JavaScript - Tableaux, objets et JSON

Jusqu'ici on a manipulé des valeurs isolées : un nom, un âge, un score. Pour les vrais programmes, on a besoin de structurer des données plus riches : une liste d'élèves, une fiche utilisateur, un panier d'achat. Les **tableaux** stockent des listes, les **objets** stockent des fiches structurées, et **JSON** est le format universel pour les échanger.

5TTR 6TTR niveau

Les tableaux — une liste ordonnée

Un tableau (`Array`) stocke **plusieurs valeurs dans une seule variable**, accessibles par leur position.

```
let fruits = ["pomme", "banane", "cerise"];
```

Chaque case a un **index**, qui commence à **0** (pas à 1 !).

```
console.log(fruits[0]);    // "pomme"
console.log(fruits[1]);    // "banane"
console.log(fruits[2]);    // "cerise"
console.log(fruits.length); // 3
```

Indexation à partir de 0. C'est la source de bug n°1 chez les débutants. Le premier élément est à l'index `0`, le dernier à `length - 1`. Avec 5 éléments → indices 0, 1, 2, 3, 4. Jamais 5.

Modifier un tableau

```
let scores = [10, 20, 30];

scores[1] = 25;             // modifier un élément
scores.push(40);            // ajouter à la fin    → [10, 25, 30, 40]
scores.pop();               // retirer le dernier  → [10, 25, 30]
scores.unshift(5);          // ajouter au début    → [5, 10, 25, 30]
scores.shift();             // retirer le premier  → [10, 25, 30]
```

Parcourir un tableau

© 2025 Mr. Lambrechts

```js
let fruits = ["pomme", "banane", "cerise"];

// Avec un for classique (utile si on a besoin de l'index)
for (let i = 0; i < fruits.length; i++) {
    console.log(`${i} : ${fruits[i]}`);
}

// Avec for...of (plus lisible si on veut juste les valeurs)
for (let fruit of fruits) {
    console.log(fruit);
}

// Avec forEach (style moderne)
fruits.forEach(fruit => {
    console.log(fruit);
});
```

Les objets — une fiche structurée

Un objet stocke des **paires clé/valeur**. Idéal pour décrire une chose qui a plusieurs caractéristiques.

```js
let personne = {
    nom: "Alice",
    age: 16,
    classe: "5TTr",
    estPresent: true
};
```

On accède aux valeurs par leur **nom de clé** (pas par index — on n'a pas d'ordre ici).

```js
console.log(personne.nom);        // "Alice"
console.log(personne.age);        // 16
console.log(personne["classe"]);  // "5TTr"  (notation alternative)
```

Dot notation vs bracket notation.

- `personne.nom` — la plus courante, à utiliser quand la clé est connue à l'écriture.
- `personne["nom"]` — utile quand la clé est dans une variable : `personne[champ]` .

Modifier un objet

```js
personne.age = 17;              // modifier
personne.email = "a@ex.com";    // ajouter (la clé n'existait pas)
```

© 2025 Mr. Lambrechts

```
delete personne.estPresent;     // supprimer
```

Tableau d'objets — le pattern le plus utile

C'est la structure de données que tu verras partout : une liste de fiches.

```
let eleves = [
    { nom: "Alice", age: 16, moyenne: 14 },
    { nom: "Bob",   age: 17, moyenne: 11 },
    { nom: "Carol", age: 16, moyenne: 17 }
];

for (let eleve of eleves) {
    console.log(`${eleve.nom} : ${eleve.moyenne}/20`);
}
```

C'est exactement comme ça qu'on représente une base de données simple, une liste de produits, un classement…

Tableau ou objet ?

Question	Réponse
Une liste ordonnée d'éléments du même type ?	**Tableau** (`[]`)
Une fiche avec des champs nommés ?	**Objet** (`{}`)
Une liste de fiches ?	**Tableau d'objets**

```
// ✅ Tableau : liste de scores du même type
let scores = [10, 20, 30, 25, 18];

// ✅ Objet : fiche structurée
let personne = { nom: "Alice", age: 16 };

// ✅ Tableau d'objets : liste de personnes
let eleves = [
    { nom: "Alice", age: 16 },
    { nom: "Bob",   age: 17 }
];

// ❌ Objet avec des clés "0", "1", "2" — utilise un tableau !
let mauvais = { 0: "a", 1: "b", 2: "c" };
```

© 2025 Mr. Lambrechts

JSON — le format d'échange

JSON (*JavaScript Object Notation*) est un **format texte** dérivé directement de la syntaxe des objets JavaScript. C'est le format universel pour échanger des données entre :

- une page web et un serveur,
- un programme et un fichier de configuration,
- deux applications différentes,
- même deux langages de programmation différents (Python lit aussi le JSON).

Visuellement, ça ressemble à un objet JS, avec deux différences :

1. Les clés sont **toujours entre guillemets**.
2. Pas de variables, pas de fonctions, pas de commentaires — juste des données.

```
{
    "nom": "Alice",
    "age": 16,
    "classe": "5TTr",
    "matieres": ["math", "info", "anglais"]
}
```

`JSON.stringify` — objet → texte

Pour stocker un objet ou l'envoyer sur le réseau, on doit le convertir en **chaîne de caractères**.

```
let personne = { nom: "Alice", age: 16 };

let texte = JSON.stringify(personne);
console.log(texte);
// '{"nom":"Alice","age":16}'
```

Pratique pour stocker dans `localStorage` ou envoyer via `fetch`.

`JSON.parse` — texte → objet

L'inverse : récupérer un objet utilisable à partir d'un texte JSON (typiquement reçu d'un serveur).

```
let texte = '{"nom":"Alice","age":16}';

let personne = JSON.parse(texte);
console.log(personne.nom);     // "Alice"
console.log(personne.age + 1); // 17
```

> `JSON.parse` **PLANTE SI LE TEXTE N'EST PAS DU JSON VALIDE.** Une virgule en trop, un guillemet manquant, une clé sans guillemets — et tout casse. Si tu reçois du JSON depuis

© 2025 Mr. Lambrechts

l'extérieur, encadre `JSON.parse` d'un `try / catch` (sujet plus avancé).

Exemple complet : afficher une liste

Combinaison typique : un tableau d'objets, parcouru avec une boucle, qui remplit une page HTML.

```
<ul id="liste"></ul>

<script>
    let eleves = [
        { nom: "Alice", moyenne: 14 },
        { nom: "Bob",   moyenne: 11 },
        { nom: "Carol", moyenne: 17 }
    ];

    const liste = document.getElementById("liste");

    for (let eleve of eleves) {
        const li = document.createElement("li");
        li.textContent = `${eleve.nom} — ${eleve.moyenne}/20`;
        liste.appendChild(li);
    }
</script>
```

Avec dix lignes, on a tout : la structure de données, la boucle, et l'affichage dynamique. C'est le pattern de base de toute application web.

À retenir

- **Tableau** `[ ]` : liste ordonnée, accès par index commençant à **0**, `length` donne la taille.
- **Objet** `{ }` : fiche structurée, accès par nom de clé : `obj.cle`.
- **Tableau d'objets** : combinaison ultra-courante (liste d'éléments, panier, classement…).
- **JSON** : format texte universel, clés entre guillemets obligatoires.
- `JSON.stringify(obj)` → texte. `JSON.parse(texte)` → objet.
- Liste de choses similaires → tableau. Une chose avec plusieurs champs → objet. Liste de fiches → tableau d'objets.

© 2025 Mr. Lambrechts