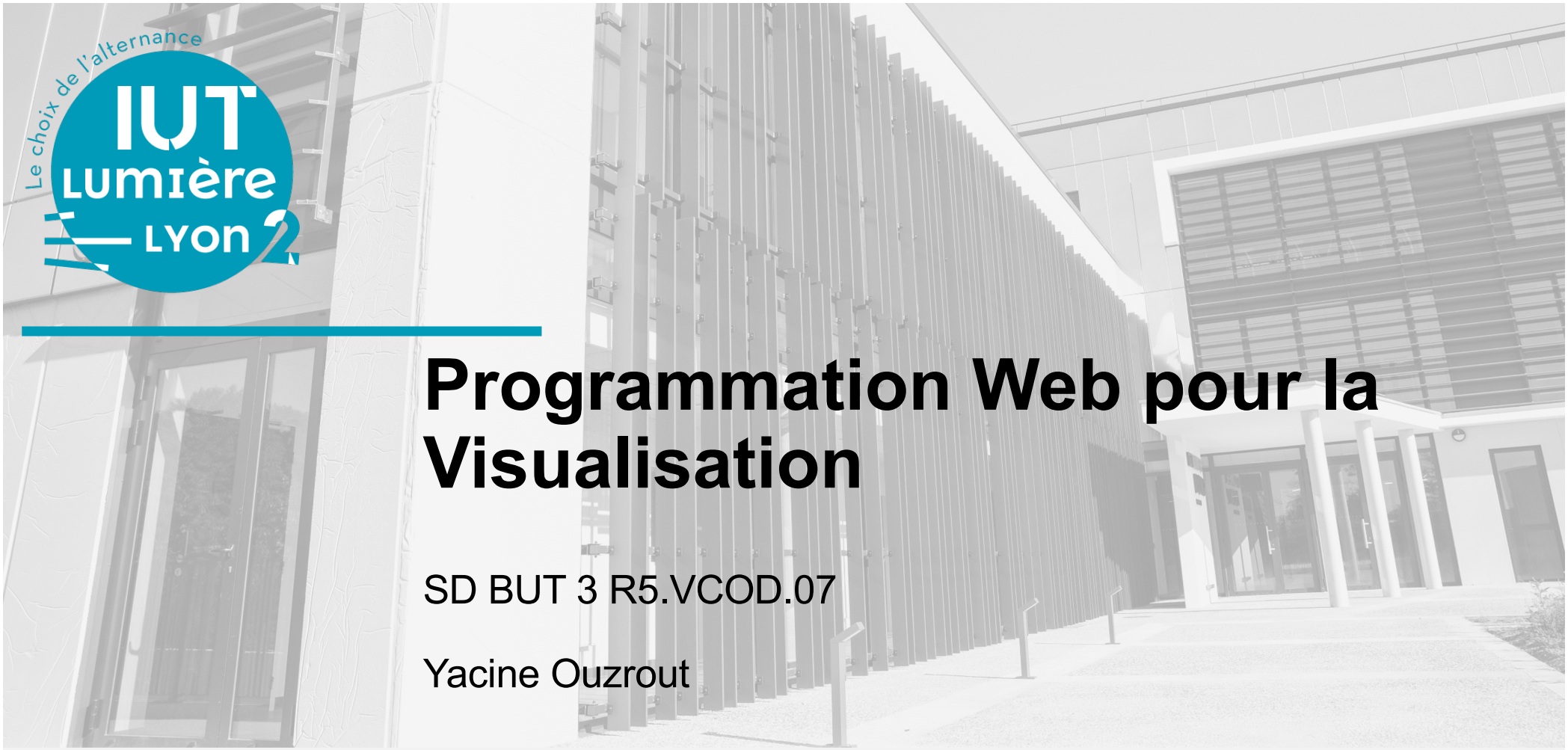


Programmation Web pour la Visualisation

SD BUT 3 R5.VCOD.07

Yacine Ouzrout

ORGANISATION DU COURS

7 TD et 10 TP

Le choix de l'alternance
IUT Lumière Lyon 2

ORGANISATION DU COURS

7 TD et 10 TP

MCCC
- 1 TD ordinateur (0,5)
- 1 DS ordinateur (0,5)

- **Partie 1 : Approfondissement Javascript**
  - Programmation Objet en JS
  - Structuration des pages Web (DOM)
- **Partie 2 : Traitement des données Json**
  - Les formats de transfert données XML, Json...
  - Automatisation du traitement des données avec Json
  - Exploitation des données en JS
- **Partie 3 : Visualisation avec D3.js** https://observablehq.com/@d3/gallery
  - Les librairies JS pour la visualisation
  - Développement de graphiques avec D3.js
- **Partie 4 : Le javascript côté serveur**
  - Découverte de la plateforme NodeJS
  - Interaction avec une base de données
  - Développement d'une application web avec le Framework Express.js
  - API REST

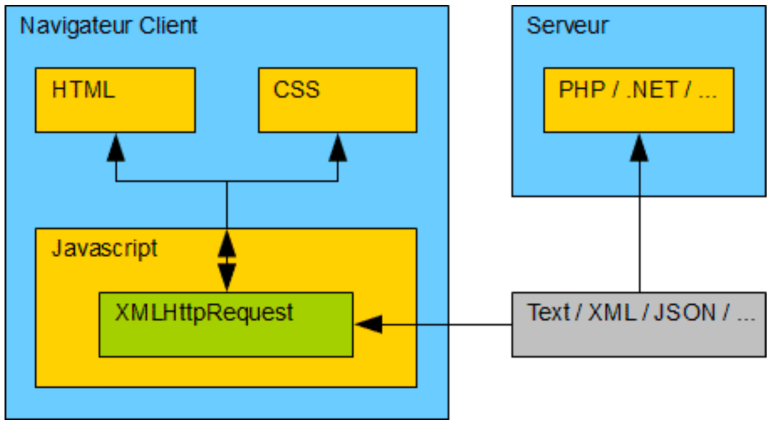
PARTIE 1 : JSON

Principe général et historique (1/2)

- Faire communiquer une page Web avec un serveur Web sans occasionner le rechargement de la page.
- **AJAX** signifie **A**synchronous **JA**vaScript and **X**ML. AJAX n'est pas un langage de prog. mais un ensemble de techniques pour envoyer et récupérer des données vers et depuis un serveur de façon asynchrone, c'est-à-dire sans avoir à recharger la page.
- Les objets **XMLHttpRequest** ou **Fetch** sont utilisés par AJAX pour la communication asynchrone.
- Echange de données avec le server : Text, HTML, XML/JSON Asynchrone : la fonction qui envoie une requête au serveur n'est pas la même que celle qui en recevra la réponse. Processus non bloquant.

A sa création, **AJAX** utilisait :

- XML pour l'échange de données avec le serveur
- L'objet *XMLHttpRequest* pour la communication asynchrone
- JavaScript pour afficher les données de manière dynamique
- HTML et CSS pour la présentation des données



Principe général et historique (2/2)

- Le XML a été largement délaissé au profit du **JSON** (**J**ava**S**cript **O**bject **N**otation)
- **JSON** est un **format standard** utilisé pour représenter des données structurées de façon semblable aux objets Javascript.
- **JSON** est utilisé pour structurer et transmettre des données sur des sites web (par exemple, envoyer des données depuis un serveur vers un client afin de les afficher sur une page web ou vice versa)
- Malgré sa syntaxe très similaire à celle des objets JavaScript, **JSON** peut être utilisé indépendamment de ce langage.

L'objet XMLHttpRequest

- Permet d'envoyer une requête HTTP au serveur, et de récupérer les données renvoyées par celui-ci.
- Les objets XMLHttpRequest permettent de récupérer des données à partir d'une URL sans avoir à rafraîchir la page. Cela permet à une page web d'être mise à jour sans perturber les actions de l'utilisateur.
- Developé par Microsoft en 1999 sous le nom de XMLHTTP (outlook, IE 5), adopté par les autres navigateurs sous le nom XMLHttpRequest. Standardisé plus tard par W3C https://www.w3.org/TR/XMLHttpRequest/
- L'API XMLHttpRequest (appelé XHR) est un objet JavaScript qui permet de construire une requête afin d'interroger un serveur pour obtenir des ressources diverses (images, texte, JSON, ou n'importe quel extrait HTML) le tout en Javascript.
- L'objet XMLHttpRequest appartient à l'interface **XMLHttpRequestEventTarget** qui implémente elle-même l'interface DOM **EventTarget**. La plupart des navigateurs récents gère également les évènements via la méthode **addEventListener()**

Un objet XMLHttpRequest (1/4)

- Le **constructeur** initialise un objet XMLHttpRequest. Il doit être appelé avant toute autre méthode :
  var vRequete = new XMLHttpRequest();

- **Les Principales Propriétés :**
  - XMLHttpRequest.**readyState** état de la requête sous la forme d'un *unsigned short*
  - XMLHttpRequest.**onreadystatechange** Un gestionnaire d'évènement invoqué quand l'attribut *readyState* change
  - XMLHttpRequest.**response** objet qui contient le corps de la réponse
  - XMLHttpRequest.**responseText** chaîne de caractères *DOMString* qui contient la réponse à la requête sous forme de texte ou la valeur null si la requête a échoué
  - XMLHttpRequest.**responseType** une valeur parmi une liste qui définit le type de réponse : *Blob*, *Document*, *ArrayBuffer, Json, Text*
  - XMLHttpRequest.**responseUrl** L'URL sérialisée de la réponse ou la chaîne vide si l'URL est nulle.
  - XMLHttpRequest.**responseXml** Un objet Document qui contient la réponse de la requête ou null si la requête a échoué ou qu'elle ne peut pas être analysée comme XML ou HTML.
  - XMLHttpRequest.**status** valeur numérique *unsigned short* qui décrit le statut de la réponse à la requête.
  - XMLHttpRequest.**statusText** chaîne *DOMString* qui contient la chaîne de caractères/réponse renvoyée par le serveur HTTP. À la différence de XMLHttpRequest.status, tout le texte du statut est inclus ("200 OK" plutôt que "200" par exemple)
  - XMLHttpRequest.**timeout** Une Un entier unsigned long qui représente le nombre de millisecondes qu'une requête peut prendre avant d'être terminée automatiquement

Un objet XMLHttpRequest (2/4)

- Le **constructeur** initialise un objet XMLHttpRequest. Il doit être appelé avant toute autre méthode : var vRequete = new XMLHttpRequest();

- **Principales méthodes :**
  - XMLHttpRequest.**open()** Initialise une requête. Cette méthode doit être utilisée par du code JavaScript.
  - XMLHttpRequest.**send()** envoie la requête. Si la requête est asynchrone, le comportement par défaut, la méthode renvoie un résultat dès que la requête est envoyée
  - XMLHttpRequest.**abort()** Interrompt la requête si elle a déjà été envoyée.
  - XMLHttpRequest. **getAllResponseHeaders()** renvoie, via une chaîne de caractères, l'ensemble des en-têtes de la réponse, séparés par (\r ou \n) ou la valeur null si aucune réponse n'a été reçue
  - XMLHttpRequest. **getResponseHeader()** renvoie la chaîne de caractères contenant le texte de l'en-tête voulue ou null si aucune des réponses n'a été reçue ou si l'en-tête n'existe pas dans la réponse
  - XMLHttpRequest.**overrideMimeType()** surcharge le type MIME renvoyé par le serveur,
  - XMLHttpRequest.**setRequestHeader()** définit la valeur d'un en-tête de requête HTTP. Cette méthode doit être appelée après *open()* mais avant *send()*

Un objet XMLHttpRequest (3/4)

- Le **constructeur** initialise un objet XMLHttpRequest. Il doit être appelé avant toute autre méthode :
  var vRequete = new XMLHttpRequest();
- **Les principaux événements :**
  - **abort** Se déclenche lorsqu'une requête a été interrompue (par exemple via *XMLHttpRequest.abort()*. Le gestionnaire *onabort* est également disponible.
  - **error** Se déclenche lorsqu'une requête a rencontré une erreur. Le gestionnaire *onerror* est également disponible.
  - **load** Se déclenche lorsqu'une transaction XMLHttpRequest se termine correctement. Le gestionnaire *onload* est également disponible.
  - **loadend** Se déclenche lorsqu'une requête est terminée. Quand elle a réussi, l'évènement a lieu après *load*. Quand elle a échoué, l'évènement survient après *error*. Le gestionnaire *onloadend* est également disponible. .
  - **loadstart** Se déclenche lorsqu'une requête commence à charger des données. Le gestionnaire *onloadstart* est également disponible.
  - **progress** Se déclenche périodiquement lorsqu'une requête reçoit des données supplémentaires. Le gestionnaire *onprogress* est également disponible.
  - **timeout** Se déclenche lorsque la progression est terminée du fait de l'expiration de la durée limite. Le gestionnaire *ontimeout* est également disponible.

Un objet XMLHttpRequest (4/4)

Exemples de gestion des événements :

- Utilisez le nom de l'événement (cf. diapo précédente) dans des méthodes telles que : **addEventListener()**

```
// Ajout d'un gestionnaire d'événements qui réagira si la requête
// HTTPXMLRequest est abandonnée

addEventListener("abort", (event) => { //fonction qui définit le traitement à effectuer  });
```

- Ou définissez la propriété d'un gestionnaire d'événement, en ajoutant « on » devant l'événement

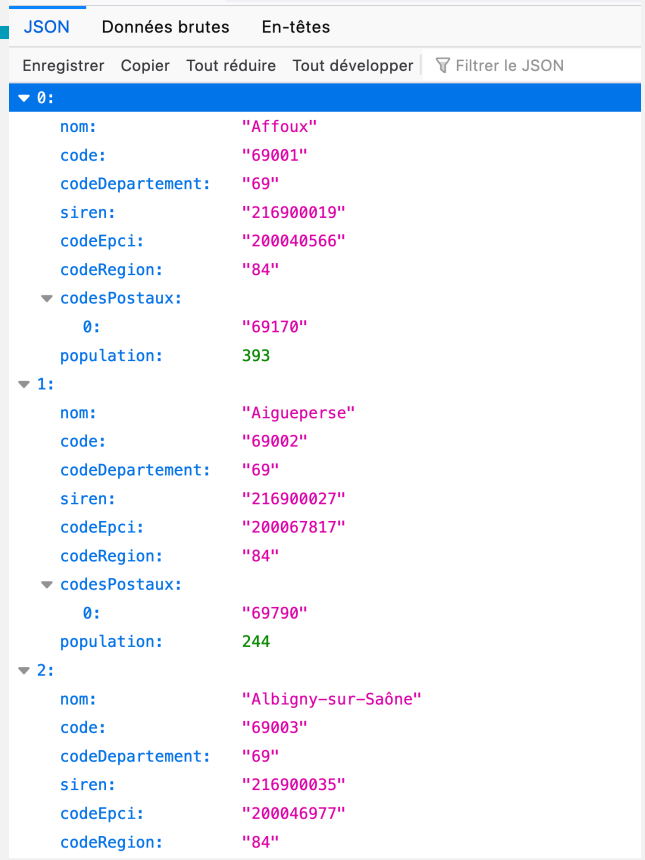
```
// Ajout d'un on devant l'événement afin de réagir si la requête
// HTTPXMLRequest est abandonnée

onabort = (event) => { //définir le traitement à effectuer  });
```

Création d'un objet XMLHttpRequest

Exemple de création d'un objet XMLHttpRequest.

```
function requete() {
    var adresse = 'https://geo.api.gouv.fr/departements/69/communes'
    var vRequete = new XMLHttpRequest();
    vRequete.onreadystatechange = gestion_requete;
    vRequete.open('GET', adresse);
    vRequete.send();
}
function gestion_requete() {
    var donnees =vRequete.responseText;
    donnees =JSON.parse(donnees);
    document.getElementById("rendu").innerHTML= donnees[0].nom; //Ecrit le nom de la 1ère commune du département 69, dans l'élément de la page HTML dont l'Id est égale à "rendu"
}
```

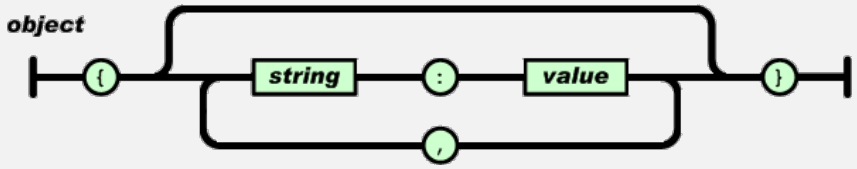


JSON

- JSON se présente sous la forme d'une chaîne de caractères.
- Il faut le convertir en un objet JavaScript natif lorsque l'on souhaite accéder aux données.
- JavaScript fournit un objet global JSON disposant des méthodes pour assurer la conversion entre les deux.
- Un objet JSON peut être stocké dans son propre fichier qui se présente sous la forme d'un fichier texte avec l'extension .json

JSON

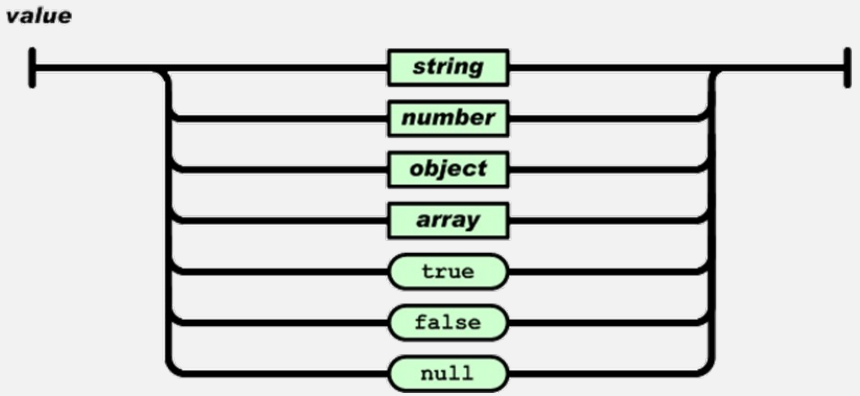
- JSON est construit par rapport à deux structures :
  - **Un objet** est un ensemble non-ordonné de paires **nom : valeur**. Un objet commence avec **{** et se termine avec **}**. Les noms sont suivis de **:** et les paires *nom : valeur* sont séparées par des **,**
  - **Un tableau** est une collection ordonnée de valeurs. Un tableau commence avec **[** et se termine avec **]**. Les valeurs sont séparées par des **,**



JSON

Un objet JSON accepte comme valeurs les mêmes types de données de base que tout autre objet Javascript :

- chaînes de caractères,
- nombres,
- tableaux,
- booléens
- date
- objets



Objet JSON

```
var person = {
   "name" : "Alex",
   "age" : "25",
   "addresse" : "Paris"
};
```

Pour accéder aux données d'un objet :

```
document.write("Le nom est: " + person.name);
document.write("L'age est: " + person.age);
document.write("L'adresse est: "+ person.addresse);
```

Objets JSON dans un tableau

```
var person = [
{
"name" : "Alex",
"age" : "25",
"address" : "Paris"
},
{
"name" : "Emily",
"age" : "22",
"address" : "Toulouse"
} ];
```

Pour accéder aux données d'un objet :

```
document.write("Le nom de la personne 1 est : " + person[0].name); //Alex
document.write("Le nom de la personne 2 est : " + person[1].name); //Emily
```

Objets JSON dans un tableau

```
var persons = {
  "p1" : {
      "name" : "Alex",
      "age" : "25",
      "address" : "Paris"
  },
  "p2" : {
      "name" : "Emily",
      "age" : "22",
      "address" : "Toulouse"
   }
}
```

Pour accéder aux données d'un objet :

```
document.write("Le nom de la personne 1 est : " + persons.p1.name);
document.write("Le nom de la personne 2 est : " + persons.p2.name);
```

```
{
  "squadName": "Super hero squad",
  "homeTown": "Metro City",
  "formed": 2016,
  "secretBase": "Super tower",
  "active": true,
  "members": [
    {
      "name": "Molecule Man",
      "age": 29,
      "secretIdentity": "Dan Jukes",
      "powers": ["Radiation resistance", "Turning tiny", "Radiation blast"]
    },
    {
      "name": "Madame Uppercut",
      "age": 39,
      "secretIdentity": "Jane Wilson",
      "powers": [
        "Million tonne punch",
        "Damage resistance",
        "Superhuman reflexes"
      ]
    },
    {
      "name": "Eternal Flame",
      "age": 1000000,
      "secretIdentity": "Unknown",
      "powers": [
        "Immortality",
        "Heat Immunity",
        "Inferno",
        "Teleportation",
        "Interdimensional travel"
      ]
    }
  ]
```

Exemple :

```
//Objet JavaScript
let utilisateur = {
  "prenom": "Jean", "nom":  "Durand",
  "adresse": { "rue": "30 av des corbeaux",
  "ville": "Lyon", "cp": 69002,
  "pays": "France" },
  "mails": [
        "jean.durand@gmail.com",
        "jdurand@gmail.com"
   ] };

//Conversion en chaine JSON
let vJson = JSON.stringify(utilisateur);
//Ecriture du résultat dans la page HTML
document.getElementById("resultat").innerHTML = "Type de la variable : "
+ typeof(vJson) + "<br> Contenu de la variable : " + vJson;
```

Données JSON : un Nom et une Valeur

- Les chaînes de caractères dans JSON doivent être écrites entre guillemets.

```
{ "nom":"John" }
```

- Les nombres dans JSON doivent être des nombres entiers ou des nombres à virgule flottante.

```
{ "age":30 }
```

- Les valeurs dans JSON peuvent être des objets.

```
{
"employe":{ "nom":"John", "age":30, "ville":”Lyon" }
}
```

- Les valeurs dans JSON peuvent être des tableaux.

```
{
"employe":[ "John", ”Eric", "Peter" ]
}
```

Exemple: XML vs JSON

XML

```
<?xml version="1.0"?>
<employees>
   <employee>
      <prenom >John</prenom> <Nom>Doe</Nom>
   </employee>
   <employee>
      <prenom >Anna</prenom> <Nom>Smith</Nom>
   </employee>
   <employee>
      < prenom >Peter</prenom> <Nom>Jones</Nom>
   </employee>
</employees>
```

JSON

```
{
"employees":[
   { ”prenom":"John", ” Nom ":"Doe" },
   { " prenom ":"Anna", " Nom ":"Smith" },
   { " prenom ":"Peter", " Nom ":"Jones" }
]}
```

XML vs JSON

JSON est comme XML parce que

- JSON et XML sont tous deux "autodescriptifs" (lisibles par l'homme).
- JSON et XML sont tous deux hiérarchiques (valeurs dans les valeurs)
- JSON et XML peuvent être analysés et utilisés par de nombreux langages de programmation.

JSON est différent de XML pour les raisons suivantes

- JSON n'utilise pas de balise de fin
- JSON est plus rapide à lire et à écrire
- JSON peut utiliser des tableaux
- JSON s'adapte mieux aux langages objets que XML

La principale différence est la suivante :

- XML doit être analysé avec un analyseur XML. JSON peut être analysé par une fonction JavaScript standard.

Javascript & JSON

Pour en savoir plus :

Cours Json :
https://developer.mozilla.org/fr/docs/Learn/JavaScript/Objects/JSON
https://www.hostinger.fr/tutoriels/quest-ce-que-json

Le langage Javascript et Json :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/JSON/parse

Cous en ligne : Javascript + Json
https://openclassrooms.com/fr/courses/7697016-creez-des-pages-web-dynamiques-avec-javascript
https://codes-sources.commentcamarche.net/faq/11706-exploiter-des-donnes-json

Cours Json + XML
https://stph.scenari-community.org/contribs/doc/cdt/json1/co/json-js.html