Les tableaux en JavaScript

> Les tableaux font partie des structures de données les plus importantes en JavaScript.
> Ils permettent de stocker et de manipuler plusieurs valeurs en une seule structure.
> Les maîtriser est essentiel pour travailler efficacement avec des collections de données.

1. Qu'est-ce qu'un tableau ?

> Un tableau (`array`) est une structure de données qui peut contenir une liste d'éléments.
>
> Chaque élément est indexé, ce qui permet d'y accéder facilement.
> Les tableaux peuvent contenir différents types de données, y compris des nombres, des chaînes de caractères, des objets, ou même d'autres tableaux.

Exemple :

```javascript
const fruits = ["pomme", "banane", "cerise"];
console.log(fruits); // ["pomme", "banane", "cerise"]
```

2. Création de tableaux

2.1. Avec des crochets (`[]`)

> C'est la méthode la plus courante pour créer un tableau.

Exemple :

```javascript
const nombres = [1, 2, 3, 4, 5];
console.log(nombres); // [1, 2, 3, 4, 5]
```

2.2. Avec le constructeur `Array`

> Moins utilisée, cette méthode utilise le constructeur `Array`.

Exemple :

```javascript
const nombres = new Array(1, 2, 3);
console.log(nombres); // [1, 2, 3]
```

2.3. Créer un tableau vide

> Vous pouvez aussi commencer par un tableau sans élément.

Exemple :

```javascript
const tableauVide = [];
console.log(tableauVide); // []
```

3. Accéder et modifier les éléments d'un tableau

> Les éléments d'un tableau sont accessibles via leur indice, commençant toujours à `0`.

Accéder à un élément :

```javascript
const animaux = ["chat", "chien", "poisson"];
console.log(animaux[1]); // "chien"
```

Modifier un élément :

```javascript
animaux[1] = "lapin";
console.log(animaux); // ["chat", "lapin", "poisson"]
```

3.1. Taille d'un tableau

> La taille d'un tableau est dynamique.
> Vous pouvez ajouter ou retirer des éléments à tout moment, et la taille du tableau s'ajustera automatiquement.
>
> La propriété `length` vous permet de connaître le nombre d'éléments dans un tableau.

Exemple :

```javascript
const fruits = ["pomme", "banane", "cerise"];
console.log(fruits.length); // 3
fruits.push("kiwi");
console.log(fruits.length); // 4
```

4. Méthodes courantes des tableaux

> Les méthodes des tableaux sont des fonctions intégrées qui permettent de manipuler facilement les tableaux.
> Il faut savoir que certaines méthodes modifient le tableau d'origine (mutables), tandis que d'autres retournent un nouveau tableau sans le modifier (immutables).
>
> Certaines méthdoes retoune parfois une valeur en plus d'acomplir leur fonctionnalité (comme `pop` qui retourne l'élément supprimé en plus de le retirer du tableau).

4.1. Ajouter ou retirer des éléments

- `push` : Ajoute un ou plusieurs éléments à la fin du tableau + retourne la nouvelle longueur du tableau.
  - Syntaxe : `push(...elementsAAjouter)`

```javascript
const fruits = ["pomme"];
let longueur = fruits.push("banane");

console.log(fruits); // ["pomme", "banane"]
console.log(longueur); // 2
```

- `pop` : Supprime le dernier élément du tableau + retourne cet élément.
  - Syntaxe : `pop()`

```javascript
const fruits = ["pomme", "banane", "cerise"];
const dernierFruit = fruits.pop();

console.log(fruits); // ["pomme", "banane"]
```

- `unshift` : Ajoute un ou plusieurs éléments au début du tableau + retourne la nouvelle longueur du tableau.
  - Syntaxe : `unshift(...elementsAAjouter)`

```javascript
const fruits = ["pomme", "banane"];
let longueur = fruits.unshift("orange");

console.log(fruits); // ["orange", "pomme", "banane"]
```

- `shift` : Supprime le premier élément du tableau + retourne cet élément.
  - Syntaxe : `shift()`

```javascript
const fruits = ["orange", "pomme", "banane"];
let premierFruit = fruits.shift();

console.log(fruits); // ["pomme", "banane"]
```

4.2. Manipulation avancée

- `splice` : Ajoute, retire ou remplace des éléments à n'importe quelle position + retourne les éléments supprimés.
  - Syntaxe : `splice(indice, nombreASupprimer, ...elementsAAjouter)`

Exemple : `splice` pour ajouter un élément

```javascript
const fruits = ["pomme", "banane", "cerise"];
let supprimes = fruits.splice(1, 0, "kiwi");

console.log(fruits); // ["pomme", "kiwi", "banane", "cerise"]
```

Exemple : `splice` pour retirer un élément

```javascript
const fruits = ["pomme", "banane", "cerise"];
let supprimes = fruits.splice(1, 1, "kiwi", "orange");

console.log(fruits); // ["pomme", "kiwi"]
```

Exemple : `splice` pour remplacer un élément

```javascript
const fruits = ["pomme", "banane", "cerise"];
let supprimes = fruits.splice(1, 1, "kiwi");

console.log(fruits); // ["pomme", "kiwi", "cerise"]
console.log(supprimes); // ["banane"]
```

- `slice` : Retourne une copie d'une partie du tableau + ne modifie pas le tableau d'origine.
  - Syntaxe : `slice(debut, fin)`
  - méthode IMMUTABLE

```javascript
const fruits = ["pomme", "kiwi", "banane", "cerise"];
const subset = fruits.slice(0, 2);

console.log(subset); // ["pomme", "kiwi"]
```

- `concat` : Fusionne plusieurs tableaux ou éléments + retourne un nouveau tableau.
  - Syntaxe : `concat(tableauxAAjouter)`

```javascript
const fruits = ["pomme", "kiwi"];
const fruitsEtLegumes = fruits.concat(["carotte", "brocoli"]);

console.log(fruitsEtLegumes); // ["pomme", "kiwi", "carotte", "brocoli"]
```

4.3. Recherche

- `indexOf` : Retourne l'indice du premier élément trouvé et `-1` si non trouvé.
  - Syntaxe : `indexOf(element)`

```javascript
const fruits = ["pomme", "kiwi", "banane"];

console.log(fruits.indexOf("kiwi")); // 1
console.log(fruits.indexOf("orange")); // -1
```

- `includes` : Vérifie si un élément est présent dans le tableau + retourne `true` ou `false` .
  - Syntaxe : `includes(element))`

```javascript
const fruits = ["pomme", "kiwi", "banane"];

console.log(fruits.includes("kiwi")); // true
console.log(fruits.includes("orange")); // false
```

5. Parcourir un tableau : itérations

5.1. *for*

> La boucle `for` classique permet une grande flexibilité.
>
> Cependant, elle est souvent plus verbeuse que d'autres méthodes modernes.
> Elle est utile lorsque vous avez besoin de contrôler l'itération de manière précise, comme en utilisant un indice.
>
> Note : Il est recommandé d'utiliser `for...of` ou `forEach` pour un code plus lisible et expressif.

Exemple:

Parcours d'un tableau avec la boucle `for`

- `fruits[i]` représente l'élément à l'indice `i` du tableau `fruits` .
- `fruits.length` donne la taille du tableau, ce qui permet de s'assurer que l'on ne dépasse pas les limites du tableau.
- `i++` incrémente `i` de 1 à chaque itération, permettant de passer à l'élément suivant du tableau.

```javascript
const fruits = ["pomme", "kiwi", "banane"];

for (let i = 0; i < fruits.length; i++) {
  console.log(fruits[i]);
}

// "pomme", "kiwi", "banane"
```

5.2. *for...of*

La boucle `for...of` est plus simple et lisible pour parcourir les éléments d'un tableau.

```javascript
const fruits = ["pomme", "kiwi", "banane"];

for (const fruit of fruits) {
  console.log(fruit);
}

// "pomme", "kiwi", "banane"
```

5.3. *forEach*

Simplifie le parcours d'un tableau.

```javascript
const fruits = ["pomme", "kiwi", "banane"];
fruits.forEach(fruit ⇒ console.log(fruit));
// "pomme", "kiwi", "banane"
```

6. Méthodes immutables extrêmement utiles et puissantes

6.1. *map*

Crée un nouveau tableau en appliquant une fonction sur chaque élément.
Elle est très utilisé dans des projets réels pour transformer des données sans prendre le risque de modifier le tableau d'origine.

Cas d'usage sur un site e-commerce : transformer les prix d'une liste de produits.

```
const fruits = ["pomme", "kiwi"];
const majuscules = fruits.map(fruit ⇒ fruit.toUpperCase());

console.log(fruits); // ["pomme", "kiwi"]
console.log(majuscules); // ["POMME", "KIWI"]
```

6.2. *filter*

Crée un nouveau tableau en filtrant les éléments selon une condition.
Si un élément passe le filtre (la condition est vraie), il est inclus dans le nouveau tableau.
Méthode très utile pour créer des sous-ensembles de données.

> Cas d'usage sur un site e-commerce : filtrer les produits selon un critère (ex: prix, catégorie).

```javascript
const fruits = ["pomme", "kiwi", "banane", "cerise", "poire"];
const fruitsCommencantParP = fruits.filter(fruit => fruit.startsWith("p"));

console.log(fruits); // ["pomme", "kiwi", "banane", "cerise", "poire"];
console.log(fruitsCommencantParP); // ["pomme", "poire"]
```

6.3. *reduce*

> Réduit tous les éléments du tableau à une seule valeur.
> Il est souvent utilisé pour calculer des totaux, accumuler des valeurs.
> Il prend une fonction de rappel et un accumulateur comme arguments.
> La fonction de rappel est appelée pour chaque élément du tableau, et l'accumulateur est mis à jour à chaque itération.

> Cas d'usage sur un site e-commerce : calculer le total d'une commande.

```javascript
const numbers = [1, 2, 3, 4];
const somme = numbers.reduce((total, nombre) => total + nombre, 0);

console.log(numbers); // [1, 2, 3, 4]
console.log(somme); // 10
```

7. Tableaux multidimensionnels

> Un tableau peut contenir d'autres tableaux.
> On appelle cela un tableau multidimensionnel.

Exemple :

```javascript
const tableau2D = [
  [1, 2, 3],
  [4, 5, 6],
  [7, 8, 9]
];

console.log(tableau2D[1]); // [4, 5, 6]
console.log(tableau2D[1][2]); // 6
```

Les boucles imbriquées sont souvent utilisées pour parcourir ces types de tableaux.

Exemple de parcours :

```javascript
for (const ligne of tableau2D) {
  for (const colonne of ligne) {
    console.log(colonne);
  }
}

// 1, 2, 3, 4, 5, 6, 7, 8, 9
```

8. Trier un tableau

La méthode `sort` permet de trier les éléments d'un tableau.

Elle modifie le tableau d'origine et retourne le tableau trié.

Par défaut, `sort` trie les éléments en ordre lexicographique (comme des chaînes de caractères).

Pour trier des nombres, il est nécessaire de fournir une fonction de comparaison.
Cette fonction doit retourner un nombre négatif, zéro ou positif selon l'ordre souhaité.

Il est important de noter que `sort` modifie le tableau d'origine.

Il est donc recommandé de faire une copie du tableau avant de le trier si vous souhaitez conserver l'ordre d'origine.

Pour ordonner un tableau contenant des chaînes de caractères, `sort` les trie par ordre lexicographique (comme dans un dictionnaire).

La méthode `localeCompare` peut être utilisée pour comparer des chaînes de caractères dans différentes langues en utilisant les règles de tri locales (ex: accents, etc.).
Par exemple, les chaînes "é" et "e" seront triées correctement en fonction de la langue choisie.

Dans le cas où vous souhaitez trier un tableau d'objets, vous pouvez également utiliser `sort` avec une fonction de comparaison personnalisée.

Cas d'usage sur un site e-commerce : trier les produits par prix ou par nom.

Exemple : Trier un tableau de nombres par ordre croissant

```javascript
const nombres = [4, 2, 5, 1, 3];

// copie du tableau avant de le trier
const croissants = [...nombres];

// tri du tableau
croissants.sort((a, b) => a - b);

console.log(croissants); // [1, 2, 3, 4, 5]
```

Exemple : Trier un tableau de nombres par ordre décroissant

```javascript
const nombres = [4, 2, 5, 1, 3];

// copie du tableau avant de le trier
const decroissants = [...nombres];

// tri du tableau
decroissants.sort((a, b) => b - a);

console.log(decroissants); // [5, 4, 3, 2, 1]
```

Exemple : Trier un tableau de chaînes de caractères

```javascript
const fruits = ["banane", "cerise", "pomme", "kiwi"];

// copie du tableau avant de le trier
const alphabetique = [...fruits];

// tri du tableau
alphabetique.sort();

console.log(alphabetique); // ["banane", "cerise", "kiwi", "pomme"]
```

Exemple : Trier un tableau de chaînes de caractères par ordre décroissant

```javascript
const mots = ["élève", "élevé", "élévation", "école", "é", "e", "ê", "è"];

// copie du tableau avant de le trier
const alphabetiqueDecroissant = [...mots];

// tri du tableau
alphabetiqueDecroissant.sort((a, b) => b.localeCompare(a));
console.log(alphabetiqueDecroissant); // ["é", "école", "élévation", "élevé",
"élève", "e", "ê", "è"]
```

Exemple : Trier un tableau d'objets

Ici nous souhaitons trier un tableau d'objets par prix croissant.

```javascript
const produits = [
  { nom: "pomme", prix: 1 },
  { nom: "banane", prix: 0.5 },
  { nom: "cerise", prix: 2 }
];

// copie du tableau avant de le trier
const produitsTries = [...produits];

// tri du tableau
produitsTries.sort((a, b) => a.prix - b.prix);
console.log(produitsTries);

// [
// { nom: "banane", prix: 0.5 },
// { nom: "pomme", prix: 1 },
// { nom: "cerise", prix: 2 }
// ]
```

9. Bonnes pratiques avec les tableaux

1. Utilisez les méthodes natives comme `map` et `filter` pour un code plus propre et expressif.
2. Évitez de modifier le tableau original lorsqu'une méthode retourne un nouveau tableau (comme `slice` au lieu de `splice` si possible).
3. Adoptez des noms explicites pour vos tableaux afin de rendre le code lisible.
4. Maîtrisez les tableaux multidimensionnels pour des données complexes.
5. N'oubliez pas que les tableaux sont des objets et non des types primitifs en JavaScript.

10. À RETENIR

- Les tableaux permettent de stocker et manipuler plusieurs éléments.
- Accédez aux éléments via leurs indices, en commençant par `0`.
- Manipulez les tableaux avec des méthodes comme `push`, `pop`, `splice`, et d'autres.
- Parcourez les tableaux avec `for`, `forEach`, ou des méthodes comme `map` et `filter`.
- Les tableaux multidimensionnels contiennent d'autres tableaux et sont utiles pour des structures complexes.
- Adoptez les meilleures pratiques pour un code clair et maintenable.

Les tableaux sont une base essentielle en JavaScript pour structurer et organiser vos données.
Un bon usage des méthodes et itérations vous permettra de les exploiter pleinement et efficacement !