CORRECTION – EXERCICES JAVASCRIPT

Objectif :

L'objectif est de réaliser une page web dynamique permettant d'afficher des albums d'artistes en fonction du nombre de vue/d'écoute. Les données qui serviront à tester notre logique sont fictives et seront issues depuis un site qui s'appelle **Jsonplaceholder**. Cette technique qui consiste à récupérer temporairement des données venant d'un système qui simule un comportement se nomme par le terme « franglais » **mocker**. Avant de savoir extraire les données depuis une ressource distante, nous les manipulerons en local pour ce TP.
Nous utiliserons différents outils JavaScript pour afficher notre page en manipulant successivement des données en **Json**, sous forme de **String**, dans des **Array**. Nous utiliserons le **DOM** pour récupérer les **event** et proposer l'affichage correspondant.

Ressources indispensables :

- *https://developer.mozilla.org/en-US/docs/Web/JavaScript*
- *https://www.w3schools.com/js/default.asp*
- *https://quickref.me/javascript*

Etape 1 : Une jolie collection

Un dépôt du projet est à récupérer sur **Github**. Vous pouvez récupérer le squelette du code avec la commande **git clone https://github.com/laurentgiustignano/exercicesjavascript** dans votre répertoire de travail. Ou bien depuis **PhpStorm**, en choisissant : **Get from Version Control** avec la même URL. Si vous utilisez **Visual Studio Code**, l'option à choisir est **Cloner un dépôt Git…** depuis la page de Démarrage

Le projet comporte 4 fichiers :

- **index.html** qui correspond à la page d'accueil de notre site.
- **style.css** permet d'améliorer le rendu de la page **index.html**
- **datas.js** qui contient la constante mesDatas les données de **Jsonplaceholder** et de les proposer au format **Json** pour JavaScript.
- **app.js** qui contiendra le code JavaScript de notre

Remarque : les trois premiers fichiers n'ont pas besoin d'être modifié. L'ensemble de votre travail devra se trouver dans le fichier **app.js**.

En chargeant la page **index.html** dans votre navigateur, 3 boutons permettant de manipuler les données sont visibles. Mais aucune action n'est disponible pour l'instant. Ouvrez le fichier app.js pour observer le code. Un appel de fonction du DOM **console.log()** permet d'afficher une message dans la console de l'interpréteur JavaScript. Ici, il s'agit de votre navigateur.

- Retournez dans votre navigateur sur la page index.html, et faites un clic-droit dans la partie visible de la fenêtre du navigateur et choisissez « inspecter ». Une fois la fenêtre ouverte, allez jusqu'à l'onglet console. Un objet complexe est visible. Vous pouvez le déplier pour observer son contenu. Déterminez son type ?

Correction :

- L'objet dans la console est un tableau contenant les données au format **Json**. En détaillant son contenu, on observe l'ensemble des données reçues par **jsonplaceholder** dans des champs « id, user, … »

Etape 2 – C'est vu !

La constante **albums** contient des données représentant des albums avec un **userId** qui pourrait représenter l'auteur, un **title** pour le titre de l'album, ainsi qu'un **id** pour différentier tous les éléments de manière unique. Il nous faudrait rajouter un champ à chaque objet qui contiendrait le nombre de vue (ou d'écoute) pour chacun des albums. Pour cette fois-ci, nous allons tricher un peu en générant de manière aléatoire un nombre de vues compris

entre 100 et 10000, mais qui augmente de 100 à chaque palier (100, 200, 300, …, 9800, 9900, 10000). Ce nombre sera ajouté dans le nouveau champs vue de chaque album.

- Trouver l'expression permettant de générer un nombre entier dans l'intervalle demandé. Les fonctions **Math.floor()** et **Math.random()** sont nécessaire.
- Complétez la fonction **ajouterVues()** qui s'occupera de rajouter le champs vue à chaque élément de albums. Effectuez son appel et enregistrez le retour dans la variable **albumsAvecVues**.

Conseil : pour vous aider, la fonction **map()** *des objets* **Array** *permet de parcourir l'ensemble d'un tableau et pour chacun de ses éléments d'y exécuter un traitement. Cette fonction ne va pas modifier le tableau d'origine, mais retournera un nouveau tableau, de même dimension, avec les modifications. L'autre outil à utiliser est le* **spread operator** *(…). Cet opérateur permet de désigner tous les éléments d'un itérable, tout en les traitants de manière individuelle.*

```
const lettres = ['a', 'e', 'i']
const voyelles = [...lettres, 'o', 'u', 'y']
console.log(voyelles)   // affiche ['a', 'e', 'i', 'o', 'u', 'y']
```

Figure 1- Exemple d'utilisation du Spread Operator

Correction

```
function ajouterVues(desAlbums) {
    return desAlbums.map((albums) => {
        return {...albums, vue: Math.floor( x: Math.random() * 100 + 1) * 100}
    })
}
```

- La fonction **map()** effectue le parcours du tableau **desAlbums** et pour chaque élément, déclenche une fonction de **callback**. Ici, la fonction est une fonction fléchée, mais nous pourrions déclarer une autre fonction, et y faire appel en transmettant le paramètre **albums**.
- L'ensemble de la fonction de **callback** doit renvoyer une valeur, qui correspondra à chaque élément du nouveau tableau retourné par **map()**.
- Ici, nous utilisons le **spread operator** pour réutiliser la donnée d'un album existant, et en rajoutant le champ **vue**.
- Pour le calcul de la valeur **vue**, **Math.random() * 100 +1** donne une valeur réelle entre **1** et **100.999999**. La fonction **Math.floor()** permet de supprimer les décimales et en multipliant son retour par **100**, on obtient une valeur entre **100** et **10000** qui évolue de centaine en centaine.
- L'appel de cette fonction se trouve dans la structure **try{}**, par exemple après le **loader.remove()**.

Étape 3 – « attrapez les tous ! »

Maintenant, nous allons afficher tous les albums dans une structure **ul/li**. Cet affichage dois se déclencher lorsque l'utilisateur appuis sur le bouton contenant la chaine « **tous les titres** ». Il faut relier l'évènement du clic sur ce bouton vers la fonction **tous()**. Ensuite, il faut créer l'élément **ul**, et autant d'élément **li** que d'albums contenu dans **albumsAvecVues**. Ne pas oublier que pour chaque nouvelle référence de **HTMLElement**, il faut l'associer au DOM avec la fonction **prepend()** ou **append()**, si nous voulons les insérer, respectivement, au début ou à la fin de l'élément associé.

- A l'aide de la fonction **addEventListener()**, effectuez la liaison entre le bouton et la fonction **tous()**.
- A l'aide de la fonction **document.createElement()**, ajouter un élément **ul** ainsi que plusieurs **li** pour vérifier le comportement.
- A l'aide de la fonction **Object.entries()**, parcourrez **albumsAvecVues** pour récupérer notamment les valeurs et insérez dans chaque **li** le titre et le nombre de vues.

Correction :

```
export function injectElements(desAlbums, laListe) {
    for (let value of Object.values(desAlbums)) {
        let liListe = createElement( tagName: 'li')
        liListe.innerText = `${value.title} - ${value.vue}`
        laListe.append(liListe)
    }
}
2 usages  new *
export function renewTag(tagName) {
    const laListe = document.querySelector(tagName)
    if (laListe !== null) {
        laListe.remove()
    }
    return createElement(tagName)
}
```

```
function tous() {
    const laListe = renewTag( tagName: 'ul');
    wrapper.append(laListe)

    injectElements(albumsAvecVues, laListe);
}
```

- La fonction **renewTag()** permet de supprimer un tag de la page, puis de le recréer et renvoie la référence du **HTMLElement**. Son utilité apparaitra mieux lorsque les autres boutons seront utilisables. Le mot clé **export** devant sa déclaration signifie qu'elle est défini dans un autre fichier, et qu'elle doit être importée pour être utilisée, comme pour la fonction **fetchJson()** du fichier **api.js**.
- La constante **laListe** représentant **ul** est ainsi ajouté avec la fonction **append()**.
- La fonction **injectElements()** s'occupe de parcourir **albumsAvecVues**, et de placer dans **laListe** les albums extraits. La fonction **Object.values(desAlbums)** permet de récupérer uniquement les valeurs. Ici, la structure de contrôle **for(of)** permet de traiter chacune de ses valeurs sous le nom **value**, et ajoute à **laListe** un **HTMLElement** de **li**, dont le contenu, affecté avec **innerText**, est un ensemble formé par **value.title** et **value.vue**.

Étape 4 – « Simply the best ! »

A présent, nous travaillerons sur le bouton **BestOf** qui nous affichera uniquement les albums qui ont plus de 9000 vues. Le traitement est semblable à la fonction **tous()**, à la différence qu'il ne doit contenir qu'une sélection de **albumsAvecVues**.

- Reliez le bouton « Le Best Of » à la fonction **bestOf()**, puis à l'aide de la fonction **filter()**, ne conserver que les éléments respectant la condition du nombre de vues désirée.

Correction :

```
function bestOf() {
    const laListe = renewTag( tagName: 'ul');
    wrapper.append(laListe)

    const albumsFiltres = albumsAvecVues
        .filter((album) => {
            const NombreDeVues = 9000;
            return album.vue >= NombreDeVues;
        })

    injectElements(albumsFiltres, laListe);
}
```

- La fonction **renewTag()** et **injectElements()** sont réutilisés pour gérer l'affichage dans le **ul**. Ajouter directement les éléments filtrés dans l'élément **ul** existant afficherait à la suite les nouvelles données. Pour simplifier, supprimer et réinjecter la balise **ul** semble être une solution adéquate.
- Pour conserver uniquement les albums qui ont plus de 9000 vues, la fonction **filter()** permet de retourner un nouveau tableau avec les éléments respectant le critère. Cette décision est définie dans un **callback** qui renvoie **true** lorsque l'on doit conserver la donnée. Ici, une expression booléenne **album.vue >= nombreDeVues**.
- Le tableau **albumsFiltres** est ensuite utilisé comme paramètre de la fonction **injectElements()**.

Étape 5 – « Il n'en restera … que 3 »

Pour terminer, le bouton **top 3** doit présenter uniquement les 3 albums qui ont le plus de vues. Il est évident de penser que nous devons effectuer en premier lieu un tri sur l'ensemble des albums, puis en conserver que les 3 premiers. Deux fonctions vont vous aider : **sort()** qui permet de trier un ensemble, et **slice()** qui permet de couper un ensemble.

Conseil : lisez attentivement la documentation pour la fonction **sort()**. *Son comportement diffère des autres fonctions que vous avez observés sur les tableaux. On dit qu'elle a un traitement «* **in place** *».*

- Reliez le bouton « Top 3 » à la fonction **top3()**, puis compléter cette fonction pour qu'elle affiche uniquement les « 3 meilleurs albums ».

Correction :

```
function top3() {
    const laListe = renewTag( tagName: 'ul');
    wrapper.append(laListe)

    const debutTop3 = 0
    const finTop3 = 3

    const albumsTries = Array.from(albumsAvecVues)
        .sort( compareFn: (album1, album2) => album2.vue - album1.vue)
        .slice(debutTop3, finTop3)

    injectElements(albumsTries, laListe);
}
```

- La fonction **renewTag()** et **injectElements()** sont réutilisés pour gérer l'affichage dans le **ul**. Ajouter directement les éléments filtrés dans l'élément **ul** existant afficherait à la suite les nouvelles données. Pour simplifier, supprimer et réinjecter la balise **ul** semble être une solution adéquate.
- La fonction de tri **sort()** effectue son traitement **in place**. Il faut dupliquer le tableau pour ne pas perdre le tableau source **albumsAvecVues**. La fonction **Array.from()** permet de cloner le tableau fourni en

paramètre. Le tri s'effectue par un **callback** qui va comparer les éléments deux par deux, et si le retour est **< 0**, c'est que le premier élément est plus petit que le second, s'il est **> 0** alors le premier est plus grand et bien sûr ils sont identiques si le retour vaut **0**. Si nous souhaitons conserver les 3 plus grands, nous devons organiser le tri par ordre décroissant, donc le critère **album2.vue – album1.vue** permet le tri déscendant en fonction de la valeur **vue**.

- Ne garder que 3 éléments se fait avec la fonction **slice()**. Il faut l'utiliser avec deux paramètres, le début et la fin (non-incluse) des éléments à conserver. Ici, **slice(debutTop3,finTop3)** correspond à notre besoin, où **debutTop3** vaut **0** et **finTop3** vaut **3**.

Bon courage…