TP 4 – Ajax et Json

ON n'oublie pas le Linter : eslint => airbnb

1. JavaScript Object Notation (JSON)

Json est un format d'échange de données qui a pour objectifs principaux d'être léger, lisible et comme sa syntaxe est celle utilisée en javascript il s'interface très facilement.
Les types principaux du langage JSON sont :

- les nombres, qu'ils soient entiers, réels ou à virgule flottante ;
- les chaînes de caractères ;
- les booléens (true et false) ;
- les tableaux classiques [valeur1, valeur2, …] ;
- les tableaux associatifs qui correspondent aux objets javascript (que l'on n'a pas abordé durant ce cours) et qui permettent d'associer des valeurs à des clés {cle1 : valeur1, cle2, valeur2, …}

```
// Exemple de fichier JSON
const jsonobject = {
  "nom":"Guillaume",
  "adresse": {"cp":17000, "ville":"La Rochelle"},
  "notes": [1, 2, 4, 8, 16, 32]
};
```

Pour parcourir du JSON, on utilise les mêmes méthodes qu'en Javascript : le symbole « . » pour accéder à un attribut d'un objet, les « [] » pour parcourir un tableau.

```
// Récupérer le nom :
jsonobject.nom;
// Récupérer la ville dans le champ adresse :
jsonobject.adresse.ville;
// Récupérer la troisième note du tableau
jsonobject.notes[2];
```

Pour convertir du texte en JSON ou l'inverse on dispose des fonctions suivantes, sachant que l'usage de la fonction eval est fortement non recommandé ! On utilise cela pour recevoir ou envoyer des données JSON.

```
// convertir une chaîne de caractère en objet. Attention il ne peut
pas y avoir de retours à la ligne simples dans une chaine en JS.
let o = JSON.parse('{"nom":"Guillaume"}');
console.log(o.nom); // -> Guillaume

// convertir un objet JSON en chaîne de caractère
let p = JSON.stringify(o);
console.log(p); // -> {"nom":"Guillaume"}

// evaluer une chaîne comme du code de caractère
eval('var o = {"nom":"Guillaume"}');
console.log(o.nom); // -> Guillaume
```

On peut enfin faire de l'affectation par décomposition, comme par exemple

```
let jsonobject = {
  "nom":"Guillaume",
  "adresse": {"cp":17000, "ville":"La Rochelle"},
  "notes": [1, 2, 4, 8, 16, 32]
};
let {nom :x} = jsonobject ; // x contient Guillaume
let {adresse} = jsonobject ; // adresse contient {"cp":17000...}
```

Exercice 1. Ecrire les instructions pour récupérer à partir du fichier suivant : le titre de la fenêtre – la 3e coordonnée de l'image – le nombre de messages – l'offset y du dernier message.

```
const data = {
  "window": {"title": "Sample Widget", "width": 500, "height": 500},
  "image": {"src": "images/logo.png", "coords": [250, 150, 350, 400]},
  "messages": [
    {"text": "Save", "offset": [10, 30]},
    {"text": "Quit", "offset": [30, 10]}]};
```

Comment faites-vous pour que le code ci-dessus passe au linter sans modifier les règles de celui-ci

Exercice 2. On souhaite stocker le menu d'un site web dans un objet JSON pour pouvoir ensuite l'afficher en javascript. Un menu est un tableau d'éléments et chaque élément est soit un titre et un lien hypertexte, soit un titre et un sous-menu. Ecrivez la structure de données permettant de gérer un menu à plusieurs niveaux.

Exercice 3. Ecrire une fonction javascript qui prend un paramètre représentant un menu en json (format de l'exercice précédent) et crée le menu HTML sous forme de listes (ul et li) imbriquées. Vous pouvez utiliser hasOwnProperty("cle") qui permet de savoir si une clé existe :

2. Asynchronous JavaScript and XML (AJAX)

A faire uniquement la partie avec le FETCH, le début c'est de la culture

L'objectif d'AJAX est de pouvoir se connecter à n'importe quel moment à un serveur web en javascript pour envoyer ou récupérer des informations. Par exemple, sur tous les réseaux sociaux en ligne, quand vous arrivez en bas de votre fil d'actualités, les actualités plus anciennes sont chargées automatiquement. Ces informations sont récupérées via une connexion au serveur, puis sont traitées et incorporées dans la page en créant de nouveaux éléments.
On considère généralement qu'AJAX est l'ensemble des traitements permettant de rendre la page dynamique. On peut également considérer qu'AJAX ne représente que la partie liée aux communications, tout le reste étant du javascript classique.

AJAX peut fonctionner en mode synchrone (bloquant) ou asynchrone (non bloquant). Le mode asynchrone est beaucoup mieux pour l'utilisateur mais beaucoup plus complexe à développer. Le *mode synchrone en dehors d'un worker en maintenant déprécié en javascript*. Il n'est donc plus possible d'avoir une requête vraiment bloquante.

Mode synchrone (A ne pas faire c'est de la culture)

En mode synchrone on utilise les fonctions open pour ouvrir une connexion et send pour envoyer la requête.

```
// Création d'un objet permettant de créer des requêtes
let xhr = new XMLHttpRequest();

// Ouverture d'une connexion avec le serveur
// methode = GET ou POST
// asynch_flag = false pour synchrone, true pour asynchrone
    xhr.open(methode, url, asynch_flag);

// Envoi d'une requête - bloquant
    xhr.send(parametres);

// Récupération du résultat
let res = xhr.responseText
open : cinq arguments (protocole, url, synchrone (true) ou asynchrone
(false), username, password)
send : un argument si des informations sont envoyées
```

Le code pour récupérer et afficher le contenu de la page https://www.univ-larochelle.fr en mode synchrone.

```
let xhr=new XMLHttpRequest();
xhr.open("GET", "https://www.univ-larochelle.fr", false);
xhr.send(null);
console.log(xhr.responseText);
```

```
Mode asynchrone
```

En mode asynchrone, c'est « presque » pareil sauf que l'appel à send n'est pas bloquant. Il faut donc définir un callback qui traitera les résultats une fois arrivés.

```
xhr.onreadystatechange  = function() {traitement du résultat}
```

Cette fonction sera appelée à chaque événement sur la connexion (établissement de la connexion, envoi, réception des données, fin de réception). On peut accéder à différentes informations dans ce callback, notamment responseText mais également xhr.readyState qui peut prendre les valeurs suivantes :

```
0    UNSENT              fonction open() pas encore appelée
1    OPENED                   fonction open() appelée
2    HEADERS_RECEIVED     function send() appelée et entêtes et statut
disponible
3    LOADING             responseText contient une partie de la réponse
4    DONE                 tout est terminé, responseText ets complet
```

Et xhr.status qui indique le statut de la connexion qui vaut 200 (ou 0 en local) si tout se passe bien et peut prendre d'autres valeurs en cas d'erreur (type erreur 404, erreur 500, etc.).

> Pour effectuer une requête vers https://www.univ-larochelle.fr en asynchrone et afficher à chaque changement d'état : le readyState, le status, la longueur de responseText.

```
var xhr = new XMLHttpRequest();
xhr.onreadystatechange=function(){
  console.log(xhr.readyState, xhr.status, xhr.responseText.length);
};
xhr.open('GET','https://www.univ-larochelle.fr',true);
xhr.send();
```

Exercice 4. Ecrire une requête AJAX asynchrone et n'afficher le résultat que quand tout est terminé et qu'il n'y a eu aucune erreur de connexion. En cas d'erreur, afficher un message adéquat.

```
var xhr = new XMLHttpRequest();
    xhr.onreadystatechange  = function() {
            if(xhr.readyState  == 4) {
                    if(xhr.status  == 200)
                            console.log("Received : "  +
xhr.responseText);
                    else
                            console.log("Error code :  " + xhr.status);
  }
};
    xhr.open("GET", "https://www.univ-larochelle.fr", true);
    xhr.send(null);
```

Mode asynchrone avec fetch (A FAIRE)

Il est maintenant possible dans la plupart des navigateurs d'utiliser la fonction fetch – en cours de développement (voir https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API) –et des promesses – depuis ES6. Les promesses sont similaires aux callbacks mais peuvent être enchaînées beaucoup plus facilement. Dans l'exemple ci-dessous on récupère une url et en cas de succès on convertit le résultat en json. Cette fonction de conversion retourne à nouveau une promesse qui, en cas de succès, affiche le résultat. Si échec un message d'erreur est affiché. L'objet réponse possède des attributs similaires au status (https://developer.mozilla.org/en-US/docs/Web/API/Response).

```
fetch(url)
.then((response) => response.json())
  // promesse de récupération en cas de succès
     //  retourne  la  promesse  d'une  conversion  en  json,  text
principalement
.then((data) => {
                          // promesse de conversion en json – utiliser
une fonction pour traiter les données
    console.log(data);
})
.catch((err) => {     // en cas d'échec
    console.log(err.message);
});
```

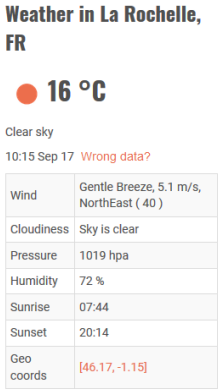
Exercice 5. Refaire l'exercice 6 en mode fetch + promesse (c'est l'exemple ci-dessus avec en plus les traitements d'erreurs).

Remarques :

- A chaque fois que vous utilisez une API externe, veillez à ne pas la surcharger de requêtes. La plupart des API (facebook, twitter, wikipedia, etc.) ont des limiteurs sur le nombre de requêtes et il est possible d'être banni en cas d'excès.
- Afin d'éviter les attaques via des injections de scripts, les navigateurs utilisent une règle liée à l'origine des requêtes AJAX qui limitent les requêtes vers le site qui a fourni le script (http://en.wikipedia.org/wiki/Same-origin_policy). De manière plus générale les règles sont fixées par le mécanisme de « Cross-Origin Ressource Sharing (https://developer.mozilla.org/fr/docs/Web/HTTP/CORS). Dans des cas simples tout se passe bien et il n'y a rien à faire. C'est notamment le cas lorsque l'on récupère du texte.
- Si vous êtes bloqué par CORS c'est normal. L'une des manières de dépasser cela est d'utiliser JSONP qui est un hack permettant de passer outre (http://en.wikipedia.org/wiki/JSONP) en utilisant le fait qu'on peut récupérer un script javascript sur un autre site. C'est très simple à utiliser en jQuery, un peu moins en javascript pur mais de nombreux exemples sont disponibles en ligne…

Exercice 6. Nous souhaitons utiliser l'API jsonplaceholder renvoyant des fausses données JSON. Nous souhaitons faire afficher sous forme de liste les données renvoyées par l'url https://jsonplaceholder.typicode.com/todos en utilisant le fetch. Vous créerez 2 listes une pour celles complétées (completed) et une autre pour les autres.

Exercice 7. Nous souhaitons utiliser l'api openweather permettant de faire afficher les données relatives au temps sur la rochelle en live. Vous utiliserez https://openweathermap.org/ et son api. Pour pouvoir utiliser celle-ci vous devez créer un compte pour obtenir la clé d'API. Utilisez le fetch et affichez les résultats sous la forme de



Pour rendre plus utile, cet exercice, vous pouvez ajouter une zone de saisie de la ville permettant d'afficher le temps de toutes les villes.

Exercice 8. Nous souhaitons faire afficher une carte montrant les lieux où se trouvent les différents défibrillateurs de la ville de Toulouse. Vous utiliserez l'API https://data.toulouse-metropole.fr/api ainsi que https://leafletjs.com/ pour afficher la carte

Exercice 9. Nous souhaitons faire une recherche (champs input) et afficher les résultats obtenus via l'API de wikipedia : récupération des résultats en JSON, parsing puis création de la liste. Vous pouvez (devez) suivre les étapes suivantes :

1. Regarder la page de l'API de wikipedia : http://www.mediawiki.org/wiki/API:Search.
2. Tester directement dans la sandbox sur une recherche. Par exemple ci-dessous pour trouver les articles contenant « maison » et récupérer le résultat au format json : https://en.wikipedia.org/wiki/Special:ApiSandbox#action=query&format=json&origin=*&list=search&srsearch=maison
3. Codez en utilisant le fetch pour faire afficher uniquement la liste des mots contenant le champ de recherche.
4. Dans l'url de (2), on utilise origin=* pour éviter les problèmes liés à la same origin (c'est géré côté serveur). Enlevez-le et retestez puis corrigez.