Développement d'applications web

Semaine 2 : NodeJs, premiers pas

Atelier 1 : Programmation asynchrone

Exercice 1 :

Soit le code javascript suivant :

```
function sendMessageWithConsoleLog(message){
    return console.log(message);
}

function sendMessageWithAlert(message){
    return alert(message);
}

function promptWithMessage(message){
    return prompt(message);
}

function confirmWithMessage(message){
    return confirm(message);
}

function sendMessageWithFromCallback(message){
    return console.log(message + " from a callback function!");
}
```

Veuillez proposer une fonction utilisant le callback pour remplacer les cinq fonctions précédentes et favoriser la réutilisation du code.

Exercice 2 :

Veuillez completer le code de la function each du script suivant afin d'obtenir les résultats des appels décrits en dessous.
N.B : La fonction acceptera deux paramètres en entrée.

```
// this function should accept 2 parameters, put them in!
function each(){
    // put your code inside here!
}


each([1,2,3,4], function(val){
    console.log(val);
});
// Here is what should be output if you wrote the function correctly

// 1
// 2
// 3
// 4

each([1,2,3,4], function(val){
    console.log(val*2);
});

// Here is what should be output if you wrote the function correctly

// 2
// 4
// 6
// 8
```

Exercice 3 :

Veuillez écrire une fonction appelée **map** qui accepte deux paramètres : un tableau et un callback.
La fonction **map** devrait renvoyer un nouveau tableau avec le résultat de chaque valeur passée à la fonction de callback.
Voici un exemple d'appel :

```
map([1,2,3,4], function(val){
    return val * 2;
}); // [2,4,6,8]
```

Exercice 4 :

Veuillez écrire une fonction appelée **reject** qui accepte deux paramètres un tableau et un callback.
La fonction doit retourner un nouveau tableau avec toutes les valeurs qui ne retournent pas true au callback.
Voici deux exemples d'appels :

```
reject([1,2,3,4], function(val){
    return val > 2;
}); // [1,2]

reject([2,3,4,5], function(val){
    return val % 2 === 0;
}); // [3,5]
```

Exercice 5 :

Veuillez écrire une fonction appelée **countdown** qui accepte un nombre comme paramètre et toutes les 1000 millisecondes décrémente la valeur et l'affiche avec console.log.
Une fois que la valeur est 0, il devrait afficher "DONE!" et s'arrêter.

Exercice 6 :

Veuillez écrire une fonction appelée **randomGame** qui sélectionne un nombre aléatoire compris entre 0 et 1 toutes les 1000 millisecondes et chaque fois qu'un nombre aléatoire est sélectionné, ajoutez 1 à un compteur.
Si le nombre est supérieur à 0,75, arrêtez le minuteur et renvoyez le nombre d'essais effectués avant de trouver un nombre supérieur à 0,75.

Exercice 7 :

Veuillez écrire une fonction appelée **inOrder** qui accepte deux callbacks et les appelle dans l'ordre.
a. Implémentez **inOrder** en utilisant le modèle de callbacks.

```
var logOne = setTimeout(function() {
  console.log("one!");
}, Math.random() * 1000);

var logTwo = setTimeout(function() {
  console.log("two!");
}, Math.random() * 1000);

inOrder(logOne, logTwo);

// one
// two

// it should always log those two in order regardless of their timing
```

b. Refactor **inOrder** pour utiliser les promesses.
c. Refactor **inOrder** pour utiliser les async/await.