

I. Gestion des tableaux en Javascript moderne

Enoncé	Correction
1. Étant donné un tableau [1, 2, 3, 4], utilise map pour retourner [2, 4, 6, 8]
2. À partir du tableau [1, 2, 3, 4, 5, 6], retourne uniquement les nombres pairs.
3. Calcule la somme des éléments du tableau [10, 20, 30, 40].
4. Affiche chaque fruit du tableau ["pomme", "banane", "orange"].
5. Transforme chaque mot en majuscules dans le tableau ["react", "javascript", "html"].
6. À partir du tableau d'objets suivant : <pre>const users = [{name: "Alice", age: 25}, {name: "Bob", age: 30}, {name: "Charlie", age: 35}];</pre> Retourne seulement les prénoms : ["Alice", "Bob", "Charlie"].
7. Trouve le premier nombre supérieur à 50 dans [10, 20, 60, 80].
8. À partir du tableau [5, 12, 8, 130, 44], garde seulement les nombres > 10 et multiplie-les par 2.
9. Compte combien de fois chaque fruit apparaît dans le tableau : <pre>const fruits = ["pomme", "banane", "pomme", "orange", "banane", "pomme"];</pre>
10. À partir du tableau [1, 2, 3, 4, 5, 6], a. garde uniquement les pairs b. multiplie-les par 3 c. calcule la somme finale.

II. gérer un panier d'une application shopping cart

Objectif : Construire une application Shopping Cart est un excellent exercice pour lier tes acquis en JavaScript (map, filter, reduce, forEach) et tes futures compétences en React.

La liste des produits :

```
const products = [
  { id: 1, name: "Laptop", price: 1200, image: "💻" },
  { id: 2, name: "Headphones", price: 200, image: "🎧" },
  { id: 3, name: "Phone", price: 800, image: "📱" },
  { id: 4, name: "Watch", price: 150, image: "⌚" },
];
```

Fonctions essentielles à coder :

1. Afficher la liste des produits (map)
2. Ajouter un produit au panier

3. Supprimer un produit du panier
4. Changer la quantité d'un produit
5. Calculer le total (reduce)

III. Énoncé détaillé – Projet Shopping Cart en JavaScript Natif

Objectif général : Créer une application web (frontend uniquement) qui gère :

- Une liste de produits.
- Un panier permettant d'ajouter, supprimer, incrémenter, décrémenter la quantité.
- Le calcul du total.

Tu dois utiliser JavaScript moderne avec les méthodes suivantes : map, forEach, find, filter, reduce. Le design doit être réalisé avec HTML + Bootstrap.

Partie 1 : Initialisation

1. Crée un fichier index.html contenant la structure de base (Bootstrap inclus). Ajoute deux sections principales :
 - Une section pour afficher la liste des produits.
 - Une section pour afficher le panier et le total.

The screenshot shows a web application titled "Shopping Cart en JavaScript Natif". At the top, there's a header with a shopping cart icon and the title. Below it is a grid of four product cards:

- Laptop: Image of a laptop, price 1200 €, "Ajouter au panier" button.
- Phone: Image of a smartphone, price 800 €, "Ajouter au panier" button.
- Headphones: Image of headphones, price 200 €, "Ajouter au panier" button.
- Watch: Image of a watch, price 150 €, "Ajouter au panier" button.

Below the grid is a section titled "Mon Panier" (My Cart) showing the items added:

Produit	Prix	Quantité	
Phone (800 €)	800 €	x2	[+] [−] [X]
Headphones (200 €)	200 €	x1	[+] [−] [X]
Watch (150 €)	150 €	x1	[+] [−] [X]

Total : 1950.00 €

2. Dans store.js, définis un tableau products contenant au moins 4 objets avec les propriétés suivantes : id, name, price, image.
(exemple : { id: 1, name: "Laptop", price: 1200, image: "💻" })
3. Crée aussi un tableau vide cart = [] pour stocker les articles du panier.

Partie 2 – Affichage des produits

4. En JavaScript (app.js), écris une fonction displayProducts() qui :
 - parcourt les products avec forEach,
 - crée pour chaque produit une carte Bootstrap (nom, prix, image),
 - ajoute un bouton "Ajouter au panier".

Question : pourquoi utiliser forEach ici plutôt que map ?

Partie 3 – Ajouter un produit au panier

5. Écris une fonction `addToCart(id)` qui :

- utilise `find` pour récupérer le produit correspondant à l'`id`.
- vérifie si le produit est déjà dans le panier :
 - s'il existe déjà → incrémente sa `quantity`.
 - sinon → l'ajoute avec `quantity = 1`.

Question : pourquoi `find` est mieux adapté que `filter` dans ce cas ?

Partie 4 – Affichage du panier

6. Crée une fonction `renderCart()` qui :

- parcourt `cart` avec `forEach`,
- affiche chaque article avec :
 - son nom, prix, quantité,
 - 3 boutons : incrémenter, décrémenter, supprimer.

Partie 5 – Modifier les quantités

7. Écris une fonction `incrementQuantity(id)` qui utilise `map` pour augmenter la quantité du produit correspondant.

8. Écris une fonction `decrementQuantity(id)` qui utilise `map` et `filter` pour :

- diminuer la quantité du produit,
- supprimer l'article du panier si la quantité devient 0.

Question : pourquoi utiliser ici `filter` en plus de `map` ?

Partie 6 – Supprimer un produit

9. Écris une fonction `removeFromCart(id)` qui utilise `filter` pour supprimer un article du panier.

Partie 7 – Calcul du total

10. Crée une fonction `updateTotal()` qui utilise `reduce` pour calculer le prix total du panier :

$$\text{total} = \sum (\text{price} \times \text{quantity})$$

Et affiche ce total dans l'HTML.

Question : pourquoi `reduce` est la méthode idéale pour ce calcul ?