Rappel JavaScript - Les essentiels pour React

Pourquoi ce rappel ?

React utilise beaucoup de fonctionnalités **JavaScript moderne (ES6+)**. Avant de commencer React, assurez-vous de maîtriser ces concepts.

---

1. Variables : let, const, var

```
// var : ancienne syntaxe, à éviter (scope fonction, hoisting)
var nom = "Victor";

// let : variable qui peut être réassignée
let age = 26;
age = 27;  // OK

// const : variable qui ne peut PAS être réassignée
const PI = 3.14159;
PI = 3;  // ERREUR !

// ⚠ ATTENTION : const ne rend pas l'objet immuable !
const user = { name: "Victor" };
user.name = "Alice";  // OK, on modifie une propriété
user = { name: "Bob" };  // ERREUR, on réassigne la variable
```

Règle en React

> **Utilisez `const` par défaut**, `let` si vous devez réassigner. **Jamais `var`**.

---

2. Template Literals (Backticks)

```
const name = "Victor";
const age = 26;

// Ancienne syntaxe
const message1 = "Bonjour, je suis " + name + " et j'ai " + age + " ans.";

// Template literals (backticks `)
const message2 = `Bonjour, je suis ${name} et j'ai ${age} ans.`;

// Multi-lignes
const html = `
  <div>
    <h1>${name}</h1>
    <p>Age: ${age}</p>
  </div>
`;

// Expressions dans ${}
const message3 = `Dans 10 ans, j'aurai ${age + 10} ans.`;
```

---

3. Fonctions Fléchées (Arrow Functions)

```
// Fonction classique
function addition(a, b) {
  return a + b;
}

// Fonction fléchée
const addition = (a, b) => {
  return a + b;
};

// Fonction fléchée courte (return implicite)
const addition = (a, b) => a + b;

// Un seul paramètre : parenthèses optionnelles
const double = n => n * 2;

// Pas de paramètre : parenthèses obligatoires
const sayHello = () => "Hello!";

// Retourner un objet : entourer de parenthèses
const createUser = (name) => ({ name: name, id: Date.now() });
```

En React

```
// Handlers d'événements
<button onClick={() => setCount(count + 1)}>+1</button>

// map() avec arrow function
{users.map(user => <UserCard key={user.id} user={user} />)}
```

---

4. Déstructuration (Destructuring)

Déstructuration d'objet

```
const user = {
  name: "Victor",
  age: 26,
  city: "Grenoble"
};

// Sans déstructuration
const name = user.name;
const age = user.age;

// Avec déstructuration
const { name, age, city } = user;

// Renommer une variable
const { name: userName } = user;
console.log(userName);  // "Victor"

// Valeur par défaut
const { name, country = "France" } = user;
console.log(country);  // "France" (n'existe pas dans user)
```

Déstructuration de tableau

```
const colors = ["rouge", "vert", "bleu"];

// Sans déstructuration
const first = colors[0];
const second = colors[1];

// Avec déstructuration
const [first, second, third] = colors;

// Ignorer des éléments
const [first, , third] = colors;  // "rouge", "bleu"

// useState en React utilise ça !
const [count, setCount] = useState(0);
//      |        |
//      |        └── 2ème élément du tableau retourné
//      └── 1er élément du tableau retourné
```

Déstructuration dans les paramètres (React++)

```
// Sans déstructuration
function UserCard(props) {
  return <h1>{props.name}</h1>;
}

// Avec déstructuration
function UserCard({ name, age, city }) {
  return <h1>{name}</h1>;
}
```

---

5. Spread Operator (...)

Copier et étendre des tableaux

```
const numbers = [1, 2, 3];

// Copier un tableau
const copy = [...numbers];

// Ajouter des éléments
const more = [...numbers, 4, 5];  // [1, 2, 3, 4, 5]
const more2 = [0, ...numbers];    // [0, 1, 2, 3]

// Fusionner des tableaux
const all = [...numbers, ...more];
```

Copier et étendre des objets

```
const user = { name: "Victor", age: 26 };

// Copier un objet
const copy = { ...user };

// Ajouter/modifier des propriétés
const updated = { ...user, age: 27, city: "Grenoble" };
// { name: "Victor", age: 27, city: "Grenoble" }

// ⚠ L'ordre compte ! La dernière valeur gagne
const result = { ...user, name: "Alice" };  // name = "Alice"
```

En React (mise à jour du state)

```
// Ajouter un élément à un tableau
setItems([...items, newItem]);

// Modifier un objet
setUser({ ...user, name: "Alice" });

// ⚠ ERREUR COURANTE : muter directement
items.push(newItem);  // NON ! Ne déclenche pas de re-render
setItems(items);      // NON ! Même référence
```

---

6. Méthodes de tableau : map, filter, find

map() - Transformer chaque élément

```
const numbers = [1, 2, 3, 4];

// Doubler chaque nombre
const doubled = numbers.map(n => n * 2);
// [2, 4, 6, 8]

// Extraire une propriété
const users = [
  { id: 1, name: "Victor" },
  { id: 2, name: "Alice" }
];
const names = users.map(user => user.name);
// ["Victor", "Alice"]
```

filter() - Garder certains éléments

```
const numbers = [1, 2, 3, 4, 5, 6];

// Garder les pairs
const evens = numbers.filter(n => n % 2 === 0);
// [2, 4, 6]

// Garder les utilisateurs actifs
const users = [
  { name: "Victor", active: true },
  { name: "Alice", active: false }
];
const activeUsers = users.filter(user => user.active);
// [{ name: "Victor", active: true }]
```

find() - Trouver UN élément

```
const users = [
  { id: 1, name: "Victor" },
  { id: 2, name: "Alice" }
];

// Trouver par id
const user = users.find(u => u.id === 2);
// { id: 2, name: "Alice" }

// Si non trouvé, retourne undefined
const notFound = users.find(u => u.id === 99);
// undefined
```

Chaîner les méthodes

```
const users = [
  { name: "Victor", age: 26, active: true },
  { name: "Alice", age: 17, active: true },
  { name: "Bob", age: 32, active: false }
];

// Garder les actifs majeurs et extraire leurs noms
const names = users
  .filter(u => u.active)
  .filter(u => u.age >= 18)
  .map(u => u.name);
// ["Victor"]
```

---

7. Opérateur Ternaire

```
// Syntaxe : condition ? siVrai : siFaux

const age = 20;
const status = age >= 18 ? "Majeur" : "Mineur";

// Équivalent à :
let status;
if (age >= 18) {
  status = "Majeur";
} else {
  status = "Mineur";
}

// Imbriqué (à éviter si trop complexe)
const category = age < 13 ? "Enfant" : age < 18 ? "Ado" : "Adulte";
```

En React (rendu conditionnel)

```
// Afficher différent contenu selon une condition
{isLoggedIn ? <Dashboard /> : <LoginForm />}

// Afficher ou non un élément
{hasError && <ErrorMessage />}

// Classes conditionnelles
<div className={`btn ${isActive ? 'active' : ''}`}>
```

---

8. Short-circuit Evaluation (&&, ||)

```
// && : retourne la 1ère valeur falsy OU la dernière valeur
true && "Hello"     // "Hello"
false && "Hello"    // false
null && "Hello"     // null
"Hi" && "Hello"     // "Hello"

// || : retourne la 1ère valeur truthy OU la dernière valeur
false || "Default"  // "Default"
null || "Default"   // "Default"
"Hi" || "Default"   // "Hi"

// ?? (nullish coalescing) : seulement si null/undefined
0 || "Default"      // "Default" (0 est falsy)
0 ?? "Default"      // 0 (0 n'est pas null/undefined)
```

En React

```
// Afficher si condition vraie
{isLoading && <Spinner />}
{error && <ErrorMessage error={error} />}
{items.length > 0 && <ItemList items={items} />}

// Valeur par défaut
const name = user.name || "Anonyme";
const count = data?.count ?? 0;
```

---

9. Optional Chaining (?.)

```
const user = {
  name: "Victor",
  address: {
    city: "Grenoble"
  }
};

// Sans optional chaining
const city = user && user.address && user.address.city;

// Avec optional chaining
const city = user?.address?.city;  // "Grenoble"

// Si le chemin n'existe pas, retourne undefined (pas d'erreur)
const country = user?.address?.country;  // undefined

// Avec des tableaux
const firstItem = items?.[0];

// Avec des fonctions
const result = obj?.method?.();
```

---

10. Import / Export (Modules ES6)

Export nommé

```
// utils.js
export const PI = 3.14159;
export function add(a, b) { return a + b; }
export const multiply = (a, b) => a * b;

// Import (accolades obligatoires, nom exact)
import { PI, add } from './utils.js';
import { add as addition } from './utils.js';  // Renommer
import * as Utils from './utils.js';  // Tout importer
Utils.add(1, 2);
```

Export par défaut

```
// UserCard.jsx
function UserCard({ user }) {
  return <div>{user.name}</div>;
}
export default UserCard;

// Import (pas d'accolades, nom au choix)
import UserCard from './UserCard';
import MonComposant from './UserCard';  // Nom différent OK
```

Combiné

```
// api.js
export const API_URL = "https://api.example.com";
export function fetchData() { ... }
export default class ApiClient { ... }

// Import
import ApiClient, { API_URL, fetchData } from './api.js';
```

---

11. Promesses et Async/Await

Promesses

```
// Créer une promesse
const promise = new Promise((resolve, reject) => {
  setTimeout(() => {
    resolve("Succès !");
    // ou reject("Erreur !");
  }, 1000);
});

// Utiliser une promesse
promise
  .then(result => console.log(result))
  .catch(error => console.error(error))
  .finally(() => console.log("Terminé"));
```

Async/Await (syntaxe moderne)

```
// Fonction async
async function fetchUser(id) {
  try {
    const response = await fetch(`/api/users/${id}`);
    const user = await response.json();
    return user;
  } catch (error) {
    console.error("Erreur:", error);
    throw error;
  }
}

// Utilisation
const user = await fetchUser(1);

// Ou avec .then()
fetchUser(1).then(user => console.log(user));
```

En React (useEffect)

```
useEffect(() => {
  // ⚠ useEffect ne peut PAS être async directement
  async function loadData() {
    const response = await fetch('/api/data');
    const data = await response.json();
    setData(data);
  }

  loadData();
}, []);
```

---

12. Tableau récapitulatif

Concept	Utilisation en React
`const` / `let`	Déclarer des variables
Template literals	JSX, strings dynamiques
Arrow functions	Handlers, callbacks, composants
Déstructuration	Props, state (useState)
Spread operator	Mise à jour immutable du state
`map()`	Afficher des listes
`filter()`	Filtrer des données
Ternaire `? :`	Rendu conditionnel
`&&`	Afficher si condition vraie
`?.`	Accès sécurisé aux propriétés
Import/Export	Organiser les composants
Async/Await	Appels API dans useEffect