Résumé JavaScript

Concept	Explication	Exemple de Code
1. Variables	Stocke des données. Utilise let, const ou var.	let x = 10; const y = "Hello";
2. Types de données	JavaScript supporte les types : nombres, chaînes, booléens, objets, etc.	let age = 25; let name = "Alice"; let isStudent = true;
3. Opérateurs arithmétiques	Pour effectuer des calculs mathématiques.	let sum = 5 + 3; // 8
4. Opérateurs de comparaison	Compare deux valeurs (==, ===, >, <, etc.).	let isEqual = (5 === "5"); // false
6. Conditions if	Exécute un bloc de code si une condition est vraie.	if (age > 18) { console.log("Majeur"); }
7. Conditions else	Exécute un bloc de code si la condition if est fausse.	if (age > 18) { console.log("Majeur"); } else { console.log("Mineur"); }
8. Conditions else if	Ajoute des conditions supplémentaires.	if (age < 13) { console.log("Enfant"); } else if (age < 18) { console.log("Ado"); }
9. Switch	Exécute un bloc de code en fonction d'une valeur.	switch (day) { case "Mon": console.log("Lundi"); break; }
10. Boucle for	Répète un bloc de code un nombre spécifié de fois.	for (let i = 0; i < 5; i++) { console.log(i); }
11. Boucle while	Répète un bloc de code tant qu'une condition est vraie.	let i = 0; while (i < 5) { console.log(i); i++; }
12. Boucle do...while	Exécute le bloc de code au moins une fois, puis vérifie la condition.	let i = 0; do { console.log(i); i++; } while (i < 5);
13. Fonctions	Bloc de code réutilisable. Peut prendre des paramètres et retourner une valeur.	function add(a, b) { return a + b; } console.log(add(2, 3)); // 5
14. Fonctions fléchées	Syntaxe concise pour écrire des fonctions.	const add = (a, b) => a + b; console.log(add(2, 3)); // 5
15. Tableaux (Arrays)	Structure de données pour stocker plusieurs valeurs.	let fruits = ["Apple", "Banana", "Orange"]; console.log(fruits[1]); // Banana
16. Méthode push()	Ajoute un élément à la fin d'un tableau.	fruits.push("Mango"); // ["Apple", "Banana", "Orange", "Mango"]
17. Méthode pop()	Supprime le dernier élément d'un tableau.	fruits.pop(); // ["Apple", "Banana"]
18. Méthode map()	Crée un nouveau tableau en appliquant une fonction à chaque élément.	let doubled = [1, 2, 3].map(n => n * 2); // [2, 4, 6]

19. Méthode filter()	Filtre les éléments d'un tableau selon une condition.	let evens = [1, 2, 3, 4].filter(n => n % 2 === 0); // [2, 4]
20. Méthode reduce()	Réduit un tableau à une seule valeur en appliquant une fonction.	let sum = [1, 2, 3].reduce((acc, n) => acc + n, 0); // 6
21. Objets	Collection de paires clé-valeur.	let person = { name: "John", age: 30 }; console.log(person.name); // John
22. Accès aux propriétés	Accéder aux propriétés d'un objet.	console.log(person["name"]); // John
23. Méthodes d'objets	Fonctions définies dans un objet.	let person = { greet() { console.log("Hello"); } }; person.greet();
24. Classes	Permet de créer des objets avec des propriétés et méthodes.	class Person { constructor(name) { this.name = name; } greet() { console.log("Hello"); } }
25. Héritage	Une classe peut hériter des propriétés et méthodes d'une autre classe.	class Student extends Person { study() { console.log("Studying"); } }
26. Manipulation du DOM	Permet de modifier le contenu et la structure d'une page web.	document.getElementById("demo").innerHTML = "Hello, World!";
27. Sélection d'éléments	Sélectionne un élément HTML par son ID, classe ou balise.	let element = document.querySelector(".myClass");
28. Événements	Gère les interactions utilisateur (clics, souris, etc.).	button.addEventListener("click", () => { alert("Button clicked!"); });
29. Promesses	Gère les opérations asynchrones.	fetch('url').then(response => response.json()).then(data => console.log(data));
30. Async/Await	Syntaxe moderne pour gérer les promesses de manière synchrone.	async function fetchData() { let data = await fetch('url'); console.log(data); }
31. Manipulation de chaînes	Méthodes pour travailler avec des chaînes de caractères.	let str = "Hello"; console.log(str.toUpperCase()); // HELLO
32. Concaténation	Combine des chaînes de caractères.	let greeting = "Hello, " + "World!";
33. Template literals	Permet d'insérer des variables dans des chaînes.	let name = "Alice"; console.log(`Hello, ${name}!`);
34. Manipulation de dates	Permet de travailler avec des dates et heures.	let date = new Date(); console.log(date.getFullYear()); // Année actuelle
35. Manipulation de tableaux	Méthodes pour manipuler des tableaux (map, filter, reduce, etc.).	let numbers = [1, 2, 3]; let doubled = numbers.map(n => n * 2); // [2, 4, 6]
36. Gestion des erreurs	Capture et gère les erreurs avec try...catch.	try { riskyCode(); } catch (error) { console.log("Erreur : ", error); }
37. Manipulation du JSON	Convertit des objets JavaScript en chaînes JSON et vice versa.	let obj = { name: "Alice" }; let json = JSON.stringify(obj); // {"name":"Alice"}

38. Modules	Permet d'organiser le code en fichiers séparés.	// fichier math.js export function add(a, b) { return a + b; }
39. Import/Export	Importe et exporte des fonctions ou variables entre fichiers.	import { add } from './math.js'; console.log(add(2, 3)); // 5
40. Fonctions récursives	Une fonction qui s'appelle elle-même.	function factorial(n) { return n <= 1 ? 1 : n * factorial(n - 1); }
41. Closure	Une fonction qui capture et conserve son environnement lexical.	function outer() { let x = 10; function inner() { console.log(x); } return inner; }
42. Hoisting	Les déclarations de variables et fonctions sont déplacées en haut de leur scope.	console.log(x); // undefined var x = 5;
43. this	Fait référence à l'objet courant.	let obj = { name: "John", greet() { console.log(this.name); } }; obj.greet();
44. bind()	Crée une nouvelle fonction avec un this fixé.	let boundFunc = obj.greet.bind(obj); boundFunc();
45. call() et apply()	Appelle une fonction avec un this spécifié et des arguments.	obj.greet.call(obj); obj.greet.apply(obj);
46. Set	Collection de valeurs uniques.	let set = new Set([1, 2, 3]); set.add(4);
47. Map	Collection de paires clé-valeur.	let map = new Map(); map.set("name", "Alice");
48. WeakSet et WeakMap	Variantes de Set et Map pour le garbage collection.	let weakSet = new WeakSet(); weakSet.add({});
49. Proxy	Permet de personnaliser le comportement d'un objet.	let proxy = new Proxy(target, handler);
50. Symbol	Type de données unique et immuable, souvent utilisé comme clé d'objet.	let id = Symbol("id"); let obj = { [i

1. **Variables** : let et const sont utilisés pour déclarer des variables. const est immuable.
2. **Types de données** : JavaScript est dynamiquement typé, pas besoin de déclarer le type.
3. **Opérateurs** : === vérifie à la fois la valeur et le type.
4. **Conditions** : if exécute un bloc de code si la condition est vraie.
5. **Boucles** : for répète un bloc de code un nombre spécifié de fois.
6. **Fonctions** : Une fonction peut retourner une valeur ou non.
7. **Tableaux** : Les tableaux sont indexés à partir de 0.
8. **Objets** : Les objets contiennent des propriétés et des méthodes.
9. **DOM** : Permet de manipuler les éléments HTML.
10. **Événements** : Réagit aux actions de l'utilisateur.
11. **Promesses** : Utilisé pour les opérations asynchrones comme les requêtes réseau.
12. **Async/Await** : Simplifie la gestion des promesses.
13. **Classes** : Permet de créer des objets avec un constructeur et des méthodes.
14. **Modules** : Organise le code en fichiers réutilisables.
15. **Chaînes** : Méthodes comme toUpperCase() modifient les chaînes.
16. **Dates** : Date permet de manipuler des dates et heures.
17. **Tableaux** : Méthodes comme map transforment les tableaux.
18. **Erreurs** : try...catch capture les erreurs pour éviter les plantages.
19. **JSON** : JSON.stringify convertit un objet en chaîne JSON.
20. **Fonctions fléchées** : Syntaxe concise pour les fonctions anonymes.