Objets, Arrays et leurs Méthodes

Structures de Données Essentielles

Pour les étudiants de 1ère année Supinfo

Au Programme

1. Les Objets - Collections de paires clé-valeurs
2. Les Arrays - Collections ordonnées de valeurs
3. Méthodes d'Arrays - find, filter, map, forEach, etc.

Qu'est-ce qu'un Objet ?

DÉFINITION

Un objet est une **collection de paires clé-valeurs** utilisée pour stocker des données structurées. Chaque clé est associée à une valeur qui peut être de n'importe quel type JavaScript.

CARACTÉRISTIQUES

- Les valeurs peuvent être de **n'importe quel type** (nombres, chaînes, booléens, fonctions, autres objets, etc.)
- Pour accéder à une valeur, on utilise la **clé associée**
- Les objets sont **mutables** : on peut ajouter, modifier ou supprimer des propriétés

SYNTAXE

```
const objet = {
clé: valeur,
clé2: valeur2,
clé3: valeur3
};
```

Créer et Accéder aux Propriétés

CRÉATION D'UN OBJET

```
const personne = {
nom: "Alice",
age: 25,
ville: "Paris"
};
```

ACCÈS AUX PROPRIÉTÉS

Notation Pointée

```
personne.nom
// "Alice"
```

Notation Crochets

```
personne["nom"]
// "Alice"
```

AJOUTER UNE PROPRIÉTÉ

```
personne.email = "alice@example.com";
// La propriété email est ajoutée à l'objet
```

Propriétés vs Méthodes

Propriétés

Une **propriété** est une valeur stockée dans un objet. Elle peut être de n'importe quel type (nombre, chaîne, booléen, objet, etc.).

```
const personne = {
nom: "Alice",
age: 25,
ville: "Paris"
};

// nom, age, ville
// sont des propriétés
```

Méthodes

Une **méthode** est une fonction stockée dans un objet. Elle permet d'effectuer des actions sur les données de l'objet.

```
const personne = {
nom: "Alice",
afficherNom: function() {
console.log(this.nom);
}
};

// afficherNom est une méthode
```

📌 **Résumé**

Dans l'exemple ci-dessus, l'objet **personne** contient **2 propriétés** (nom, age, ville) et **1 méthode** (afficherNom). Les propriétés stockent les données, les méthodes effectuent les actions.

Le Mot-clé 'this' dans les Objets

DÉFINITION

this fait référence au contexte d'exécution de la fonction. Dans un objet, **this** est l'objet lui-même et fait référence à l'objet qui contient la méthode.

UTILISATION DANS UN OBJET

```
const personne = {
nom: "Alice",
afficherNom: function() {
console.log(this.nom); // "Alice"
}
};
personne.afficherNom();
```

PIÈGE : FONCTIONS FLÉCHÉES

Une fonction fléchée ne fait **pas** référence au contexte de la fonction, mais au contexte englobant (la page du navigateur). Cela peut causer des erreurs !

```
const personne = {
nom: "Alice",
afficherNom: () => {
console.log(this.nom); // undefined ✗
}
};
```

> ⚠️ **Important**
>
> Utilisez des **fonctions classiques** pour les méthodes d'objets, pas des fonctions fléchées ! Les fonctions fléchées sont utiles pour les callbacks, mais pas pour les méthodes.

Propriété const et Modification d'Objets

IMPORTANT À COMPRENDRE

Même si un objet est déclaré avec **const**, il est possible de **modifier ses propriétés**. Le mot-clé **const** empêche la **réassignation** de la variable, pas la modification de ses propriétés.

✓ Autorisé

```
const personne = {
nom: "Alice",
age: 25
};

personne.age = 26;
// ✓ Modification OK
```

✗ Interdit

```
const personne = {
nom: "Alice",
age: 25
};

personne = { nom: "Bob" };
// ✗ Erreur !
```

BONNES PRATIQUES

Utilisez **const** pour déclarer les objets par défaut, même si vous allez modifier leurs propriétés. Cela empêche les réassignations accidentelles et rend votre code plus sûr et prévisible.

Qu'est-ce qu'un Array ?

DÉFINITION

Un array (tableau) est une **collection ordonnée de valeurs**, pouvant contenir différents types de données. Les éléments sont indexés à partir de **0**.

CARACTÉRISTIQUES

- Les éléments sont **indexés à partir de 0** : le premier élément est à l'index 0, le deuxième à l'index 1, etc.
- Peut contenir des **nombres, chaînes, booléens, objets, autres arrays**, etc.
- Accès par index : **array[0]**, **array[1]**, etc.

EXEMPLES

```
const fruits = ["pomme", "banane", "orange"];
const nombres = [1, 2, 3, 4, 5];
const mixte = [1, "deux", true, { nom: "Alice" }];
```

Propriétés et Méthodes Essentielles des Arrays

`.length`

Propriété qui donne le **nombre d'éléments** dans le tableau.

```
const fruits = ["pomme", "banane", "orange"];
console.log(fruits.length); // 3
```

`.push(element)`

Ajoute un élément à la **fin** du tableau. Modifie le tableau original.

```
fruits.push("raisin");
// ["pomme", "banane", "orange", "raisin"]
```

`.pop()`

Supprime et retourne le **dernier élément**. Modifie le tableau original.

```
const last = fruits.pop();
// last = "raisin"
```

`.shift()`

Supprime et retourne le **premier élément**. Modifie le tableau original.

```
const first = fruits.shift();
// first = "pomme"
```

`.unshift(element)`

Ajoute un élément au **début** du tableau. Modifie le tableau original.

```
fruits.unshift("fraise");
// ["fraise", "banane", "orange"]
```

`.slice(start, end)`

Retourne une **copie** d'une partie du tableau. Ne modifie pas l'original.

```
const partie = fruits.slice(0, 2);
// ["fraise", "banane"]
```

`.indexOf(element)`

Retourne l'**index** du premier élément trouvé, ou -1 si non trouvé.

```
const index = fruits.indexOf("banane");
// 1
```

Méthodes de Recherche : .find() et .findIndex()

`.find()`

Retourne le **premier élément** qui satisfait la condition spécifiée dans la fonction callback. Retourne **undefined** si aucun élément ne correspond.

```
const utilisateurs = [
{ id: 1, nom: "Alice" },
{ id: 2, nom: "Bob" }
];

const user = utilisateurs
.find(u => u.id === 2);

// Retourne:
// { id: 2, nom: "Bob" }
```

`.findIndex()`

Retourne l'**index du premier élément** qui satisfait la condition spécifiée. Retourne **-1** si aucun élément ne correspond.

```
const utilisateurs = [
{ id: 1, nom: "Alice" },
{ id: 2, nom: "Bob" }
];

const index = utilisateurs
.findIndex(u => u.id === 2);

// Retourne: 1
```

💡 **Différence Clé**

.find() retourne l'élément lui-même, tandis que **.findIndex()** retourne sa position dans le tableau. Utilisez **.find()** quand vous avez besoin de l'élément, et **.findIndex()** quand vous avez besoin de sa position pour le modifier ou le supprimer.

Méthodes de Vérification : .every() et .some()

`.every()`

Retourne **true** si **tous les éléments** satisfont la condition, **false** sinon.

```
const nombres = [2, 4, 6, 8];
const tousParirs = nombres.every(
n => n % 2 === 0
);
// Retourne: true

const nombres2 = [1, 3, 5, 8];
const tousParirs2 = nombres2.every(
n => n % 2 === 0
);
// Retourne: false (1, 3, 5 ne sont pas pairs)
```

`.some()`

Retourne **true** si **au moins un élément** satisfait la condition, **false** sinon.

```
const nombres = [1, 3, 5, 8];
const unPair = nombres.some(
n => n % 2 === 0
);
// Retourne: true (8 est pair)

const nombres2 = [1, 3, 5, 7];
const unPair2 = nombres2.some(
n => n % 2 === 0
);
// Retourne: false (aucun pair)
```

💡 **Différence Clé**

.every() vérifie que **TOUS** les éléments satisfont la condition (ET logique). **.some()** vérifie qu'**AU MOINS UN** élément satisfait la condition (OU logique). Ces deux méthodes retournent un booléen (true ou false) et ne modifient pas le tableau original.

Méthodes de Transformation : .filter() et .map()

.filter()

Retourne un **nouveau tableau** contenant les éléments qui satisfont la condition. Les éléments qui ne satisfont pas la condition sont exclus.

```
const nombres = [1, 2, 3, 4, 5, 6];
const pairs = nombres.filter(
n => n % 2 === 0
);
// Retourne: [2, 4, 6]
```

.map()

Retourne un **nouveau tableau** en appliquant une fonction à chaque élément. Transforme chaque élément selon la fonction fournie.

```
const nombres = [1, 2, 3];
const doubles = nombres.map(
n => n * 2
);
// Retourne: [2, 4, 6]
```

💡 Différences Clés

.filter() sélectionne les éléments qui satisfont une condition (réduit la taille du tableau). **.map()** transforme chaque élément (garde la même taille). Tous les deux retournent un nouveau tableau sans modifier l'original.

Itération : .forEach()

DÉFINITION

.forEach() exécute une fonction pour **chaque élément** du tableau. Ne retourne rien (undefined). C'est la façon moderne et lisible d'itérer sur tous les éléments.

EXEMPLE SIMPLE

```
const fruits = ["pomme","banane","orange"]; fruits.forEach(f => console.log(f)); // Affiche: pomme, banane, orange
```

AVEC INDEX ET ARRAY

La fonction callback peut recevoir jusqu'à **3 paramètres** : l'élément, l'index, et le tableau complet.

```
fruits.forEach((fruit, i) => console.log(`${i}: ${fruit}`));
```

COMPARAISON : .FOREACH() VS FOR

forEach (moderne)

```
fruits.forEach(f => console.log(f));
```

for classique (ancien)

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

> 💡 **Quand utiliser .forEach() ?**
>
> Utilisez **.forEach()** quand vous avez besoin d'exécuter une action pour chaque élément. Préférez **.map()** si vous avez besoin d'un nouveau tableau, et **.filter()** pour sélectionner certains éléments.

Bonnes Pratiques et Résumé

OBJETS

- Utilisez les objets pour stocker des **données structurées** avec des clés significatives
- Utilisez **fonctions classiques** pour les méthodes, pas les fonctions fléchées (à cause de **this**)
- Déclarez les objets avec **const**, même si vous modifiez leurs propriétés

ARRAYS

- Utilisez les arrays pour stocker des **collections ordonnées** de valeurs
- Préférez les **méthodes modernes** (.map(), .filter(), .forEach()) aux boucles for classiques
- **Chaînez les méthodes** pour des opérations complexes et lisibles

CHAÎNAGE DE MÉTHODES

```
const resultat = nombres
.filter(n => n > 5)
.map(n => n * 2)
.forEach(n => console.log(n));
```

Questions ?

N'hésitez pas à poser vos questions ! La maîtrise des **Objets et Arrays** est essentielle pour manipuler les données en JavaScript.

POUR APPROFONDIR

📚 **MDN Web Docs** - Documentation officielle et complète

🔗 **JavaScript.info** - Tutoriels interactifs et détaillés

💻 **Eloquent JavaScript** - Livre gratuit en ligne

🎓 **Supinfo Resources** - Matériel pédagogique de l'école

BESOIN D'AIDE ?

Consultez vos instructeurs ou les ressources Supinfo