Par Robert DIASSÉ

Les Tableaux et les Objets en JavaScript

Les Tableaux en JavaScript

Les tableaux en JavaScript sont des structures de données utilisées pour stocker une collection d'éléments. Ils peuvent contenir une variété de types de données, y compris des nombres, des chaînes de caractères, des objets, voire d'autres tableaux. Les tableaux permettent d'organiser et de manipuler efficacement des données dans vos programmes JavaScript.

Introduction aux Tableaux

Un tableau en JavaScript est une collection ordonnée d'éléments, indexée par des entiers, commençant par 0 pour le premier élément. Voici comment déclarer un tableau :

```
let fruits = ["Pomme", "Banane", "Orange", "Fraise"];
```

Dans cet exemple, nous avons déclaré un tableau nommé `fruits` contenant quatre éléments : "Pomme", "Banane", "Orange" et "Fraise".

Accès aux Éléments d'un Tableau

Pour accéder à un élément spécifique d'un tableau, vous pouvez utiliser son index entre crochets. Par exemple, pour accéder au deuxième élément du tableau `fruits`, vous pouvez utiliser `fruits[1]` :

```
console.log(fruits[1]); // Affiche "Banane"
```

Modification des Éléments d'un Tableau

Vous pouvez modifier les éléments d'un tableau en utilisant leur index. Par exemple, pour remplacer "Banane" par "Ananas", vous pouvez faire :

```

```

```
fruits[1] = "Ananas";
console.log(fruits); // Affiche ["Pomme", "Ananas", "Orange", "Fraise"]
```

Ajout et Suppression d'Éléments

Vous pouvez ajouter de nouveaux éléments à un tableau en utilisant la méthode `push()` ou en spécifiant directement un index. De même, vous pouvez supprimer des éléments en utilisant la méthode `pop()` pour supprimer le dernier élément, ou `splice()` pour supprimer un élément à un index spécifique.

```
// Ajouter un élément à la fin du tableau
fruits.push("Cerise");

// Supprimer le dernier élément du tableau
fruits.pop();

// Ajouter un élément à un index spécifique
fruits.splice(1, 0, "Kiwi");

// Supprimer un élément à un index spécifique
fruits.splice(2, 1);

console.log(fruits); // Affiche ["Pomme", "Kiwi", "Orange", "Fraise"]
```

Parcourir un Tableau

Vous pouvez parcourir tous les éléments d'un tableau en utilisant une boucle `for`, `while`, `do...while`, ou `forEach()`. Par exemple :

```
// Utilisation d'une boucle for
for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);
}

// Utilisation de la méthode forEach()
fruits.forEach(function(fruit) {
    console.log(fruit);
});
```

Utilisation de `while`

La boucle `while` continue d'exécuter son bloc de code tant qu'une condition donnée est vraie. Voici un exemple de parcours d'un tableau avec `while` :

```
let fruits = ["Pomme", "Banane", "Orange", "Fraise"];
let index = 0;
```

```
while (index < fruits.length) {
    console.log(fruits[index]);
    index++;
}
```

Utilisation de `do...while`

La boucle `do...while` est similaire à la boucle `while`, mais elle s'assure que le bloc de code est exécuté au moins une fois avant de vérifier la condition. Voici un exemple de parcours d'un tableau avec `do...while` :

```
let fruits = ["Pomme", "Banane", "Orange", "Fraise"];
let index = 0;

do {
    console.log(fruits[index]);
    index++;
} while (index < fruits.length);
```

Fonctions Couramment Utilisées avec les Tableaux

JavaScript fournit plusieurs fonctions intégrées pour travailler avec des tableaux, voici quelques-unes des plus couramment utilisées :

1. **push() :** Ajoute un ou plusieurs éléments à la fin du tableau.
2. **pop() :** Supprime le dernier élément du tableau et renvoie cet élément.
3. **shift() :** Supprime le premier élément du tableau et renvoie cet élément.
4. **unshift() :** Ajoute un ou plusieurs éléments au début du tableau.
5. **splice() :** Modifie le contenu d'un tableau en supprimant, remplaçant ou ajoutant des éléments.
6. **concat() :** Fusionne deux ou plusieurs tableaux en un seul tableau.
7. **slice() :** Extrait une partie d'un tableau et renvoie une nouvelle tableau.
8. **forEach() :** Exécute une fonction donnée sur chaque élément du tableau.
9. **map() :** Crée un nouveau tableau contenant les résultats de l'application d'une fonction à chaque élément du tableau d'origine.
10. **filter() :** Crée un nouveau tableau contenant les éléments du tableau d'origine qui passent un test spécifié.

Exemples d'utilisation de certaines fonctions importantes sur les tableaux :

Utilisation de `concat()`

La méthode `concat()` est utilisée pour fusionner deux ou plusieurs tableaux en un seul tableau. Voici un exemple :

```
let fruits = ["Pomme", "Banane", "Orange"];
let legumes = ["Carotte", "Pomme de terre", "Tomate"];
```

```
let aliments = fruits.concat(legumes);
console.log(aliments); // Affiche ["Pomme", "Banane", "Orange", "Carotte", "Pomme
de terre", "Tomate"]
```

Utilisation de `slice()`

La méthode `slice()` est utilisée pour extraire une partie d'un tableau et renvoyer un nouveau tableau. Voici un exemple :

```
let nombres = [1, 2, 3, 4, 5];
let sousTableau = nombres.slice(2, 4);
console.log(sousTableau); // Affiche [3, 4]
```

Utilisation de `map()`

La méthode `map()` est utilisée pour créer un nouveau tableau contenant les résultats de l'application d'une fonction à chaque élément du tableau d'origine. Voici un exemple :

```
let nombres = [1, 2, 3, 4, 5];
let doubles = nombres.map(function(nombre) {
    return nombre * 2;
});
console.log(doubles); // Affiche [2, 4, 6, 8, 10]
```

Ces exemples illustrent quelques-unes des fonctions importantes disponibles pour manipuler et travailler avec des tableaux en JavaScript.

Exercices

Exercice 1 : Création et Modification

1. Créez un tableau `nombres` contenant les nombres de 1 à 5.
2. Remplacez le troisième élément du tableau par 10.
3. Ajoutez les nombres 6 et 7 à la fin du tableau.
4. Affichez le contenu final du tableau.

Exercice 2 : Parcours et Affichage

1. Créez un tableau `couleurs` contenant les couleurs "Rouge", "Vert" et "Bleu".
2. Utilisez une boucle `for` pour afficher chaque couleur du tableau.

Exercice 3 : Suppression

1. Créez un tableau `voyelles` contenant les voyelles de l'alphabet (a, e, i, o, u).
2. Supprimez la dernière voyelle du tableau.
3. Affichez le contenu final du tableau.

Les Objets en JavaScript

Les objets en JavaScript sont des collections de propriétés, et une propriété est une association entre un nom (ou clé) et une valeur. Les valeurs des propriétés peuvent être des fonctions, ce qui permet aux objets de stocker des comportements ainsi que des données.

Introduction aux Objets

Voici comment déclarer un objet en JavaScript :

```
let personne = {
    nom: "Jean",
    age: 30,
    profession: "Développeur"
};
```

Dans cet exemple, nous avons déclaré un objet nommé `personne` avec trois propriétés : `nom`, `age`, et `profession`.

Accès aux Propriétés d'un Objet

Vous pouvez accéder aux propriétés d'un objet en utilisant la notation par point ou la notation par crochets. Par exemple :

```
// Notation par point
console.log(personne.nom); // Affiche "Jean"

// Notation par crochets
console.log(personne["age"]); // Affiche 30
```

Modification des Propriétés d'un Objet

Vous pouvez modifier les propriétés d'un objet en utilisant la notation par point ou par crochets. Par exemple, pour changer la profession de `personne` :

```
personne.profession = "Designer";
console.log(personne.profession); // Affiche "Designer"
```

Ajout et Suppression de Propriétés

Vous pouvez ajouter de nouvelles propriétés à un objet ou supprimer des propriétés existantes :

```
// Ajouter une nouvelle propriété
personne.nationalite = "Française";
```

```
// Supprimer une propriété
delete personne.age;

console.log(personne); // Affiche { nom: "Jean", profession: "Designer",
nationalite: "Française" }
```

Méthodes d'un Objet

Les méthodes sont des fonctions définies à l'intérieur d'un objet. Elles permettent aux objets d'avoir des comportements.

```
let voiture = {
    marque: "Toyota",
    modele: "Corolla",
    annee: 2020,
    afficherDetails: function() {
        console.log(`Marque: ${this.marque}, Modèle: ${this.modele}, Année:
${this.annee}`);
    }
};
```

L'utilisation du mot-clé `this` en JavaScript peut être un peu déroutante au début, mais il est crucial pour travailler efficacement avec des objets et des méthodes.

Comprendre `this` en JavaScript

Le mot-clé `this` fait référence à l'objet "propriétaire" actuel du contexte d'exécution. En d'autres termes, `this` fait référence à l'objet à partir duquel une méthode est appelée.

Décomposition de l'exemple

1. **Déclaration de l'objet `voiture` :**

   ```
   let voiture = {
       marque: "Toyota",
       modele: "Corolla",
       annee: 2020,
       afficherDetails: function() {
           console.log(`Marque: ${this.marque}, Modèle: ${this.modele}, Année:
   ${this.annee}`);
       }
   };
   ```

   - Ici, nous avons un objet `voiture` avec les propriétés `marque`, `modele` et `annee`.
   - L'objet contient également une méthode `afficherDetails`.

2. **Utilisation de `this` dans la méthode `afficherDetails` :**

```
afficherDetails: function() {
    console.log(`Marque: ${this.marque}, Modèle: ${this.modele}, Année:
${this.annee}`);
}
```

   - Dans cette méthode, `this` fait référence à l'objet `voiture`.
   - Lorsque vous appelez `voiture.afficherDetails()`, `this` à l'intérieur de la méthode fait référence à l'objet `voiture` lui-même.
   - Donc, `this.marque` accède à la propriété `marque` de l'objet `voiture`, `this.modele` accède à la propriété `modele`, et `this.annee` accède à la propriété `annee`.

3. **Appel de la méthode :**

```
voiture.afficherDetails(); // Affiche "Marque: Toyota, Modèle: Corolla,
Année: 2020"
```

   - Ici, nous appelons la méthode `afficherDetails` de l'objet `voiture`.
   - Lors de l'exécution de cette méthode, `this` fait référence à l'objet `voiture`, ce qui permet d'accéder à ses propriétés à l'intérieur de la méthode.

Importance de `this`

L'utilisation de `this` est essentielle pour rendre les méthodes réutilisables et pour accéder aux propriétés de l'objet auquel la méthode appartient. Sans `this`, il serait difficile d'écrire des méthodes génériques qui peuvent être utilisées par n'importe quel objet.

Exemple supplémentaire pour illustrer `this`

Imaginons que nous ayons un autre objet `voiture2` :

```
let voiture2 = {
    marque: "Honda",
    modele: "Civic",
    annee: 2022,
    afficherDetails: function() {
        console.log(`Marque: ${this.marque}, Modèle: ${this.modele}, Année:
${this.annee}`);
    }
};

// Appel de la méthode afficherDetails pour voiture2
voiture2.afficherDetails(); // Affiche "Marque: Honda, Modèle: Civic, Année: 2022"
```

Dans cet exemple :

- `this` dans la méthode `afficherDetails` de `voiture2` fait référence à l'objet `voiture2`.
- Même méthode, mais `this` s'adapte à l'objet actuel qui appelle la méthode.

Résumé

- **`this`** dans une méthode d'objet fait référence à l'objet qui appelle cette méthode.
- Cela permet aux méthodes d'accéder aux autres propriétés de l'objet.
- `this` est particulièrement utile pour écrire des méthodes génériques et réutilisables au sein des objets.

En comprenant et en utilisant `this`, vous pouvez créer des objets avec des méthodes qui peuvent interagir avec les propriétés de ces objets de manière dynamique et flexible.

Boucle sur les Propriétés d'un Objet

Vous pouvez utiliser une boucle `for...in` pour itérer sur toutes les propriétés d'un objet.

```
for (let propriete in personne) {
    console.log(propriete + ": " + personne[propriete]);
}
```

Exercices

Exercice 1 : Création et Modification

1. Créez un objet `livre` avec les propriétés `titre`, `auteur`, et `anneePublication`.
2. Modifiez la propriété `anneePublication` du livre.
3. Ajoutez une nouvelle propriété `genre` à l'objet `livre`.
4. Affichez l'objet final dans la console.

Exercice 2 : Boucle sur les Propriétés

1. Créez un objet `etudiant` avec les propriétés `nom`, `prenom`, `matricule`.
2. Utilisez une boucle `for...in` pour afficher toutes les propriétés et leurs valeurs.

Ces exercices permettent de pratiquer la création, la modification et l'itération des propriétés des objets en JavaScript, ainsi que l'ajout et l'utilisation de méthodes.