XML : Introduction

- L'*Extensible Markup Language* (XML), « langage de balisage extensible » en français.
- XML est un métalangage informatique de balisage générique qui dérive du SGML.
- XML a été créé par l'organisme W3C.

XML : Exemple

```xml
<?xml version="1.0" encoding="UTF-8"?>
<vehicule>
    <type>4 Roues</type>
    <constructeur>Quadro</constructeur>
    <annee>2018</annee>
    <couleur>Rouge</couleur>
    <modele>Scooter</modele>
    <matricule>9999AB</matricule>
</vehicule>
```

XML : Règles

L’entête :

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

Le document doit contenir au moins une balise.

Le nom de chaque balise doit respecter les règles suivantes :

- Le nom peut contenir des lettres, des chiffres ou d'autres caractères.
- Il ne doit pas commencer par un chiffre ou un signe de ponctuation.
- Il ne doit pas non plus commencer par xml, Xml, XML...
- Aucun espace dans le nom d'une balise n'est toléré.

XML : Règles

- Chaque balise ouvrante doit avoir une balise fermante :

  Exemple : <exemple></exemple>

- Si la balise est vide, <exemple></exemple> par exemple, une balise abrégée peut être utilisée en lieu et place, comme ceci : <exemple/>
- Les balises ne peuvent se chevaucher.

  Exemple : <data><element></data></element>
- Les balises sont sensibles à la casse.
- Exemple : <ELEMENT><element>

XML : Règles

- Les valeurs des attributs des balises doivent **obligatoirement** être encadrées avec des quottes simples/doubles.
- Toutes les balises du document doivent obligatoirement être contenues dans une balise unique englobant tout le document. Cette balise s'appelle également la balise racine. Exemple :

```
<data>
   <element attribut="valeur">élément 1</element>
</data>
```

XML : Règles

Les caractères spéciaux interdits sont :

< (Doit être remplacé par<)

> (Doit être remplacé par>)

& (Doit être remplacé par&)

En revanche, il est possible d'utiliser une balise spéciale qui indique aux interpréteurs de ne pas tenir compte des caractères spéciaux. Exemple :

```
<description>
      <![CDATA[Dans cette description, "l'interdit" est bravé <o_o>.]]>
</description>
```

XML : Attributs

```xml
<personne sexe="F">
    <prenom>Sara</prenom>
    <nom>Smith</nom>
</personne>
```

```xml
<personne>
    <sexe>F</sexe>
    <firstname>Sara</firstname>
    <lastname>Smith</lastname>
</personne>
```

XML : Exercice

Créez un fichier XML nommé iut.xml, dont l'élément racine est **iut**. Les éléments principaux nommés **etudiant** ont comme attributs id (numéro d'inscription) et nom. Chaque élément **etudiant** peut contenir autant d'éléments **uv** que désiré. Chaque UV doit avoir un nom, une durée et une note enregistrés dans des sous-éléments. Visualisez ce fichier dans un navigateur pour vérifier qu'il est bien formé.

XML : Corrigé

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<iut>
    <etudiant id="123" nom="Lemaire">
        <uv>
            <nom>Programmation PHP 5</nom>
            <duree>4 semaines</duree>
            <note>14</note>
        </uv>
        <uv>
            <nom>MySQL</nom>
            <duree>6 semaines</duree>
            <note>16</note>
        </uv>
    </etudiant>
<etudiant id="345" nom="Legros">
        <uv>
            <nom>Programmation PHP 5</nom>
            <duree>4 semaines</duree>
            <note>12</note>
        </uv>
        <uv>
            <nom>SQLite</nom>
            <duree>6 semaines</duree>
            <note>16</note>
        </uv>
    </etudiant>
</iut>
```

JSON : Introduction

- JSON: JavaScript Object Notation.
- JSON permet le stockage et d'échange de données.
- JSON est écrit avec la notation d'objet JavaScript.

JSON : Règles

La syntaxe JSON est dérivée de la syntaxe de notation d'objet JavaScript :

- Les données sont dans des paires clé/valeur.
- Les données sont séparées par des virgules.
- Les accolades contiennent des objets.
- Les crochets contiennent des tableaux

JSON : Règles

data.json

```json
{"employees":[
    { "firstName":"John", "lastName":"Doe" },
    { "firstName":"Anna", "lastName":"Smith" },
    { "firstName":"Peter", "lastName":"Jones" }
]}
```

data.xml

```xml
<employees>
  <employee>
    <firstName>John</firstName> <lastName>Doe</lastName>
  </employee>
  <employee>
    <firstName>Anna</firstName> <lastName>Smith</lastName>
  </employee>
  <employee>
    <firstName>Peter</firstName> <lastName>Jones</lastName>
  </employee>
</employees>
```

JSON : Règles

Type	Exemple
String	`{"name" : "John"}`
Number	`{"age" : 30}`
Object	`{"employee" : {"name" : "John", "age" : 30, "city" : "New York"}}`
Array	`{"employees" : [ "John", "Anna", "Peter" ]}`
Boolean	`{"sale" : true}`
Null	`{"middlename" : null}`

JSON : Exercice

Transformer la ressource ci-dessous en JSON :

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<person>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <age>22</age>
    <address>
        <streetAddress>21 2nd Street</streetAddress>
        <city>New York</city>
        <state postalCode="10021" name="NY"/>
    </address>
    <phoneNumber type="home">
        212 555-1234
    </phoneNumber>
    <phoneNumber type="fax">
        646 555-4567
    </phoneNumber>
</person>
```

JSON : Corrigé

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

L’Ajax : INTRO

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <p id="hello"></p>
6  <script>
7
8  class MyClass {
9      constructor(){
10         this.onload = null;
11         this.responseText = "Hello tout le monde!";
12     }
13 }
14
15 m = new MyClass();
16 m.onload = function(){
17     document.getElementById("hello").innerHTML = this.responseText;
18 }
19 m.onload() ; // au chargement de la page le script sera exécuté (onLoad)
20 </script>
21 </body>
22 </html>
```

L’Ajax : XMLHttpRequest

```
<!DOCTYPE html>
<html>
<meta charset="utf-8"/>
<body>
<script>

// Create an XMLHttpRequest object
var xmlhttp = new XMLHttpRequest();

// Define a callback function
xmlhttp.onload = function() {
  let data = this.responseText;
  console.log(data);
}

// Send a request
xmlhttp.open("GET", "https://geo.api.gouv.fr/regions");
xmlhttp.send();

</script>
</body>
</html>
```

L’Ajax : Parsing JSON

```
1  <!DOCTYPE html>
2  <html>
3  <meta charset="utf-8"/>
4  <body>
6  <script>
7
8  // Create an XMLHttpRequest object
9  var xmlhttp = new XMLHttpRequest()
10
11 // Define a callback function
12 xmlhttp.onload = function() {
13   let data = JSON.parse(this.responseText)
14   console.log(data[0].nom)
15 }
16
17 // Send a request
18 xmlhttp.open("GET", "https://geo.api.gouv.fr/regions")
19 xmlhttp.send()
20
21 </script>
22 </body>
23 </html>
```

L’Ajax : Parsing XML

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<person>
    <firstName>John</firstName>
    <lastName>Smith</lastName>
    <age>22</age>
    <address>
        <streetAddress>21 2nd Street</streetAddress>
        <city>New York</city>
        <state postalCode="10021" name="NY"/>
    </address>
    <phoneNumber type="home">
        212 555-1234
    </phoneNumber>
    <phoneNumber type="fax">
        646 555-4567
    </phoneNumber>
</person>
```

L’Ajax : Parsing XML

```
<!DOCTYPE html>
<html>
<meta charset="utf-8"/>
<body>

<span id="demo"></span>

<script>

var xmlhttp = new XMLHttpRequest()

xmlhttp.onload = function() {
  let data = this.responseXML
  console.log(data)
  document.getElementById("demo").innerHTML = data.getElementsByTagName("firstName")[0].innerHTML
}

xmlhttp.open("GET", "http://localhost/api/data.xml")
xmlhttp.send()

</script>
</body>
</html>
```