📘 JavaScript : Concepts Avancés

🔹 1. Le Scope (portée des variables)

Le *scope* d'une variable définit l'endroit où elle est accessible dans le code.

➤ Scope global vs local :

JAVASCRIPT

```
let a = 10; // portée globale

function test() {
  let b = 20; // portée locale
  console.log(a); // OK
  console.log(b); // OK
}

console.log(a); // OK
console.log(b); // Erreur
```

➤ Bloc de scope (avec let/const) :

JAVASCRIPT

```
{
  let x = 5;
  console.log(x); // OK
}
console.log(x); // Erreur
```

> ⚠️ `var` n'a pas de bloc de scope (seulement fonctionnel).

2. Le Hoisting

Le *hoisting* signifie que les déclarations de variables (avec `var`) et de fonctions sont "remontées" en haut de leur scope.

➤ Exemple avec `var` :

```javascript
console.log(x); // undefined
var x = 10;
```

> JavaScript le comprend comme :

```javascript
var x;
console.log(x); // undefined
x = 10;
```

➤ Avec `let` et `const` : pas de hoisting utilisable !

```javascript
console.log(y); // Erreur
let y = 5;
```

3. Les fonctions fléchées

Une syntaxe plus courte pour définir des fonctions.

```javascript
// Fonction classique
function direBonjour(nom) {
  return "Bonjour " + nom;
```

```
}

// Fonction fléchée
const direBonjour = (nom) => "Bonjour " + nom;

console.log(direBonjour("Maimouna")); // Bonjour Maimouna
```

> Elles ne possèdent pas leur propre `this`.

🔹 4. Les Callbacks

Un *callback* est une fonction passée en argument à une autre fonction.

➤ Exemple :

JAVASCRIPT

```
function traitement(callback) {
  console.log("Traitement en cours...");
  callback();
}

traitement(function() {
  console.log("Callback exécuté !");
});
```

🔹 5. setTimeout & setInterval

➤ `setTimeout` : exécuter une fois après un délai

```javascript
setTimeout(function() {
  console.log("Hello après 2 secondes");
}, 2000);
```

➤ `setInterval` : exécuter régulièrement

```javascript
setInterval(function() {
  console.log("Je s'affiche toutes les 3s");
}, 3000);
```

🔹 6. Les Promesses (Promises)

Les promesses permettent de gérer des opérations asynchrones comme des requêtes serveur.

➤ Création d'une promesse :

```javascript
let promesse = new Promise(function(resolve, reject) {
  let reussi = true;

  if (reussi) {
    resolve("Succès !");
  } else {
    reject("Erreur...");
  }
});

promesse
  .then(function(resultat) {
    console.log(resultat); // Succès !
  })
  .catch(function(erreur) {
```

```
  console.log(erreur); // Erreur...
});
```

🔹 7. async / await

Une façon plus simple d'écrire du code asynchrone.

➤ Exemple :

JAVASCRIPT

```
function attendre(ms) {
  return new Promise(resolve => setTimeout(resolve, ms));
}

async function demarrer() {
  console.log("Début");
  await attendre(2000);
  console.log("2 secondes plus tard");
}

demarrer();
```