Module : JavaScript

Exercice n°10 - Callbacks

# Durée de l’exercice : 30minCallbacks

Les fonctions de rappel. En JavaScript, les fonctions sont des citoyens de première classe. Par conséquent, vous pouvez passer d'une fonction à une autre fonction en tant qu'argument.

Par définition, un callback est une fonction que vous passez à une autre fonction comme argument pour l'exécuter plus tard. Il s'agit d'une caractéristique très importante de la programmation asynchrone, qui permet à la fonction qui reçoit le callback d'appeler notre code lorsqu'il est terminé.

Cette caractéristique très importante de la programmation asynchrone permet à la fonction qui reçoit le callback d'appeler notre code lorsqu'elle termine une longue tâche, tout en nous permettant de poursuivre l'exécution du code.

**Objectif de l'exercice** : Connaître et comprendre les callbacks, comprendre le fonctionnement avec JavaScript.

**Énoncé** : Suivez les étapes ci-dessous pour réaliser votre exercice de callback.

Créez un callback de dossier et chaque nom de fichier pour réaliser les exercices suivants.

# callback-1.js

console.log('Je vais d'abord imprimer') ;

let callback = function () {

console.log('Je vais attendre 5 secondes et terminer la tâche') ;

} ;

setTimeout(callback, 5000) ;

# callback-2.js

Écrivez une fonction newArray qui prend 2 paramètres : arr et callback.

Ici, arr est un tableau de nombres et callback est une fonction qui prend les éléments de arr comme paramètre et renvoie un nouveau tableau basé sur la condition.

function newArray(arr, callback) {

//Todo : votre code va ici

}

let arr = [5, 6, 7, 4, 5, 6, 7, 1, 2] ;

console.log(newArray(arr, (x) => x + 1)) ; //incrémente l'élément du tableau par 1.

console.log(newArray(arr, (x) => x \* 2)) ; //double chaque élément du tableau.

console.log(newArray(arr, (x) => x \* x)) ; //équerre chaque élément du tableau.

Fonction d'ordre supérieur en JavaScript

#callback-3.js

Pratique, les méthodes de tableau suivantes,

Filtre, map, find, forEach, some, every, reduce, includes

const lists = [

{nom : 'Ordinateur', prix : 200 },

{nom : 'TV', prix : 300 },

{nom : 'Livre', prix : 100 },

{nom : 'Téléphone', prix : 600 },

{nom : 'Souris', prix : 50 },

{nom : "Clavier", prix : 75 },

{nom : 'Voiture', prix : 2000 },

]

// Écrivez une fonction pour chaque méthode de tableau et présentez la solution.

// Exemple :

// # filter.js

const filterItems = lists.filter((item) => {

return item.price > 100 ;

}) ;

console.log(lists) ;

console.log(filterItems) ;

Quelques autres problèmes à résoudre

Créez un fichier for-each-sum.js et résolvez la fonction ci-dessous :

// Ecrivez une fonction `forEachSum` qui accepte en argument un tableau de nombres comme argument et retourne la somme totale de tous les nombres.

// Votre fonction doit utiliser la méthode native Array#forEach.

console.log(forEachSum([10, 3, 5])) ; // 18

console.log(forEachSum([-6, 7, 9, 4])) ; // 14

console.log(forEachSum([])) ; // 0

Créez un fichier common-elements.js et résolvez la fonction ci-dessous :

// Écrivez une fonction `commonElements` qui accepte deux tableaux en tant que arguments.

// La fonction doit retourner un nouveau tableau contenant les éléments qui se trouvent dans les deux tableaux d'entrée.

// Votre fonction doit utiliser la méthode native Array#filter.

console.log(commonElements(['a', 'b', 'c'], ['c', 'a'])) ; // ['a', 'c']

console.log(commonElements(['chat', 'chien', 'souris', 'poisson'], ['chien', 'rat'])) ; // ['chien'].

console.log(commonElements(['skip', 'jump'], ['swim', 'hop'])) ; // []