>
      <xsd:element name="telephone" type="xsd:decimal"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Entete

Définition de la balise complexe *entree*

Définition de la balise String *nom*

Définition de la balise de type Décimal *telephone*

V.2. Les constituants d'un XML-Schéma

Un schéma XML est un fichier xml AVEC EXTENSION **'' .xsd''**. Un schéma XML commence par le prologue puis l'élément racine.

Pour la déclaration de l'entête : l'élément **<xsd:schema>** permet de déclarer un document XML-schema.

********** FICHIER XSD**********

< !- - prologue - ->

< !- -xsd :schema xmlns :xsd='' https://www.w3.org/2001/XMLSchema **''- ->**

< !- - déclarations des éléments - ->

</xsd :schema>

xmlns :xsd='' https://www.w3.org/2001/XMLSchema: indique que les éléments de types de données utilisé dans le schéma proviennent de l'espace de noms https://www.w3.org/2001/XMLSchema. Et doivent être préfixés par **''xsd :''.**

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
            targetNamespace="http://www.annuaire.org"
            xmlns="http://www.annuaire.org"
            elementFormDefault="qualified"/>
```

L'attribut targetNamespace permet de préciser l'espace de nommage de ce type de documents.

L'attribut elementFormDefault précise si les documents XML respectant cette grammaire doivent référer à cet espace de nommage.

****FICHIER XML*****

On doit ajouter deux attributs au niveau de root-element du fichier XML.

< !- - prologue - ->

<elt-racine xmlns :xsi=''https://www.w3.org/2001/XMLSchema-instance''

Xsi :noNamespaceSchemaLocation=''test.xsd ''>

Exemple

```
<racine
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
   xsi:noNamespaceSchemaLocation="monSchema.xsd">
  ...
</racine>
```

V.2 déclarations des éléments

<xsd :element name= ''nom-elt'' type=''type-elt''>

<xsd :element name= ''prenom '' type= '' xsd :string''>

<xsd :element name= ''numero '' type= '' xsd :integer''>

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

V.2.1. Les éléments de types Simples

Un élément simple est un élément XML qui ne peut contenir que du texte (type prédéfinis/type personnalisé) .il ne peut contenir aucun autre élément ou attribut.

<etudiant> e1</etudiant> //etudiant elt simple

<etudiant><nom>oooo</nom></etudiant>etudiant n'est pas un elt simple.

```
<!– Ne contient ni attribut ni aucun autre élément => élément simple –>
<nom>ROBERT</nom>

<!– Contient un attribut => n'est pas un élément simple –>
<!– Cependant l'attribut "sexe" est un élément simple –>
<personne sexe="masculin">Robert DUPONT</personne>

<!– La balise personne contient d'autres éléments (les balises nom et prénom) => n'est pas
un élément simple –>
<personne>
  <!– Ne contient ni attribut ni aucun autre élément => élément simple –>
  <nom>DUPONT</nom>

  <!– Ne contient ni attribut ni aucun autre élément => élément simple –>
  <prenom>Robert</prenom>
</personne>
```

- **Types prédéfinis :**
  - *byte, unsignedByte, hexBinary, integer, positiveInteger, negativeInteger, int, unsignedInt, long, unsignedLong, short, unsignedShort, decimal, float, double…*
  - *string, NormalizedString, token*
  - *boolean, anyURI, language*
  - *time, dateTime, duration, date, gMonth, gYear, gYearMonth, gDay, gMonthDay*
  - *ID, IDREF, IDREFS, ENTITY, ENTITIES, NOTATIN, NMTOKEN, NMTOKENS*

Exemple : <xsd:element name="comment" type="xsd:string"/>

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

V.2.2. Les éléments de types Complexes

Un élément est dit de type complexe s'il contient autres éléments ou/et attributs.

- L'attribut contient du texte il est de type simple. Il peut être global (cad réutilisable au sein de plusieurs définitions de type complexe.

<xsd :attribute name= ''nom-attr'' type=''type-atrr''/>

<xsd : attribute name= ''service '' type= '' xsd :string''/>

<xsd : **attribute** name= ‘’id ‘’ type= ‘’ xsd :integer’’/>

Remarque : on peut ajouter l’attribut **default** pour spécifier la valeur par défaut ou l’attribut **fixed** pour spécifier une valeur fixe égale à valeur.

Ainsi on peut ajouter l’attribut **use** :**optional/required/prohibited** (interdit).

- Pour définir un type complexe on peut utiliser la balise **complexeType**

<xsd :complexeType….>

…

</xsd :complexeType>

Exemple : le type de données *TypeAdresse* se compose de 6 éléments *Numero*, *Rue1*, *Rue2*, *Ville*, *CP* et *Pays* :

```
<xsd:complexType name="TypeAdresse">
  <xsd:sequence>
   <xsd:element name="Numero"  type="xsd:positiveInteger"/>
   <xsd:element name="Rue1"    type="xsd:string"/>
   <xsd:element name="Rue2"    type="xsd:string"/>
   <xsd:element name="Ville"   type="xsd:string"/>
   <xsd:element name="CP"      type="xsd:decimal"/>
   <xsd:element name="Pays"    type="xsd: NMTOKEN" fixed="France"/>
 </xsd:sequence>
<xsd:element name="adresse" type="TypeAdresse"/>
```

Exemple complet :

Exercice1 :

//fichier XML

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

<etudiant xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="test.xsd">

<nom>alaoui</nom>

<prenom>mohammed></prenom>

</etudiant>

//fichier XSD

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="etudiant">
<xs:complexType>
<xs:sequence>
<xs:element name="nom"  type="xs:string"/>
<xs:element name="prenom"  type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

Exercice2 :

Fichier xml :

```
<?xml version="1.0" encoding="UTF-8"?>
<personne orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test1.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
```

```
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</personne>
```

Fichier xsd :

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="personne">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="address" type="xs:string"/>
            <xs:element name="city" type="xs:string"/>
            <xs:element name="country" type="xs:string"/>
          </xs:sequence>
        </xs:complexType>
```

```
    </xs:element>

    <xs:element name="item" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="title" type="xs:string"/>
       <xs:element name="note" type="xs:string" minOccurs="0"/>
       <xs:element name="quantity" type="xs:positiveInteger"/>
       <xs:element name="price" type="xs:decimal"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
</xs:schema>
```

V.2.3. Les indicateurs (pour les types complexes)

Il existe 3 types d'indicateurs : **indicateurs d'ordre** (sequence, choice, all). **Indicateurs d'occurrences (**maxOccurs,minOccurs). **Indicateurs de groupes** (xsd :group, xsd :attributeGroup) ….

- **Indicateurs d'ordre : les opérateurs de séquences**
  xsd :sequence définit un nouveau type formé d'une suite des éléments en séquence ordonnée c'est équivalent de l'opérateur ‘’,’’ des DTD.
  Exemple :

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

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

- **Indicateurs d'ordre : les opérateurs de choix xsd :choice** définit un nouveau type formé d'une suite des éléments énumérés de choix. C'est l'équivalent de l'opérateur ''|''.

  Exemple :

```
<xs:element name="vehicule">
<xs:complexType>
<xs:choice>
<xs:element name="voiture"  type="xs:string"/>
<xs:element name="camion"  type="xs:string"/>
</xs:sequence>
</xs:complexType>
</xs:element>
```

- **Indicateurs d'ordre : les opérateurs d'ensemble : xsd :all** il définit un nouveau type dont chacun des éléments doit apparaitre une fois dans un ordre quelconque.

- **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 : ?, *, + en DTD

  Les attributs **minOccurs** prend un entier comme valeur. L'attribut **maxOccurs** prend un entier ou la chaîne **unbounded (plusieurs)** comme valeur pour indiquer qu'il n'y a pas de nombre maximal.

  La valeur par default de ces deux attributs est la valeur 1.

  Exemple :

```
<xs:element name="item" maxOccurs="unbounded">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="note" type="xs:string" minOccurs="0"/>
        <xs:element name="quantity" type="xs:positiveInteger"/>
        <xs:element name="price" type="xs:decimal"/>
      </xs:sequence>
```

```
  </xs:complexType>
</xs:element>
```

- **Indicateurs de groupes :** 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 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 inclus dans un elt **xsd :sequence**, **xsd :choice** ou **xsd :all.**
  L'utilisation d'un groupe est à l'insertion de son contenu.

  Exemple :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:group name="custGroup">
  <xs:sequence>
    <xs:element name="customer" type="xs:string"/>
    <xs:element name="orderdetails" type="xs:string"/>
    <xs:element name="billto" type="xs:string"/>
    <xs:element name="shipto" type="xs:string"/>
  </xs:sequence>
</xs:group>

<xs:element name="order" type="ordertype"/>

<xs:complexType name="ordertype">
  <xs:group ref="custGroup"/>
  <xs:attribute name="status" type="xs:string"/>
</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'element **xsd :attribute**

  Exemple :

```
<xs:attributeGroup name="personattr">
  <xs:attribute name="attr1" type="string"/>
  <xs:attribute name="attr2" type="integer"/>
</xs:attributeGroup>

<xs:complexType name="person">
  <xs:attributeGroup ref="personattr"/>
</xs:complexType>
```

V.2.4. Autres éléments de types simples

- Pour définir un nouveau type simple on peut utiliser la balise **simpleType**

<xsd :simpleType….>

…

</xsd :simpleType>

- **Xsd :anyAtomicType** se sont les types prédéfinis xsd :string xsd :integer…..
- **Xsd :list** L'élément de liste définit un élément de type simple comme une liste de valeurs d'un type de données spécifié.

Exemple1 : L'exemple suivant montre un type simple qui est une liste d'entiers :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="intvalues" type="valuelist"/>

<xs:simpleType name="valuelist">
  <xs:list itemType="xs:integer"/>
</xs:simpleType>

</xs:schema>


<intvalues>100 34 56 -23 1567</intvalues>
```

Exemple 2 : L'exemple suivant montre un type simple qui est une liste de chaînes :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="stringvalues" type="valuelist"/>
<xs:simpleType name="valuelist">
  <xs:list itemType="xs:string"/>
</xs:simpleType>
</xs:schema>
----
<stringvalues>I love XML Schema</stringvalues>
```

- **Xsd :union :** L'élément union définit un type simple comme une collection (union) de valeurs à partir de types de données simples spécifiés.

Exemple :

```
<xs:element name="jeans_size">
  <xs:simpleType>
    <xs:union memberTypes="xs :string  xs :date" />
  </xs:simpleType>
</xs:element>
```

Exercice

Soit un doc xml contient un élément produit avec un prix de type union (integer/float). Donner le XSD qui permet de valider ce doc XML

```
<?xml version="1.0" encoding="UTF-8"?>
<produit xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="test2.xsd">
 <prix>5</prix>
 <prix>100.5</prix>
</produit>
```

Correction :

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="type1">
<xs:union memberTypes="xs:integer xs:float"/>
</xs:simpleType>
<xs:element name="produit">
 <xs:complexType>
  <xs:sequence>
<xs:element name="prix" type ="type1" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

- **xsd :restriction**
  Les restrictions sont utilisées pour définir des valeurs acceptables pour les éléments ou les attributs XML. Les restrictions sur les éléments XML sont appelées facettes. (Utilisée pour simpleType ou complexeType)

Restrictions sur les valeurs

Exemple : L'exemple suivant définit un élément appelé « âge » avec une restriction. La valeur de l'âge ne peut pas être inférieure à 0 ou supérieure à 120 :

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions sur un ensemble de valeurs

Pour limiter le contenu d'un élément XML à un ensemble de valeurs acceptables, nous utiliserions la contrainte d'énumération.

Exemple : L'exemple ci-dessous définit un élément appelé « car » avec une restriction. Les seules valeurs acceptables sont : Audi, Golf, BMW:

Méthode 1 : type local

```
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Méthode 2 : type global

```
<xs:element name="car" type="carType"/>

<xs:simpleType name="carType">
  <xs:restriction base="xs:string">
    <xs:enumeration value="Audi"/>
    <xs:enumeration value="Golf"/>
    <xs:enumeration value="BMW"/>
  </xs:restriction>
</xs:simpleType>
```

Restrictions sur une série de valeurs

Pour limiter le contenu d'un élément XML pour définir une série de nombres ou de lettres pouvant être utilisés, nous utiliserons la contrainte de modèle.

Exemple 1: L'exemple ci-dessous définit un élément appelé « lettre » avec une restriction. La seule valeur acceptable est l'une des lettres MINUSCULES d'un à z :

```
<xs:element name="letter">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Exemple 2: L'exemple suivant définit un élément appelé « initiales » avec une restriction. La seule valeur acceptable est trois des lettres MAJUSCULES d'a à z :

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[A-Z][A-Z][A-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Exemple 3 : L'exemple suivant définit également un élément appelé « initials » avec une restriction. La seule valeur acceptable est trois des lettres MINUSCULES OU MAJUSCULES d'a à z :

```
<xs:element name="initials">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:pattern value="[a-zA-Z][a-zA-Z][a-zA-Z]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Exemple 4 : L'exemple suivant définit un élément appelé « prodid » avec une restriction. La seule valeur acceptable est cinq chiffres dans une séquence, et chaque chiffre doit être dans une plage de 0 à 9:

```
<xs:element name="prodid">
  <xs:simpleType>
```

```
    <xs:restriction base="xs:integer">
      <xs:pattern value="[0-9][0-9][0-9][0-9][0-9]"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions sur la longueur

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

Exemple : Cet exemple définit un élément appelé « mot de passe » avec une restriction. La valeur doit être exactement huit caractères :

```
<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:minLength value="4"/>
      <xs:maxLength value="8"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>

Ou bien

<xs:element name="password">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:length value="4"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

Restrictions sur les caractères Whitespace

Pour spécifier comment les caractères whitespace doivent être manipulés, nous utiliserions la contrainte WhiteSpace. On utilisant soit la valeur « preserve »/ «

Exemple 1: Cet exemple définit un élément appelé « adresse » avec une restriction. La contrainte WhiteSpace est définie sur « **preserve** », ce qui signifie que **le processeur XML ne supprimera PAS les caractères d'espace blanc** :

```
<xs:element name="address">
  <xs:simpleType>
```

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

Exemple 2 : Cet exemple définit également un élément appelé « adresse » avec une restriction. La contrainte whiteSpace est définie sur « **`replace`** », ce qui signifie que **le processeur XML remplacera tous les caractères d'espace blanc (flux de ligne, onglets, espaces et retours de chariot) avec des espaces** :

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

Exemple 3 : Cet exemple définit également un élément appelé « adresse » avec une restriction. La contrainte whiteSpace est définie sur « **`collapse`** », ce qui signifie que **le processeur XML supprimera tous les caractères d'espace blanc (flux de ligne, onglets, espaces, retours de chariot sont remplacés par des espaces, les espaces de début et de fin sont supprimés, et plusieurs espaces sont réduits à un seul espace)** :

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

Restrictions pour les types de données

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maximum number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be equal to or greater than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be less than or equal to this value)
maxLength	Specifies the maximum number of characters or list items allowed. Must be equal to or greater than zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be greater than or equal to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be equal to or greater than zero
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

- **xsd :extension** L'élément d'extension étend un élément simpleType ou complexType existant.
- **Éléments parent: simpleContent, complexContent**

Exemple1 : L'exemple suivant étend un simpleType existant en ajoutant un attribut :

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:simpleType name="size">
  <xs:restriction base="xs:string">
    <xs:enumeration value="small" />
    <xs:enumeration value="medium" />
    <xs:enumeration value="large" />
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="jeans">
  <xs:simpleContent>
```

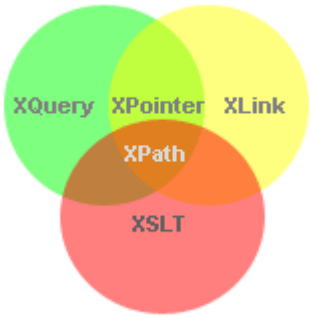
```
    <xs:extension base="size">
      <xs:attribute name="sex">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="male" />
            <xs:enumeration value="female" />
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

</xs:schema>
```

Exemple2 : L'exemple suivant étend un élément complexType existant en ajoutant trois éléments :

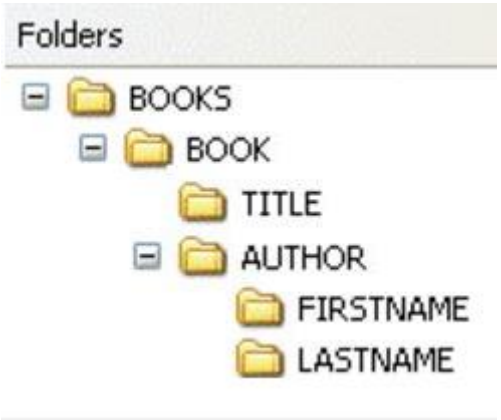
```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<xs:element name="employee" type="fullpersoninfo"/>

<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="fullpersoninfo">
  <xs:complexContent>
    <xs:extension base="personinfo">
      <xs:sequence>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="city" type="xs:string"/>
        <xs:element name="country" type="xs:string"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

</xs:schema>
```

VI.Xpath (XML path Language)

VI. 1.introduction



- XPath signifie Langage de chemin d'accès XML
- XPath utilise la syntaxe « path like » pour identifier et naviguer dans les nœuds d'un document XML
- XPath contient plus de 200 fonctions intégrées {count, contains , not, normalize-space ….}(http://www.gchagnon.fr/cours/xml/fonctionsxpath.html)
- XPath est un élément majeur de la norme XSLT
- XPath est une recommandation W3C.
- XPath utilise des expressions de chemin d'accès pour sélectionner des nœuds ou des jeux de nœuds dans un document XML.
- Ces expressions de chemin d'accès ressemblent beaucoup aux expressions de chemin que vous utilisez avec les systèmes de fichiers informatiques traditionnels :



- XPath est un élément majeur de la norme XSLT.
- Grâce à la connaissance de XPath, vous serez en mesure de tirer un grand profit de vos connaissances XSLT.

❖ **XPath est une recommandation W3C**

- XPath 1.0 est devenu une recommandation du 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 du W3C le 8 avril 2014.

VI.2 Terminologie Xpath

a) Nœud :

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

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

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
</bookstore>
```

Exemple de nœuds dans le document XML ci-dessus :

- `<bookstore>` (root element node)

  `<author>J K. Rowling</author>` (element node)

  `lang="en"` (attribute node)

b) Valeurs atomiques

Les valeurs atomiques sont des nœuds sans enfants ni parent.
Exemple de valeurs atomiques :

```
J K. Rowling

"en"
```

c) Articles

Les éléments sont des valeurs atomiques ou des nœuds.

d) Relation des nœuds

❖ **Parent**

Chaque élément et attribut a un parent.

Dans l'exemple suivant ; l'élément book est le parent du titre, de l'auteur, de l'année et du prix :

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

❖ Enfants (children)

Les nœuds d'éléments peuvent avoir zéro, un ou plusieurs enfants.
Dans l'exemple suivant ; le titre, l'auteur, l'année et les éléments de prix sont tous des enfants de l'élément book:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

❖ Frères (siblings)

Nœuds ayant le même parent.
Dans l'exemple suivant ; le titre, l'auteur, l'année et les éléments de prix sont tous frères et sœurs:

```
<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>
```

❖ Ancêtres

Le parent d'un nœud, le parent, etc.
Dans l'exemple suivant ; les ancêtres de l'élément titre sont l'élément book et l'élément bookstore:

```
<bookstore>

<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

❖ Descendants

Les enfants d'un nœud, les enfants, etc.
Dans l'exemple suivant ; descendants de l'élément bookstore sont le book, le titre, l'auteur, l'année, et les éléments de prix:

```
<bookstore>

<book>
  <title>Harry Potter</title>
  <author>J K. Rowling</author>
  <year>2005</year>
  <price>29.99</price>
</book>

</bookstore>
```

VI.3. Syntaxe XPath

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

VI.3.1.Sélection des nœuds

Les expressions de chemin d'accès les plus utiles sont répertoriées ci-dessous :

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

Nous utiliserons le document XML suivant dans les exemples ci-dessous.

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

<book>
  <title lang="en">Harry Potter</title>
  <price>29.99</price>
</book>
```

```
<book>
  <title lang="en">Learning XML</title>
  <price>39.95</price>
</book>
</bookstore>
```

Dans le tableau ci-dessous, nous avons répertorié quelques expressions de chemin d'accès et le résultat des expressions :

Remarque : Pour tester le Xpath en ligne on peut utiliser le lien suivant
http://xpather.com/

Le code XML	Expression Xpath	Résulat
<?xml version="1.0" encoding="UTF-8"?> <bookstore> <book> <title lang="en">Harry Potter</title> <price>29.99</price> </book> <book> <title lang="en">Learning XML</title> <price>39.95</price> </book> </bookstore>	/bookstore	<bookstore> <book> <title lang="en">Harry Potter</title> <price>29.99</price> </book> <book> <title lang="en">Learning XML</title> <price>39.95</price> </book> </bookstore>
	/bookstore/book	<book> <title lang="en">Harry Potter</title> <price>29.99</price> </book> <book> <title lang="en">Learning XML</title> <price>39.95</price> </book>
	//book	<book> <title lang="en">Harry Potter</title> <price>29.99</price> </book> <book> <title lang="en">Learning XML</title> <price>39.95</price> </book>
	/bookstore/book/price	<price>29.99</price> <price>39.95</price>

	//@lang	<title lang="en">Harry Potter</title> <title lang="en">Learning XML</title>
	bookstore/book/title/@lang	<title lang="en">Harry Potter</title> <title lang="en">Learning XML</title>
	bookstore/book/title/.	<title lang="en">Harry Potter</title> <title lang="en">Learning XML</title>
	bookstore/book/title/..	<book> <title lang="en">Harry Potter</title> <price>29.99</price> </book> <book> <title lang="en">Learning XML</title> <price>39.95</price> </book>

VI.3.2. Prédicats

- Les prédicats sont utilisés pour trouver un nœud spécifique ou un nœud qui contient une valeur spécifique.
- Les prédicats sont toujours intégrés entre crochets carrés.
- Dans le tableau ci-dessous, nous avons répertorié quelques expressions de chemin d'accès avec des prédicats et le résultat des expressions :

Expression Path	Description	Résultat
/bookstore/book[1]	Sélectionne le premier book	<book> <title lang="en">Harry Potter</title> <price>29.99</price> </book>
/bookstore/book[last()]	Sélectionne le dernier book	<book> <title lang="en">Learning XML</title> <price>39.95</price> </book>
/bookstore/book[last()-1]	Sélectionne l'avant dernier	<book> <title lang="en">Harry Potter</title> <price>29.99</price> </book>
/bookstore/book[position()<3]	Sélectionne les deux premiers book	<book> <title lang="en">Harry Potter</title> <price>29.99</price> </book> <book> <title lang="en">Learning XML</title> <price>39.95</price> </book>
//title[@lang]	Sélectionne tous les book dont leur attribut nommé « lang »	1. <title lang="en">Harry Potter</title> 2. <title lang="en">Learning XML</title>
//title[@lang='en']	Sélectionne tous les book dont leur attribut nommé « lang » avec la valeur « en »	1. <title lang="en">Harry Potter</title> 2. <title lang="en">Learning XML</title>

/bookstore/book[price>35.00]	Sélectionne tous les book qui ont un price>35.00	<book> <title lang="en">Learning XML</title> <price>39.95</price> </book>
/bookstore/book[price>35.00]/title	Sélectionne tous les title des book qui ont un price>35.00	<title lang="en">Learning XML</title>

VI.3.3. Sélections des nœuds inconnus

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

Caractères Génériques	Description
*	N'importe quel nœud d'élément
@*	N'importe quel nœud d'attribut
node()	N'importe quel nœud de toute sorte

Exemple :

- **/bookstore/book/* :** sélectionne tous les éléments du book.
- **//@*:** sélectionne les éléments qui ont au moins un attribut.
- **//title[@*] :** sélectionne tous les attributs de l'élément title.

VI.3.4. Sélection de plusieurs chemins d'accès

En utilisant le | dans une expression XPath, vous pouvez sélectionner plusieurs chemins d'accès. (UNION)

Dans les exemples ci-dessous, nous avons répertorié quelques expressions de chemin d'accès et le résultat des expressions :

- **.//book/title | //book/price :** sélectionne tous les éléments title Et price de l'element book.
- **//title | //price :** séletionne tous les éléments title ET price du document.
- **/bookstore/book/title | //price :** sélectionne tous les elements tiltle de l'element book ET l'element price du document

VI.3.5. Opérateurs XPath

Vous trouverez ci-dessous une liste des opérateurs pouvant être utilisés dans les expressions XPath :

Operator	Description	Example
|	Computes two node-sets	//book | //cd
+	Addition	6 + 4
-	Subtraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
=	Equal	price=9.80
!=	Not equal	price!=9.80
<	Less than	price<9.80
<=	Less than or equal to	price<=9.80
>	Greater than	price>9.80
>=	Greater than or equal to	price>=9.80
or	or	price=9.80 or price=9.70
and	and	price>9.00 and price<9.90
mod	Modulus (division remainder)	5 mod 2

Remarque :

Vous pouvez améliorer vos connaissances et vos compétences en langage web je vous propose le site web suivant c'est LE PLUS GRAND SITE DE DÉVELOPPEMENT WEB AU MONDE

https://www.w3schools.com/xml/