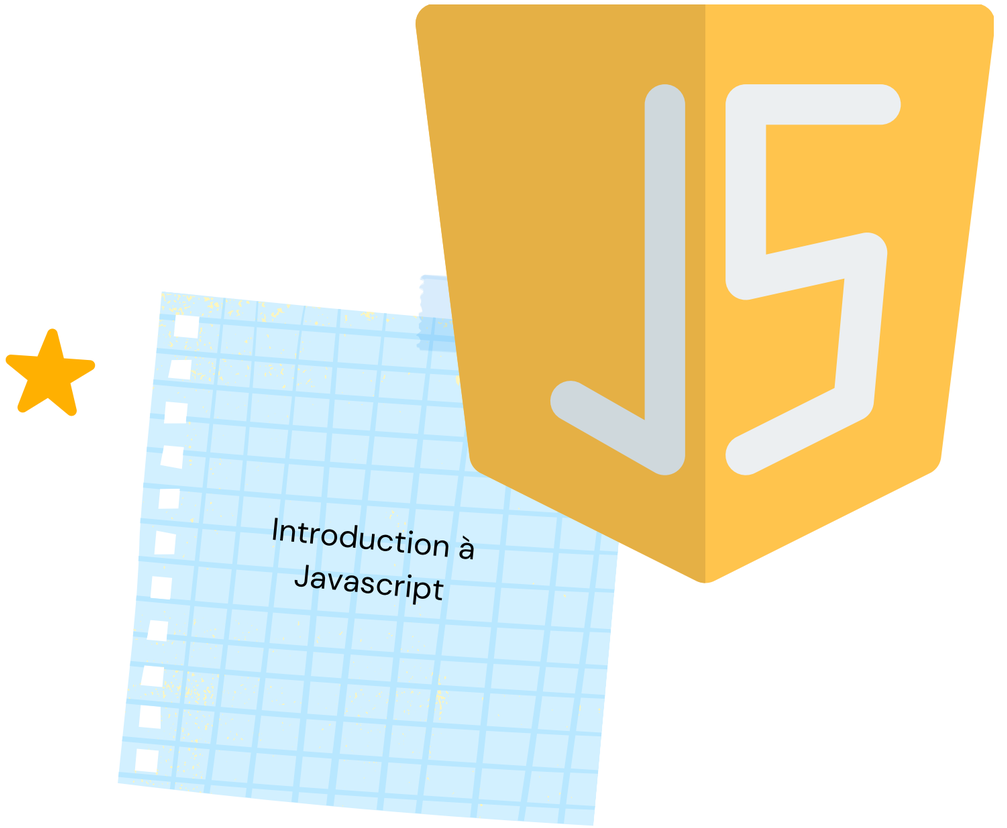
BAH SOUMANY Oualid
Kolawole

2024

# Javascript (part-1)



BAH SOUMANY Oualid
Kolawole

2024

# Programme d'Aujourd'hui

1. Introduction à JavaScript
2. Configuration de l'environnement
3. Écrire le premier script
4. Variables et Types de Données
3. Opérateurs et Expressions
4. Boucles et Fonctions
3. Tableaux et Objets
4. DOM et Événements

# Introduction

## Qu'est-ce que Javascript ?

Le langage de programmation JavaScript est principalement utilisé pour ajouter de l'interactivité aux pages web. Il permet de créer des fonctionnalités dynamiques comme des formulaires interactifs, des animations et des menus déroulants.

JavaScript peut être utilisé pour programmer des serveurs, permettant de gérer les requêtes des utilisateurs et de communiquer avec des bases de données

JavaScript est utilisé pour écrire des scripts qui automatisent des tâches sur des pages web.

# COMPARAISON

- HTML : Structure le contenu (squelettes de la page).
- CSS : Style le contenu (apparence et mise en page).
- JavaScript : Dynamise le contenu (interaction et comportement).

- Installation d'un éditeur de code (VS Code, Sublime Text, etc.)

2024

# IDE DE BASE

# Console de Développement

La console de développement est un outil intégré dans les navigateurs web qui permet d'écrire, tester et déboguer du code JavaScript directement sur une page web.

**Pourquoi l'utiliser ?**

- Tester du code rapidement : Vous pouvez exécuter du JavaScript sans créer de fichiers.
- Déboguer : Voir les erreurs et les messages de log pour résoudre les problèmes.

**Accéder à la console sur Google Chrome**

- Ouvrez Chrome.
- Appuyez sur Ctrl + Shift + J (Windows/Linux) ou Cmd + Option + J (macOS).
- Ou cliquez avec le bouton droit sur une page web et sélectionnez "Inspecter", puis allez à l'onglet "Console".

Pour intégrer JavaScript dans une page HTML, il existe principalement trois méthodes : les scripts internes, les scripts externes et les événements HTML.

# Script Interne

Vous pouvez écrire du JavaScript directement dans votre fichier HTML à l'intérieur de balises <script>.

```
<!DOCTYPE html>
<html>
<head>
    <title>Ma Page Web</title>
</head>
<body>
    <h1>Bienvenue sur ma page web</h1>
    <script>
        // JavaScript intégré directement dans le HTML
        console.log('Bonjour depuis un script interne !');
        alert('Bienvenue sur ma page web !');
    </script>
</body>
</html>
```

# Script Externe

Vous pouvez placer votre code JavaScript dans un fichier séparé et l'inclure dans votre fichier HTML à l'aide de la balise <script> avec l'attribut src.

## Fichier HTML :

```
<!DOCTYPE html>
<html>
<head>
    <title>Ma Page Web</title>
</head>
<body>
    <h1>Bienvenue sur ma page web</h1>
    <script src="script.js"></script>
</body>
</html>
```

## Fichier JavaScript (script.js) :

```
// JavaScript dans un fichier externe
console.log('Bonjour depuis un script externe !');
alert('Bienvenue sur ma page web !');
```

# Événements HTML

Vous pouvez aussi intégrer JavaScript directement dans les attributs HTML pour répondre à des événements utilisateur comme des clics de bouton.

```
<!DOCTYPE html>
<html>
<head>
    <title>Ma Page Web</title>
</head>
<body>
    <h1>Bienvenue sur ma page web</h1>
    <button onclick="alert('Bouton cliqué !')">Cliquez-moi</button>
</body>
</html>
```

La fonction console.log en JavaScript est utilisée pour afficher des messages dans la console de développement du navigateur. Cela est très utile pour le débogage et pour voir la sortie de votre code.

## EXERCICE :

- Ouvrir la Console de Développement sur votre navigateur
- Écrire du Code JavaScript pour afficher le message *“Bienvenue à la session juin 2024 de RightCom Académie.”*

BAH SOUMANY Oualid
Kolawole

2024

# Les Bases de JavaScript

En JavaScript, les variables sont utilisées pour stocker des données. Il existe trois façons principales de déclarer des variables : **var, let et const**. Chacune a ses propres caractéristiques et usages.

# var

- **Scope (Portée) :** La portée de var est la fonction dans laquelle elle est déclarée ou globale si elle est déclarée en dehors de toute fonction.
- **Re-déclaration :** Vous pouvez redéclarer une variable var dans la même portée sans erreur.
- **Hoisting :** Les variables déclarées avec var sont "hoistées", c'est-à-dire que leur déclaration est déplacée en haut de leur portée, mais pas leur initialisation.

```
var message = 'Bonjour';
console.log(message);  // Affiche 'Bonjour'

var message = 'Hello';  // Re-déclaration permise
console.log(message);  // Affiche 'Hello'
```

# Let

- **Scope (Portée) :** La portée de let est le bloc dans lequel elle est déclarée (délimité par {}).
- **Re-déclaration :** Vous ne pouvez pas redéclarer une variable let dans la même portée.
- **Hoisting :** Les variables déclarées avec let sont hoistées mais non initialisées. Elles ne sont accessibles qu'après leur déclaration.

```
let greeting = 'Salut';
console.log(greeting);  // Affiche 'Salut'

// let greeting = 'Hi';   // Erreur : Identifier 'greeting' has already been
declared

if (true) {
    let greeting = 'Hello';
    console.log(greeting);  // Affiche 'Hello' (portée du bloc)
}

console.log(greeting);  // Affiche 'Salut' (portée globale)
```

# CONST

- **Scope (Portée) :** La portée de const est le bloc dans lequel elle est déclarée (délimité par {}).
- **Re-déclaration :** Vous ne pouvez pas redéclarer une variable const dans la même portée.
- **Constante :** Une fois assignée, la valeur d'une variable const ne peut pas être changée. Cependant, si la variable contient un objet ou un tableau, les propriétés de cet objet ou les éléments de ce tableau peuvent être modifiés.
- **Hoisting :** Les variables déclarées avec const sont hoistées mais non initialisées. Elles ne sont accessibles qu'après leur déclaration.

```
const PI = 3.14159;
console.log(PI);  // Affiche 3.14159

// PI = 3.14;  // Erreur : Assignment to constant variable

const fruits = ['Pomme', 'Banane'];
fruits.push('Orange');
console.log(fruits);  // Affiche ['Pomme', 'Banane', 'Orange']

// const fruits = ['Cerise'];  // Erreur : Identifier 'fruits' has already been declared
```

# En Résumé

- **var :** Utiliser pour les variables avec une portée de fonction. Permet la redéclaration. Sujet au hoisting.
- let : Utiliser pour les variables avec une portée de bloc. Ne permet pas la redéclaration dans la même portée. Protégé contre les problèmes de hoisting.
- const : Utiliser pour les constantes avec une portée de bloc. Ne permet pas la redéclaration ni la réassignation, mais les objets et tableaux peuvent être modifiés.

**Utilisez let et const de préférence à var pour éviter des comportements inattendus et améliorer la lisibilité et la maintenance de votre code.**

**En JavaScript, il existe cinq types de données primaires : Number, String, Boolean, Null, et Undefined.**

# Number

Représente les nombres, qu'ils soient entiers ou à virgule flottante.

```
let age = 25;         // Entier
let temperature = 36.6;  // Nombre à virgule flottante
```

# String

Représente des chaînes de caractères, utilisées pour le texte.

```
let greeting = "Bonjour";
let name = 'Alice';
```

# Boolean

Représente une valeur logique pouvant être true ou false

```
let isSunny = true;
let hasFinished = false;
```

# Null

Représente l'absence intentionnelle de valeur.

```
let emptyValue = null;
```

# Undefined

Indique qu'une variable a été déclarée mais n'a pas encore été assignée.

```
let notAssigned;
console.log(notAssigned);  // Affiche undefined
```

# En Résumé

- **Number :** Nombres (entiers ou flottants).
- **String :** Chaînes de caractères (texte).
- **Boolean :** Valeurs logiques (true ou false).
- **Null :** Absence intentionnelle de valeur.
- **Undefined :** Variable déclarée, mais non assignée.

BAH SOUMANY Oualid
Kolawole

2024

# Opérateurs et Expressions

JavaScript comprend plusieurs types d'opérateurs qui sont essentiels pour effectuer des opérations arithmétiques, de comparaison et logiques. Voici une explication brève de chaque type

# Opérateurs Arithmétiques

Addition (+) : Additionne deux valeurs.

```
let sum = 10 + 5;  // sum vaut 15
```

Soustraction (-) : Soustrait la seconde valeur de la première.

```
let difference = 10 - 5;  // difference vaut 5
```

Multiplication (*) : Multiplie deux valeurs.

```
let product = 10 * 5;  // product vaut 50
```

Division (/) : Divise la première valeur par la seconde.

```
let quotient = 10 / 5;  // quotient vaut 2
```

Modulo (%) : Donne le reste de la division de la première valeur par la seconde.

```
let remainder = 10 % 3;
```

# Opérateurs de Comparaison

**Égal (==) :** Vérifie si deux valeurs sont égales, en convertissant les types si nécessaire.

Égal Strict (===) : Vérifie si deux valeurs sont égales en comparant le type et la valeur.

Différent (!= ou !==) : Vérifie si deux valeurs sont différentes.

Supérieur (>), Inférieur (<), Supérieur ou égal (>=), Inférieur ou égal (<=) : Comparaisons numériques.

# Opérateurs de Comparaison (EXEMPLE)

```
let x = 10;
let y = 5;

console.log(x == y);    // false
console.log(x === '10');  // false (type différent)
console.log(x !== y);   // true
console.log(x > y);     // true
```

# Opérateurs Logiques

ET (&&) : Renvoie true si les deux expressions sont vraies.

OU (||) : Renvoie true si au moins l'une des expressions est vraie.

**NON (!) :** Inverse la valeur booléenne de l'expression.

# Opérateurs Logiques (EXEMPLE)

```
let a = true;
let b = false;

console.log(a && b);   // false
console.log(a || b);   // true
console.log(!a);       // false
```

# En Résumé

- Opérateurs Arithmétiques : +, -, *, /, %
- Opérateurs de Comparaison : ==, ===, !=, !==, >, <, >=, <=
- Opérateurs Logiques : &&, ||, !

Ces opérateurs sont fondamentaux pour manipuler et comparer des valeurs en JavaScript, et ils sont largement utilisés dans le développement web pour créer des conditions, des boucles et effectuer des calculs.

Les expressions sont des éléments clés en programmation JavaScript car elles permettent de combiner des valeurs et des opérateurs pour produire un résultat.

# Assignation de Valeurs

Vous pouvez utiliser des expressions pour assigner des valeurs à des variables :

```
let a = 5;  // Expression simple
let b = a * 2;  // Utilisation d'une expression (a * 2) pour assigner une valeur à b
```

# Calculs et Opérations

Les expressions sont souvent utilisées pour effectuer des calculs :

```
let total = (5 + 3) * 2;  // Utilisation d'une expression complexe pour calculer total
```

# Conditions et Comparaisons

Les expressions sont couramment utilisées dans les structures conditionnelles :

```
let age = 18;
let isAdult = age >= 18;  // Utilisation d'une expression pour vérifier si age est supérieur ou égal à 18
```

# Concaténation de Chaînes

Pour concaténer des chaînes de caractères, utilisez des expressions :

```
let firstName = 'John';
let lastName = 'Doe';
let fullName = firstName + ' ' + lastName;  // Expression pour concaténer des chaînes de caractères
```

# Appels de Fonction

Les expressions peuvent être utilisées pour passer des arguments à des fonctions :

```
function addNumbers(a, b) {
    return a + b;
}

let result = addNumbers(3, 4);  // Utilisation d'une expression (addNumbers(3, 4)) pour appeler une fonction
```

# Utilisation dans les Boucles

Les expressions sont couramment utilisées pour définir les conditions des boucles :

```
for (let i = 0; i < 5; i++) {
    console.log(i);  // Utilisation d'expressions (i < 5 et i++) pour contrôler la boucle
}
```

# Utilisation dans les Boucles

Les expressions sont couramment utilisées pour définir les conditions des boucles :

```
for (let i = 0; i < 5; i++) {
    console.log(i);  // Utilisation d'expressions (i < 5 et i++) pour contrôler la boucle
}
```

# Avantages des Expressions

- **Flexibilité :** Elles permettent de combiner différents types de données et d'opérateurs pour obtenir des résultats variés.
- **Lisibilité :** En utilisant des expressions claires et concises, votre code devient plus facile à comprendre et à maintenir.
- **Réutilisabilité :** Vous pouvez réutiliser des expressions dans différentes parties de votre code pour éviter la redondance.

En résumé, les expressions sont essentielles en JavaScript car elles facilitent la manipulation de données, le contrôle de flux et la création de fonctionnalités dynamiques dans vos applications web.

BAH SOUMANY Oualid
Kolawole

2024

# Conditions

En JavaScript, les structures de contrôle if, else, et else if permettent d'exécuter du code conditionnel en fonction de certaines conditions :

# if

Utilisé pour exécuter un bloc de code si une condition est vraie.

```
let age = 18;

if (age >= 18) {
    console.log("Personne majeure");
}
```

# else

Optionnellement utilisé avec if pour exécuter un bloc de code si la condition n'est pas vraie.

```
let age = 15;

if (age >= 18) {
    console.log("Personne majeure");
} else {
    console.log("Personne mineure");
}
```

# else if

Utilisé pour spécifier une nouvelle condition si la première condition n'est pas vraie.

```
let time = 14;

if (time < 12) {
    console.log("Bonjour");
} else if (time < 18) {
    console.log("Bon après-midi");
} else {
    console.log("Bonsoir");
}
```

# En Résumé

- **if :** Exécute un bloc de code si une condition est vraie.
- **else :** Exécute un bloc de code si la condition if n'est pas vraie.
- **else if :** Exécute un bloc de code si une condition précédente n'est pas vraie mais que la condition actuelle l'est.

Ces structures de contrôle sont fondamentales pour diriger le flux d'exécution dans votre programme en fonction des conditions spécifiées.

Les opérateurs ternaires sont une manière concise et efficace d'écrire des expressions conditionnelles en JavaScript.

**Structure de l'Opérateur Ternaire**

(condition) ? valeurSiVraie : valeurSiFausse

Les opérateurs ternaires sont une manière concise et efficace d'écrire des expressions conditionnelles en JavaScript.

**Structure de l'Opérateur Ternaire**

(condition) ? valeurSiVraie : valeurSiFausse

- Condition : Une expression évaluée comme vrai ou faux.
- valeurSiVraie : Valeur renvoyée si la condition est vraie.
- valeurSiFausse : Valeur renvoyée si la condition est fausse.

# Exemple

## Exemple Simple

```
let age = 20;
let message = (age >= 18) ? 'Majeur' : 'Mineur';
console.log(message);  // Affiche 'Majeur'
```

## Assignation de Valeurs

```
let isLoggedIn = false;
let status = (isLoggedIn) ? 'Connecté' : 'Déconnecté';
console.log(status);  // Affiche 'Déconnecté'
```

## Utilisation dans une Chaîne de Caractères

```
let isRaining = true;
let weather = (isRaining) ? 'Il pleut' : 'Il ne pleut pas';
console.log(weather);  // Affiche 'Il pleut'
```

# En Résumé

**Avantages des Opérateurs Ternaires**

- **Concision :** Permet d'écrire des conditions en une seule ligne, ce qui rend le code plus compact.
- **Lisibilité :** Bien utilisés, ils peuvent rendre le code plus clair et facile à comprendre.
- **Équivalence avec if/else :** Ils peuvent être utilisés là où une structure if/else simple est nécessaire, mais de manière plus concise.

**Points à Considérer**

- **Complexité :** Les opérateurs ternaires peuvent rendre le code difficile à lire s'ils sont utilisés de manière excessive ou si la condition est complexe.
- **Usage approprié :** Idéaux pour les conditions simples où vous devez déterminer une seule valeur en fonction d'une condition.

BAH SOUMANY Oualid
Kolawole

2024

# Boucles et Fonctions

En JavaScript, il existe plusieurs types de boucles pour répéter des blocs de code tant qu'une condition est vraie ou pour itérer sur des collections de données.

# Boucle for

La boucle for est utilisée lorsque le nombre d'itérations est connu à l'avance.

```
for (let i = 0; i < 5; i++) {
    console.log(i);
}
```

- **Initialisation (let i = 0) :** Déclare et initialise la variable de contrôle i.
- **Condition (i < 5) :** La boucle continue tant que cette condition est vraie.
- **Incrémentation (i++) :** Incrémente la variable de contrôle après chaque itération.

# Boucle while

La boucle while est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance, mais la condition de continuation est évaluée avant l'exécution du bloc de code.

```
let i = 0;
while (i < 5) {
    console.log(i);
    i++;
}
```

- **Condition (i < 5) :** La boucle continue tant que cette condition est vraie.
- **Incrémentation (i++) :** Assurez-vous de mettre à jour la variable de contrôle (i dans cet exemple) à l'intérieur de la boucle pour éviter une boucle infinie.

# Boucle do...while

La boucle do...while est similaire à while, mais elle garantit qu'au moins une itération du bloc de code est exécutée avant d'évaluer la condition de continuation.

```
let i = 0;
do {
    console.log(i);
    i++;
} while (i < 5);
```

- **Bloc de code (console.log(i); i++;) :** Le bloc de code à exécuter au moins une fois.
- **Condition (i < 5) :** La boucle continue tant que cette condition est vraie après la première itération.

# Choix de la Boucle

- **for :** Utilisé lorsque le nombre d'itérations est connu à l'avance.
- **while :** Utilisé lorsque le nombre d'itérations n'est pas défini à l'avance.
- **do...while :** Utilisé lorsque vous voulez vous assurer qu'un bloc de code s'exécute au moins une fois, même si la condition est fausse dès le départ.

# Points Importants

- Assurez-vous de mettre à jour correctement la variable de contrôle à l'intérieur de la boucle pour éviter les boucles infinies.
- Utilisez chaque type de boucle en fonction de la logique et des besoins spécifiques de votre programme.

En JavaScript, les fonctions sont des blocs de code réutilisables qui peuvent être déclarés et appelés à plusieurs reprises dans votre programme.

# Fonction Nomée

## Déclaration

```
f// Déclaration d'une fonction nommée
function greet(name) {
    console.log(`Bonjour, ${name} !`);
}
```

## Invocation

```
// Appel de la fonction
greet('Alice');  // Affiche "Bonjour, Alice !"
```

# Fonction Anonyme (assignée à une variable)

## Déclaration

```
// Déclaration d'une fonction anonyme
let greet = function(name) {
    console.log(`Bonjour, ${name} !`);
};
```

## Invocation

```
// Appel de la fonction
greet('Alice');  // Affiche "Bonjour, Alice !"
```

# Fonction fléchée

## Déclaration

```
// Déclaration d'une fonction fléchée
let greet = (name) => {
    console.log(`Bonjour, ${name} !`);
};
```

## Invocation

```
// Appel de la fonction
greet('Alice');  // Affiche "Bonjour, Alice !"
```

# Fonctions avec Valeur de Retour

Les fonctions peuvent également retourner une valeur à l'endroit où elles sont appelées.

```
// Déclaration d'une fonction qui retourne une valeur
function add(a, b) {
    return a + b;
}

// Appel de la fonction et utilisation du résultat
let sum = add(3, 4);
console.log(sum);  // Affiche 7
```

# Utilisation de Fonctions dans des Expressions

Les fonctions peuvent être utilisées comme n'importe quelle autre valeur en JavaScript, y compris être passées comme arguments à d'autres fonctions.

```
// Fonction qui prend une autre fonction comme argument
function greet(name, callback) {
    let message = `Bonjour, ${name} !`;
    callback(message);
}

// Appel de la fonction avec une fonction anonyme comme callback
greet('Bob', function(msg) {
    console.log(msg);  // Affiche "Bonjour, Bob !"
});
```

# Résumé

En résumé, déclarer et invoquer des fonctions en JavaScript est essentiel pour structurer et réutiliser votre code de manière efficace, en encapsulant des actions et des calculs spécifiques dans des blocs de code facilement appelables à partir d'autres parties de votre programme.

BAH SOUMANY Oualid
Kolawole

2024

# Tableaux et Objets

En JavaScript, les tableaux sont des structures de données flexibles permettant de stocker plusieurs valeurs dans une seule variable.

# Déclaration de Tableaux

## Déclaration d'un tableau vide

```
let fruits = [];  // Déclaration d'un tableau vide
```

## Initialisation d'un tableau avec des éléments

```
let fruits = ['Pomme', 'Banane', 'Orange'];  // Tableau avec des éléments initiaux
```

# Accès aux Éléments d'un Tableau

Les éléments d'un tableau sont indexés à partir de zéro.
Vous pouvez accéder à un élément en utilisant son index.

```
let fruits = ['Pomme', 'Banane', 'Orange'];  // Tableau avec des éléments initiaux

console.log(fruits[0]);  // Affiche le premier élément : "Pomme"
console.log(fruits[1]);  // Affiche le deuxième élément : "Banane"
```

# Modification d'un Élément du Tableau

Vous pouvez modifier un élément existant en utilisant son index.

```
let fruits = ['Pomme', 'Banane', 'Orange'];  // Tableau avec des éléments initiaux

fruits[2] = 'Fraise';  // Modifie le troisième élément
console.log(fruits);   // Affiche le tableau mis à jour : ['Pomme', 'Banane', 'Fraise']
```

# Ajout et Suppression d'Éléments

```
let fruits = ['Pomme', 'Banane', 'Orange'];  // Tableau avec des éléments initiaux
```

Ajout d'éléments à la fin du tableau

```
fruits.push('Mangue');  // Ajoute 'Mangue' à la fin du tableau
console.log(fruits);    // Affiche le tableau avec 'Mangue' ajouté : ['Pomme', 'Banane', 'Fraise', 'Mangue']
```

Suppression du dernier élément du tableau

```
fruits.pop();  // Supprime le dernier élément du tableau ('Mangue')
console.log(fruits);  // Affiche le tableau mis à jour : ['Pomme', 'Banane', 'Fraise']
```

# Propriété length d'un Tableau

La propriété length vous permet de connaître le nombre d'éléments dans un tableau.

```
let fruits = ['Pomme', 'Banane', 'Orange'];  // Tableau avec des éléments initiaux

console.log(fruits.length);  // Affiche la longueur du tableau : 3
```

# Boucler à travers un Tableau

Vous pouvez utiliser différentes boucles pour parcourir les éléments d'un tableau.

```
let fruits = ['Pomme', 'Banane', 'Orange'];  // Tableau avec des éléments initiaux
```

Boucle for

```
for (let i = 0; i < fruits.length; i++) {
    console.log(fruits[i]);  // Affiche chaque élément du tableau
}
```

Boucle for...of

```
for (let fruit of fruits) {
    console.log(fruit);  // Affiche chaque élément du tableau
}
```

# Méthodes de Manipulation de Tableaux

JavaScript propose plusieurs méthodes intégrées pour manipuler les tableaux, telles que **push**, **pop**, **shift**, **unshift**, **splice**, **slice**, **concat**, **indexOf**, **includes**, etc. Ces méthodes facilitent l'ajout, la suppression, la recherche et la manipulation des éléments d'un tableau.

# Résumé

les tableaux en JavaScript sont des structures de données puissantes et flexibles qui vous permettent de stocker et de manipuler efficacement des collections d'éléments. Ils sont largement utilisés dans le développement web pour gérer des données structurées comme des listes d'éléments, des résultats de requêtes, etc.

les objets sont des structures de données permettant de stocker des collections de propriétés et de méthodes.

# Création d'Objets

Il existe plusieurs façons de créer des objets en JavaScript :

## Objet Literal

```
let person = {
    firstName: 'John',
    lastName: 'Doe',
    age: 30,
    fullName: function() {
        return `${this.firstName} ${this.lastName}`;
    }
};
```

# Création d'Objets

## Utilisation de new Object()

```
let person = new Object();
person.firstName = 'John';
person.lastName = 'Doe';
person.age = 30;
person.fullName = function() {
    return `${this.firstName} ${this.lastName}`;
};
```

# Création d'Objets

## Fonction Constructeur

```
function Person(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.fullName = function() {
        return `${this.firstName} ${this.lastName}`;
    };
}

let person = new Person('John', 'Doe', 30);
```

# Création d'Objets

## Fonction Constructeur

```
function Person(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName = lastName;
    this.age = age;
    this.fullName = function() {
        return `${this.firstName} ${this.lastName}`;
    };
}

let person = new Person('John', 'Doe', 30);
```

# Accès aux Propriétés d'un Objet

Vous pouvez accéder aux propriétés d'un objet en utilisant la notation pointée (.) ou la notation entre crochets ([]).

```
console.log(person.firstName);  // Affiche 'John'
console.log(person['lastName']);  // Affiche 'Doe'
```

# Modification des Propriétés d'un Objet

Vous pouvez modifier les propriétés d'un objet simplement en affectant de nouvelles valeurs.

```
person.age = 35;
console.log(person.age);  // Affiche 35
```

# Ajout de Nouvelles Propriétés

Vous pouvez ajouter de nouvelles propriétés à un objet à tout moment.

```
person.email = 'john.doe@example.com';
console.log(person.email);  // Affiche 'john.doe@example.com'
```

# Méthodes dans les Objets

Les objets en JavaScript peuvent contenir des fonctions, appelées méthodes, qui peuvent être appelées comme n'importe quelle autre fonction.

```
console.log(person.fullName());  // Affiche 'John Doe'
```

# Itération sur les Propriétés d'un Objet

Vous pouvez itérer sur les propriétés d'un objet à l'aide de la boucle **for...in**.

```
for (let key in person) {
    console.log(`${key}: ${person[key]}`);
}
```

# Suppression de Propriétés d'un Objet

Vous pouvez supprimer une propriété d'un objet à l'aide de l'opérateur **delete**.

```
delete person.age;
console.log(person.age);  // Affiche 'undefined'
```

# Résumé

Les objets en JavaScript sont des structures de données puissantes qui permettent d'organiser des données complexes et d'encapsuler la logique dans des méthodes, facilitant ainsi le développement d'applications web robustes et dynamiques.

BAH SOUMANY Oualid
Kolawole

2024

**« Apprendre, c'est la seule chose que l'esprit ne regrettera jamais. »**

Léonard de Vinci

BAH SOUMANY Oualid
Kolawole

2024

# Merci !