

# REPRÉSENTATION DES DONNÉES

---

IFT287

(Thème 2)

# Introduction

- Il arrive qu'un programme ait besoin de rendre ses données persistantes
  - Sur disque
  - Pour communiquer par le réseau
  - Etc.
- Le fait de rendre les données persistantes s'appelle la ***sérialisation***
- Le processus inverse s'appelle la ***désérialisation***

# Introduction

- Java fournit directement une façon de sérialiser et désérialiser les objets
- Dans le cours, nous ne nous attarderons pas sur ces techniques.
- Nous allons plutôt utiliser différents standards de représentation des données

# Introduction

HOW STANDARDS PROLIFERATE:  
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)



# Introduction

- XML
  - eXtensible Markup Language
- JSON
  - JavaScript Object Notation
- YAML
  - Acronyme récursif pour « YAML Ain't Markup Language » ou au début « Yet Another Markup Language »

# Introduction

- XAML
  - eXtensible Application Markup Language
  - Permet de représenter une interface logicielle pour WPF (Microsoft)
- HTML
  - HyperText Markup Language
  - Définis l'apparence et les données d'une page WEB

# XML

- Un document au format XML représente un arbre
- Chaque nœud de l'arbre est défini par une balise
- Les balises peuvent être définies par l'utilisateur
- XML fournit des mécanismes de validation
  - Pour la structure du document
  - Pour les types de données

# XML

- Un document XML est sensible à la casse
  - `<Membres>` et `<membres>` sont deux balises différentes
- L'encodage des caractères n'est pas restreint à l'ASCII
- Un document XML peut contenir des commentaires
  - Débutant par `<!--`
  - Et terminant pas `-->`
  - Les commentaires peuvent être insérés n'importe où dans le document



# XML - Entête

- Un document XML débute toujours par cette ligne

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
```

- *version* indique la version du standard à utiliser
- *encoding* indique le jeu de caractères (ASCII, UTF8, etc.)
- *standalone* indique si le document est indépendant
  - Doit être *no* si le document réfère un DTD

# XML - Exemple

```
<?xml version="1" encoding="UTF8" standalone="yes"?>
<Bibliotheque>
  <Auteur id="1" nom="J.K. Rowling"/>
  <Livre>
    <Titre>Harry Potter à l'école des sorciers</Titre>
    <Auteur>1</Auteur>
    <Description>Un jeune sorcier orphelin apprend qu'il est un
                  sorcier et passe sa première année à Poudlard.
    </Description>
    <!--Ce commentaire n'est qu'un exemple-->
  </Livre>
  <Livre>
    ...
  </Livre>
</Bibliotheque>
```

# XML - Corps

```
<?xml version="1" encoding="UTF8" standalone="yes"?>
```

## **<Bibliotheque>**

```
<Auteur id="1" nom="J.K. Rowling"/>
```

```
<Livre>
```

```
<Titre>Harry Potter à l'école des sorciers</Titre>
```

```
<Auteur>1</Auteur>
```

```
<Description>Un jeune sorcier orphelin apprend qu'il est un  
sorcier et passe sa première année à Poudlard.
```

```
</Description>
```

```
<!--Ce commentaire n'est qu'un exemple-->
```

```
</Livre>
```

```
<Livre>
```

```
...
```

```
</Livre>
```

## **</Bibliotheque>**

# XML - Corps

- Le nom d'une balise
  - Commence par une lettre
  - Suivi d'une suite de lettre ou de chiffre

`<Bibliotheque>`

...

`</Bibliotheque>`

`<B123A56>`

...

`</B123A56>`

# XML - Corps

```
<?xml version="1" encoding="UTF8" standalone="yes"?>
<Bibliotheque>
  <Auteur id="1" nom="J.K. Rowling"/>
  <Livre>
    <Titre>Harry Potter à l'école des sorciers</Titre>
    <Auteur>1</Auteur>
    <Description>Un jeune sorcier orphelin apprend qu'il est un
                  sorcier et passe sa première année à Poudlard.
    </Description>
    <!--Ce commentaire n'est qu'un exemple-->
  </Livre>
  <Livre>
    ...
  </Livre>
</Bibliotheque>
```

# XML - Corps

- La valeur d'un attribut est délimitée par
  - De simples guillemets ( ' )
  - Des doubles guillemets ( " )
- La valeur d'un attribut **ne doit pas** contenir
  - Le symbole <
  - Le symbole &
- Il est possible d'inclure des guillemets simples si la valeur est délimitée par des guillemets doubles, et vice-versa

# XML - Corps

- Pour ajouter les caractères interdits, on utilise les entités

Entité	Caractère
<code>&amp;amp;</code>	<code>&amp;</code>
<code>&amp;lt;</code>	<code>&lt;</code>
<code>&amp;gt;</code>	<code>&gt;</code>
<code>&amp;quot;</code>	<code>"</code>
<code>&amp;apos;</code>	<code>'</code>

# XML - Corps

```
<?xml version="1" encoding="UTF8" standalone="yes"?>
<Equations>
  <Equation id="1" condition="x<b>lt;y&amp;&amp;y&gt;x"/>
  <Equation id="2" description="Il faut l<b>apos;enlever"/>
</Equations>
```



# XML - Corps

- Le contenu d'un élément comprend tous les sous-éléments, ainsi que le texte, présents entre la balise de début et la balise de fin
- Lorsqu'un élément n'a aucun contenu, il n'est pas obligatoire d'avoir une balise de fin
- Le texte contenu dans un élément **ne peut pas** contenir
  - <
  - &
  - ]]>

# XML - DTD

- Il est possible de valider le format d'un fichier XML à l'aide d'un DTD (*Document Type Definition*)
- Le DTD permet
  - De définir les balises du document
  - De définir la structure et les relations entre les balises
- Un fichier XML peut
  - Contenir la définition d'un DTD
  - Faire référence à un DTD

# XML - DTD

```
<?xml version= "1.0" encoding="ISO-8859-1"?>
<!-- DTD de la bibliothèque -->
<!ELEMENT biblio (livre|membre)*>
<!-- Livre -->
<!ELEMENT livre (idLivre, titre, auteur, dateAcquisition,
                 idMembre?, datePret?)>
<!ELEMENT idLivre (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT dateAcquisition (#PCDATA)>
<!ELEMENT datePret (#PCDATA)>
<!-- membre -->
<!ELEMENT membre (idMembre, nom, telephone, limitePret, nbPret)>
<!ELEMENT idMembre (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
<!ELEMENT limitePret (#PCDATA)>
<!ELEMENT nbPret (#PCDATA)>
```

# XML - DTD

```
<?xml version= "1.0" encoding="ISO-8859-1"?>
<!-- DTD de la bibliothèque -->
<!ELEMENT biblio (livre|membre)*>
<!-- Livre -->
<!ELEMENT livre (idLivre, titre, auteur, dateAcquisition,
                 idMembre?, datePret?)>
<!ELEMENT idLivre (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT dateAcquisition (#PCDATA)>
<!ELEMENT datePret (#PCDATA)>
<!-- membre -->
<!ELEMENT membre (idMembre, nom, telephone, limitePret, nbPret)>
<!ELEMENT idMembre (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
<!ELEMENT limitePret (#PCDATA)>
<!ELEMENT nbPret (#PCDATA)>
```

← Racine du document XML

# XML - DTD

- Chaque élément définit ses sous-éléments à l'aide d'opérateurs

Opérateur	Définition
#PCDATA	Suite de caractères
x	Un sous-élément x
?	Zéro ou une occurrence
*	Zéro ou plusieurs occurrences
+	Une ou plusieurs occurrences
$e_1   \dots   e_n$	Choix entre les expressions
$e_1, \dots, e_n$	Séquence d'expression

# XML - DTD

```

<?xml version= "1.0" encoding="ISO-8859-1"?>
<!-- DTD de la bibliothèque -->
<!ELEMENT biblio (livre|member)*>
<!-- Livre -->
<!ELEMENT livre (idLivre, titre, auteur, dateAcquisition,
                  idMembre?, datePret?)>
<!ELEMENT idLivre (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (#PCDATA)>
<!ELEMENT dateAcquisition (#PCDATA)>
<!ELEMENT datePret (#PCDATA)>
<!-- membre -->
<!ELEMENT membre (idMembre, nom, telephone, limitePret, nbPret)>
<!ELEMENT idMembre (#PCDATA)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT telephone (#PCDATA)>
<!ELEMENT limitePret (#PCDATA)>
<!ELEMENT nbPret (#PCDATA)>

```

Choix entre livre ou membre

Zéro ou plusieurs fois

Zéro ou une seule fois (donc optionnel)

Du texte brut

Un nom suivi d'un téléphone

# XML - DTD

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!ELEMENT biblio (livre|membre)*>
<!ELEMENT livre EMPTY>
<!ATTLIST livre
    idLivre CDATA #REQUIRED
    titre CDATA #REQUIRED
    auteur CDATA #REQUIRED
    dateAcquisition CDATA #REQUIRED
    idMembre CDATA #IMPLIED
    datePret CDATA #IMPLIED>
<!ELEMENT membre EMPTY>
<!ATTLIST membre
    idMembre CDATA #REQUIRED
    nom CDATA #REQUIRED
    telephone CDATA #REQUIRED
    limitePret CDATA #REQUIRED
    nbPret CDATA #REQUIRED>
```

# XML - DTD

- Chaque attribut possède un type et une valeur par défaut

Type	Définition
CDATA	Suite de caractères
ID	Nom unique pour tout le document XML
IDREF	Référence à un ID
( $v_1$   ...   $v_n$ )	Énumération des valeurs possibles

Valeur par défaut	Définition
#REQUIRED	L'attribut est obligatoire
#IMPLIED	L'attribut est optionnel
#FIXED $v$	L'attribut est optionnel, mais si présent, la valeur est $v$
$v$	L'attribut est optionnel et sa valeur par défaut est $v$



# XML - DTD

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

```
<!ELEMENT terre (personne)*>
```

```
<!ELEMENT personne EMPTY>
```

```
<!ATTLIST personne
```

```
  nom ID #REQUIRED
```

L'attribut nom est obligatoire, et la valeur est unique dans le document XML

```
  surnom CDATA #IMPLIED
```

```
  etat (vivant | mort) "vivant"
```

L'attribut est optionnel, peut prendre la valeur vivant ou mort, et s'il n'est pas présent, la valeur est vivant

```
  parrain IDREF #IMPLIED
```

L'attribut parrain est optionnel, et il réfère à un ID unique

```
  sexe (masculin | feminin ) #REQUIRED
```

```
  origine CDATA #FIXED "terrien" >
```

L'attribut est optionnel, mais s'il est présent, la valeur doit être terrien

# XML - DTD

```
<?xml version="1.0" encoding="ISO-8859-1" Standalone="no"?>
<!DOCTYPE terre SYSTEM "terre.dtd">
<terre>
  <personne
    nom="Paul"
    surnom="Polo"
    etat="mort"
    sexe="masculin"
    origine="terrien"/>
  <personne
    nom="Jean"
    parrain="Paul"
    sexe="masculin"
  />
</terre>
```

# XML - Schéma

- Il est possible de valider le format d'un fichier XML à l'aide d'un schéma
- Le schéma permet
  - De définir des types et de les assigner à des éléments
  - De définir la structure et les relations entre les éléments
  - De définir plus en détail les types de données
- Dans un schéma, un type décrit la structure d'un élément
  - Ses attributs
  - Ses sous-éléments

# XML - Schéma

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema
    targetNamespace="urn:ift287:biblio:element"
    xmlns="urn:ift287:biblio:element"
    elementFormDefault="qualified">
  <xsd:element name="biblio" type="Biblio"/>
  <xsd:complexType name="Biblio">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="livre" type="Livre"/>
      <xsd:element name="membre" type="Membre"/>
    </xsd:choice>
  </xsd:complexType>
  <xsd:complexType name="Livre">
    <xsd:attribute name="titre" type="xsd:string" use="required"/>
    <xsd:attribute name="idLivre" type="xsd:integer" use="required"/>
    <xsd:attribute name="datePret" type="xsd:date" use="optional"/>
  </xsd:complexType>
</xsd:schema>
```

# XML - Schéma

```
<xsd:schema xmlns:xsd=http://www.w3.org/2001/XMLSchema  
  targetNamespace="urn:ift287:biblio:element"  
  xmlns="urn:ift287:biblio:element"  
  elementFormDefault="qualified">
```

- Tous les éléments associés au schéma seront utilisés avec un préfixe défini par

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

- Voir les notes de cours pour la description des autres attributs

# XML - Schéma

- Il existe deux catégories de types dans un schéma
  - Simple
  - Complexe
- Un type simple est utilisé pour
  - Un élément sans attributs ni sous-élément
  - Un attribut d'un élément
- Des types simples sont prédéfinis pour les « *types de base* »

# XML - Schéma

Type	Exemple de valeurs
string	abc123
integer	..., -1, 0, 1, ...
long	-9223372036854775808,...,-1,0,1,...,92233722036854775807
int	-2147483648, ..., -1, 0, 1, ..., 2147483647
short	-32768, ..., -1, 0, 1, 32767
byte	-128, ..., -1, 0, 1, 127
decimal	-1.23, 0, 123,4, 1000.0
float	-INF, -1 <sup>E</sup> 4, -0, 0, 12.78 <sup>E</sup> -2, 12, INF, NaN (32 bits)
double	-INF, -1 <sup>E</sup> 4, -0, 0, 12.78 <sup>E</sup> -2, 12, INF, NaN (64 bits)
boolean	true, false, 1, 0
duration	P1Y2M3DT10H30M12.3S = 1 an, 2 mois, 3 jours, 10 heures...
dateTime	1999-05-31T13:20:00.000-05:00 = 31 mai 1999, 13h20, -5GMT
date	1999-05-31
time	13:20:00.000, 13:20:00.000-05:00

# XML - Schéma

- Il est possible de déclarer de nouveaux types simples

```
<xsd:simpleType name="integerMinMax">  
  <xsd:restriction base="xsd:integer">  
    <xsd:minInclusive value="0"/>  
    <xsd:maxInclusive value="10"/>  
  </xsd:restriction>  
</xsd:simpleType>
```

← Indique le type de base

→ Ajoute des restrictions sur le type (facettes)

- Regardez les notes et la documentation pour les détails sur les facettes




# XML - Schéma

- Il est possible de déclarer des types complexes contenant des sous-éléments

Indique que le type est composé d'un choix entre les deux éléments qui suivent

```
<xsd:complexType name="Biblio">  
  <xsd:choice minOccurs="0" maxOccurs="unbounded">  
    <xsd:element name="livre" type="Livre"/>  
    <xsd:element name="membre" type="Membre"/>  
  </xsd:choice>  
</xsd:complexType>
```



# XML - Schéma

- Un type complexe est construit à l'aide de
- `xsd:choice` : équivalent à l'opérateur `|` d'un DTD
- `xsd:sequence` : équivalent à l'opérateur `,` d'un DTD
- `xsd:all` : un ensemble de sous-éléments, chacun apparaissant au plus une seule fois, dans un ordre arbitraire

# XML - Schéma

- Les attributs `minOccurs` et `maxOccurs` permettent de reproduire le comportement des symboles « \*, +, ? » d'un DTD

DTD	Schéma	
	<code>minOccurs</code>	<code>maxOccurs</code>
*	0	unbounded
+	1	unbounded
?	0	1

# XML - Schéma

- Il est possible de déclarer des types complexes qui possèdent des attributs

Le type est toujours  
un type simple



```
<xsd:complexType name="Livre">  
  <xsd:attribute name="titre"      type="xsd:string"          use="required"/>  
  <xsd:attribute name="idLivre"    type="xsd:integerMinMax"   use="required"/>  
  <xsd:attribute name="datePret"   type="xsd:date"            use="optional"/>  
</xsd:complexType>
```

- La balise `attribute` ne peut apparaître que dans un type complexe

# XML - Schéma

- Il n'est pas obligatoire de nommer les types
  - Utile lorsqu'un type n'est utilisé que pour un élément
- Il faut spécifier le type directement dans la déclaration de l'élément

```
<xsd:element name="biblio">
  <xsd:complexType>
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
      <xsd:element name="livre" type="Livre"/>
      <xsd:element name="membre" type="Membre"/>
    </xsd:choice>
  </xsd:complexType>
</xsd:element>
```

# XML - Schéma

- Pour utiliser un schéma dans un fichier XML

```
<?xml version="1.0" encoding="ISO-8859-1"?>  
<biblio xmlns="urn:ift287:biblio:element"  
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
        xsi:schemaLocation="urn:ift287:biblio:element  
                            file:../biblio-element.xsd">
```



Le fichier du schéma (ou  
l'URL s'il est en ligne)

# JSON

- JSON est un autre format pour échanger des données
- Comme XML, il est facile à lire pour des humains
- Il est facile à analyser et à construire pour un programme informatique
- Comme XML, JSON est indépendant du langage de programmation utilisé par les programmes qui l'utilisent

# JSON

- Le format JSON est basé sur deux structures de données
  - Une collection de clés/valeurs
  - Une liste ordonnée de valeurs

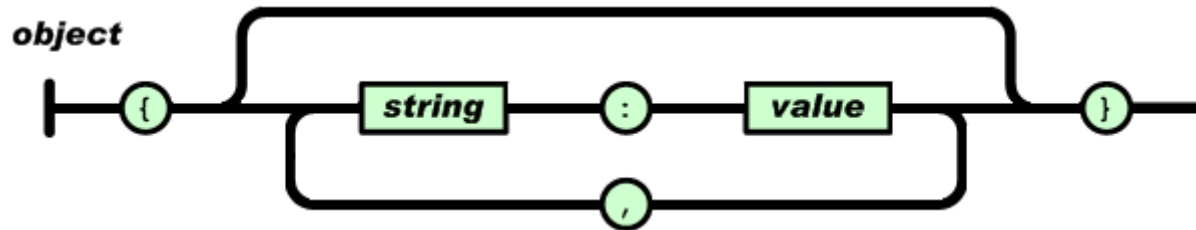


# JSON

- Un fichier JSON contient les éléments suivants, tous basés sur les deux structures précédentes
  - Un Object
  - Un Array
  - Une Value
  - Une String
  - Un Number

# JSON

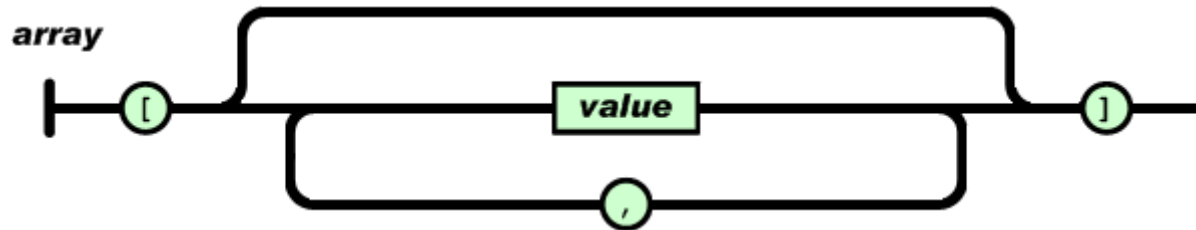
- Un `object` est défini par un ensemble non ordonné de clés/valeurs



- Un `object` débute par { suivi de zéro ou plusieurs ensembles de clés/valeurs, séparés par des virgules et terminé par }
- Chaque clé/valeur s'écrit sous la forme de « clé : valeur »

# JSON

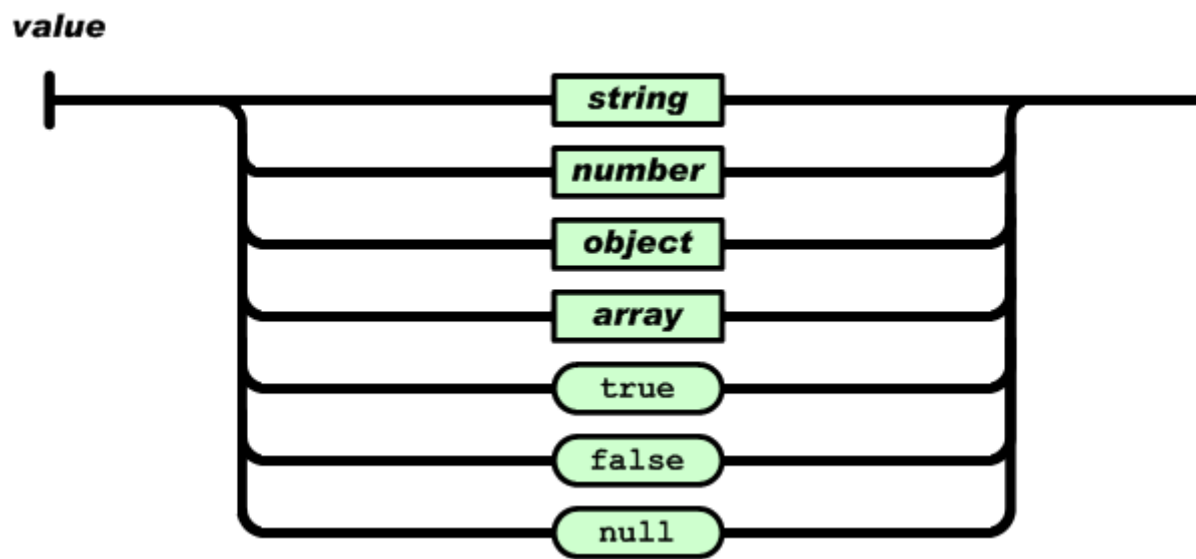
- Un `array` est une collection ordonnée de clé/valeur



- Un `array` débute par `[` suivi de zéro ou plusieurs valeurs, séparées par des virgules et terminé par `]`

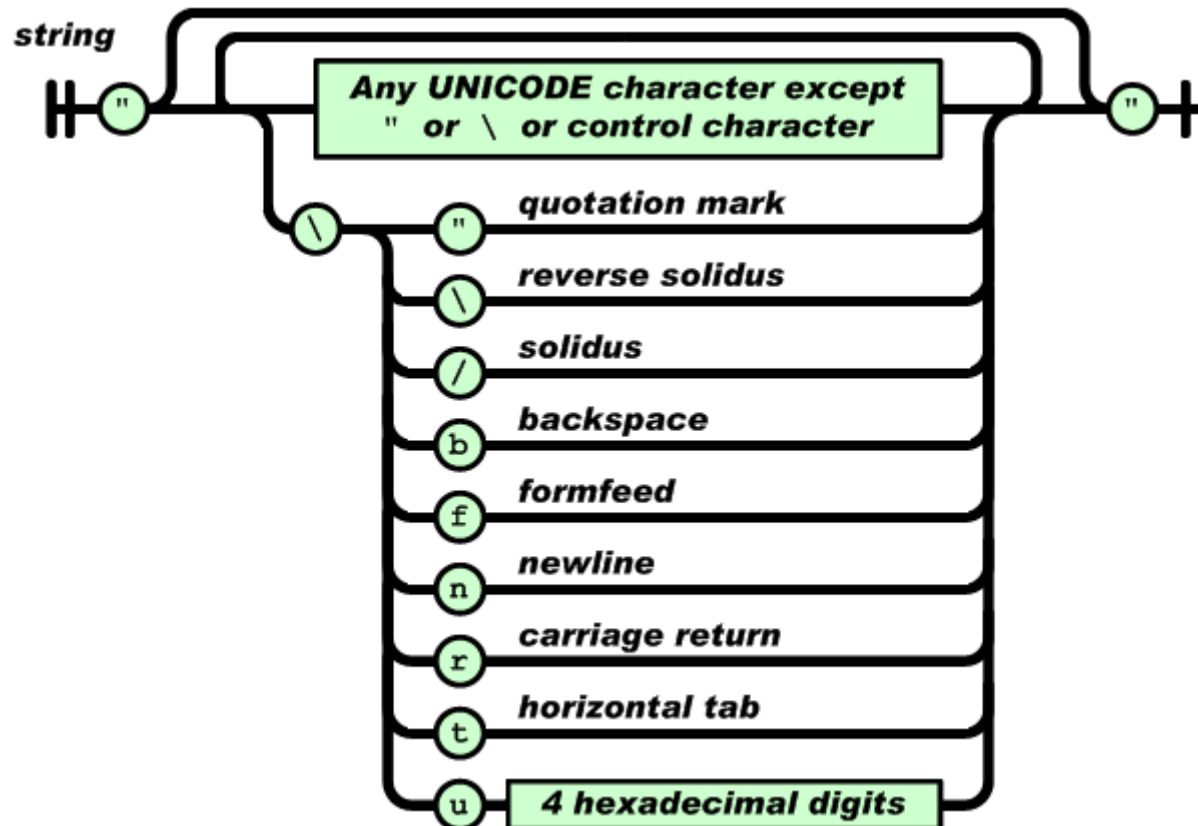
# JSON

- Une `value` peut être une `string` (entre guillemets doubles), un `number`, les mots clés `true`, `false` ou `null`, un `object` ou un `array`. Les structures peuvent être imbriquées.



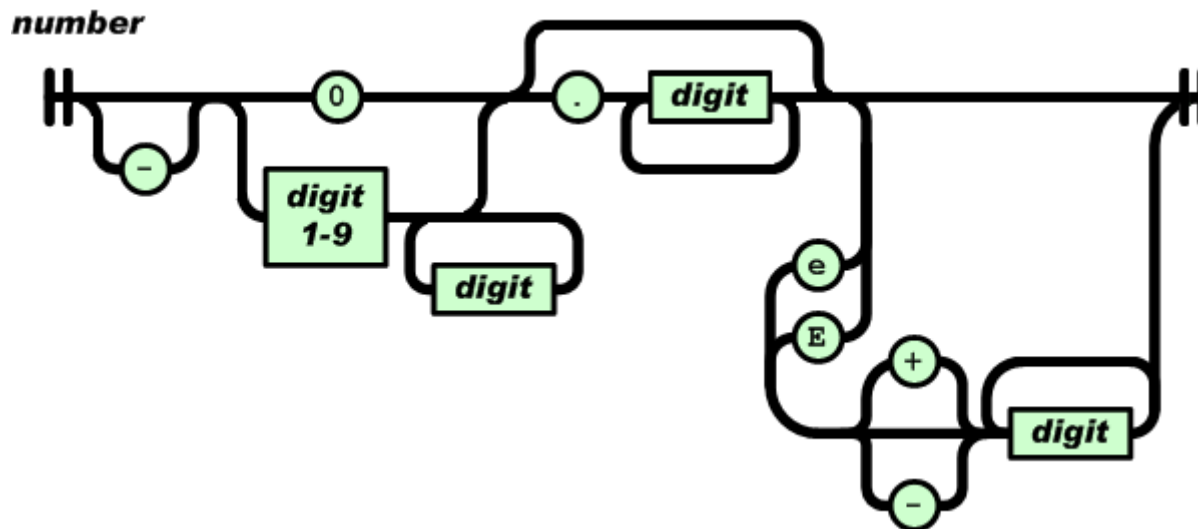
# JSON

- Une `string` est très semblable aux chaînes de caractères de C++ et Java



# JSON

- Un `number` est la même chose qu'en C++ ou Java
- Il n'y a cependant pas de représentation binaire, octale ou hexadécimale



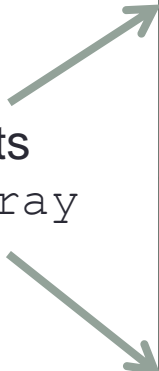
# JSON

```
{ "biblio" : [  
  { "idLivre" : 1,  
    "titre" : "Harry Potter",  
    "auteur" : "J.K. Rowling",  
    "description" : "Un jeune sorcier orphelin apprend  
                    qu'il est un sorcier et passe sa  
                    première année à Poudlard." },  
  { "idLivre" : 2,  
    "titre" : "Le hobbit",  
    "auteur" : "J.R.R. Tolkien",  
    "description" : "Bilbon Saquet part à l'aventure  
                    avec 13 nains et le sorcier Gandalf  
                    afin de reprendre la montagne  
                    solitaire." }  
]  
}
```

# JSON

```
{ "biblio" : [  
  { "idLivre" : 1,  
    "titre" : "Harry Potter",  
    "auteur" : "J.K. Rowling",  
    "description" : "Un jeune sorcier orphelin apprend  
                    qu'il est un sorcier et passe sa  
                    première année à Poudlard." },  
  { "idLivre" : 2,  
    "titre" : "Le hobbit",  
    "auteur" : "J.R.R. Tolkien",  
    "description" : "Bilbon Saquet part à l'aventure  
                    avec 13 nains et le sorcier Gandalf  
                    afin de reprendre la montagne  
                    solitaire." }  
]  
}
```

Éléments  
d'un array





# JSON

```
{  
  "firstName" : "John",  
  "lastName" : "Smith",  
  "isAlive" : true,  
  "age" : 25,  
  "address" : { "streetAddress" : "21 2nd Street",  
                "city" : "New York",  
                "state" : "NY",  
                "postalCode" : "10021-3100" },  
  "phoneNumbers" : [ { "type" : "home",  
                        "number" : "212 555-1234" },  
                      { "type" : "office",  
                        "number" : "646 555-4567" } ],  
  "children" : [],  
  "spouse" : null  
}
```

# JSON - Schéma

- Comme en XML, permet de valider la structure d'un fichier JSON
- Comme en XML, la structure d'un fichier de schéma JSON est écrite dans le langage qu'il valide
- Un schéma commence toujours par la définition d'un objet qui représente le schéma

# JSON - Schéma

```
{  
  "title" : "Example Schema",  
  "type" : "object",  
  "properties" : { "firstName" : { "type" : "string" },  
                   "lastName" : { "type" : "string" },  
                   "age" : {  
                     "description" : "Age in years",  
                     "type" : "integer",  
                     "minimum" : 0 }  
                 },  
  "required" : ["firstName", "lastName"]  
}
```

# JSON - Schéma

- Le champ `type` peut avoir une des valeurs suivantes
  - `"array"`
  - `"boolean"`
  - `"integer"`
  - `"number"`
  - `"null"`
  - `"object"`
  - `"string"`
- Vous pouvez aller voir la documentation pour connaître les autres propriétés possibles

# Comparatif

- XML et JSON permettent tous les deux de représenter des structures de données complexes issues d'un programme
- Les deux sont indépendants du langage de programmation utilisé dans les applications
- Les deux formats ont des procédés pour valider la structure du document

# Comparatif

- Alors quel format est le meilleur?
- Dans quelle circonstance est-il plus pratique d'utiliser un format plus que l'autre?
- Est-ce que la taille des fichiers XML et JSON sont très différents?

# Comparatif

```
{  
  "firstName": "John",  
  "lastName": "Smith",  
  "age": 25,  
  "address": { "streetAddress": "21 2nd Street",  
               "city": "New York",  
               "state": "NY",  
               "postalCode": "10021-3100" },  
  "phoneNumber": [ { "type": "home",  
                     "number": "212 555-1234" },  
                   { "type": "fax",  
                     "number": "646 555-4567" } ],  
  "gender": { "type": "male" }  
}
```

# Comparatif

```
<person>
  <firstName>John</firstName>
  <lastName>Smith</lastName>
  <age>25</age>
  <address>
    <streetAddress>21 2nd Street</streetAddress>
    <city>New York</city>
    <state>NY</state>
    <postalCode>10021-3100</postalCode>
  </address>
  ...
```



# Comparatif

...

```
<phoneNumbers>
  <phoneNumber>
    <type>home</type>
    <number>212 555-1234</number>
  </phoneNumber>
  <phoneNumber>
    <type>fax</type>
    <number>646 555-4567</number>
  </phoneNumber>
</phoneNumbers>
<gender>
  <type>male</type>
</gender>
</person>
```

# Comparatif

```
<person firstName="John" lastName="Smith" age="25">  
  <address streetAddress="21 2nd Street"  
    city="New York"  
    state="NY"  
    postalCode="10021-3100" />  
  <phoneNumbers>  
    <phoneNumber type="home" number="212 555-1234"/>  
    <phoneNumber type="fax" number="646 555-4567"/>  
  </phoneNumbers>  
  <gender type="male"/>  
</person>
```

# Documentation

- XML
  - <http://www.w3schools.com/xml/>
  - [http://www.w3schools.com/xml/xml\\_what\\_is.asp](http://www.w3schools.com/xml/xml_what_is.asp)
  - <https://www.w3.org/XML/>
- JSON
  - <http://json.org/>
  - <http://json-schema.org/>