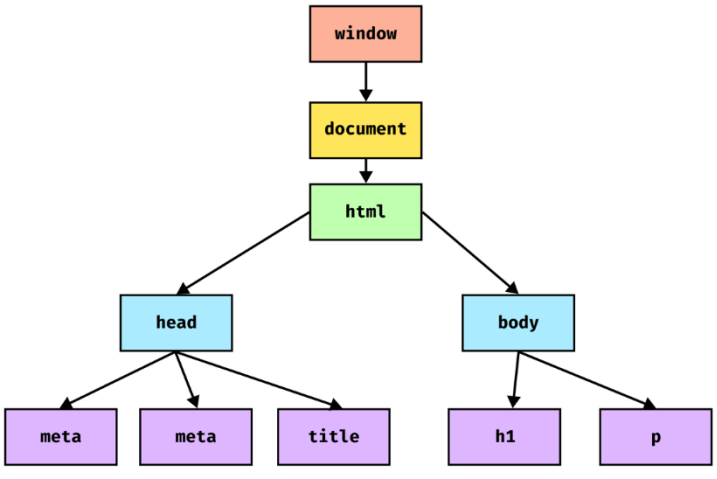
I. Introduction :

JavaScript est un langage de programmation essentiel pour le développement web. Il est principalement utilisé pour ajouter de l'interactivité aux sites web, comme les animations, la validation de formulaires, ou les mises à jour dynamiques sans recharger la page.

Avec JavaScript, on peut manipuler le DOM (Document Object Model) pour modifier dynamiquement le contenu et le style d'une page et créer des applications web interactives.

Exemple arbre DOM :

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Document</title>
</head>
<body>
<h1>Hello World!</h1>
<p>This is a paragraph element</p>
</body>
</html>
```



II. Intégration de JavaScript dans le HTML :

Il existe plusieurs façons d'intégrer JavaScript dans un fichier HTML. Voici les méthodes principales :

1. Directement dans une balise **<script>** (intégration interne)

Vous pouvez écrire du code JavaScript directement dans une balise <script> à l'intérieur du fichier HTML, généralement placée dans la section <head> ou avant la fermeture de <body>.

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple JavaScript</title>
    <script>
        // Exemple de code JavaScript
        console.log("Bonjour depuis JavaScript !");
    </script>
</head>
<body>
    <h1>Intégration interne de JavaScript</h1>
</body>
</html>
```

2. Dans un fichier externe (intégration externe)

Vous pouvez écrire le code JavaScript dans un fichier séparé (par exemple, script.js) et le lier au fichier HTML avec une balise <script> contenant l'attribut src.

Fichier HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple JavaScript Externe</title>
    <script src="script.js"></script>
</head>
<body>
    <h1>Intégration externe de JavaScript</h1>
</body>
</html>
```

Fichier script.js

```
console.log("Bonjour depuis un fichier externe !");
```

3. Avec JavaScript directement dans les attributs HTML (événements)

Vous pouvez également écrire du JavaScript directement dans certains attributs HTML, comme onclick, onmouseover, etc. Cependant, cette méthode est moins recommandée, car elle mélange le code HTML et JavaScript.

```
<!DOCTYPE html>
<html>
<head>
    <title>Exemple JavaScript Inline</title>
</head>
<body>
    <h1>Exemple avec un événement JavaScript</h1>
    <button onclick="alert('Bonjour !')">Cliquez-moi</button>
</body>
</html>
```

III. Affichage de données :

JavaScript offre plusieurs méthodes pour afficher du contenu, chacune adaptée à un usage spécifique. Voici les trois principales : alert, document.write, et console.log.

1. alert

La méthode alert affiche un message dans une fenêtre contextuelle (popup) au navigateur. Cette fenêtre est modale, ce qui signifie qu'elle bloque l'interaction avec la page jusqu'à ce que l'utilisateur clique sur "OK".

Exemples :

```
alert("Bienvenue sur mon site !");
// Affiche une fenêtre avec le message "Bienvenue sur mon site !"
```

Points importants :

- Principalement utilisée pour des notifications ou des messages d'information simples.
- Elle peut être intrusive et gênante si elle est utilisée fréquemment.
- Ne permet pas de personnaliser la fenêtre.

2. document.write

La méthode document.write écrit directement dans le document HTML. Elle peut être utilisée pour ajouter du contenu pendant le chargement de la page.

Exemples :

```
document.write("Bonjour, monde !");
// Affiche "Bonjour, monde !" directement dans la page.
```

Points importants :

- Si utilisée après le chargement initial de la page, elle remplace tout le contenu existant de la page.
- Rarement utilisée aujourd'hui, sauf pour des démonstrations ou des scripts très simples.

3. console.log

La méthode console.log envoie un message à la console JavaScript du navigateur (accessible via les outils de développement).

Exemples :

```
console.log("Message de débogage.");
console.log(42); // Affiche le nombre 42
console.log({ nom: "Jean", age: 30 }); // Affiche un objet dans la
console
```

Points importants :

- Idéale pour le débogage ou pour afficher des informations utiles pendant le développement.
- Les données affichées dans la console ne sont pas visibles par les utilisateurs finaux.
- Permet d'afficher des objets, des tableaux et d'autres structures complexes.

IV. Déclaration et lecture des variables :

1. var (ancienne méthode)

- Utilisée avant ES6 (ancienne version de JavaScript).
- Portée fonctionnelle (visible uniquement à l'intérieur de la fonction dans laquelle elle est déclarée) ou globale si définie en dehors d'une fonction.
- Peut être redéclarée et modifiée.

```
var nom = "Alice";
console.log(nom); // Alice
var nom = "Bob"; // Redéclaration possible
console.log(nom); // Bob
```

⚠ **Inconvénients** :

- Portée difficile à gérer dans des projets complexes.
- Permet des comportements imprévisibles, comme la redéclaration.

2. let (méthode moderne et recommandée)

- Introduite avec ES6.
- Portée limitée au **bloc** dans lequel elle est définie (par exemple, à l'intérieur d'une boucle ou d'une condition).
- Ne peut pas être redéclarée dans la même portée.

```
let age = 25;
console.log(age); // 25
// Redéclaration dans la même portée provoque une erreur
// let age = 30; // Erreur
age = 30; // Modification autorisée
console.log(age); // 30
```

3. const (pour les constantes)

- Introduite avec ES6.
- Utilisée pour déclarer des constantes (valeurs qui ne changent pas).
- Portée limitée au **bloc**.
- Doit être initialisée au moment de la déclaration et ne peut pas être modifiée ensuite.

```
const PI = 3.14;
console.log(PI); // 3.14
// PI = 3.14159; // Erreur : une constante ne peut pas être modifiée
```

4. Lecture des variables avec prompt

La méthode prompt permet d'afficher une boîte de dialogue qui demande à l'utilisateur de saisir une valeur. Cette méthode est utilisée pour lire des entrées utilisateur directement depuis le navigateur.

Syntaxe de prompt

```
let variable = prompt(message, valeurParDefaut);
```

- **message** : Le message affiché dans la boîte de dialogue pour demander une saisie à l'utilisateur.
- **valeurParDefaut** *(facultatif)* : Une valeur initiale affichée dans le champ de saisie. Si elle n'est pas définie, le champ sera vide par défaut.

V. typage dans JavaScript

1. Number (Nombres)

En JavaScript, les nombres sont représentés par le type number, qu'il s'agisse d'entiers ou de nombres à virgule flottante.

Exemples de déclaration :

```
let entier = 42;          // Entier
let decimal = 3.14;       // Nombre décimal
let negatif = -10;        // Négatif
let nan = NaN;            // "Not a Number"
```

Opérations sur les nombres :

a. **Opérations arithmétiques de base :**

```
let a = 10
let b = 4;
console.log(a + b); // Addition : 14
console.log(a - b); // Soustraction : 6
console.log(a * b); // Multiplication : 40
console.log(a / b); // Division : 2.5
console.log(a % b); // Modulo : 2
```

b. **Mathématiques avancées avec Math :**

```
console.log(Math.abs(-8));  // Valeur absolu : 8
console.log(Math.sqrt(16));  // Racine carrée : 4
console.log(Math.round(4.6)); // Arrondi : 5
console.log(Math.trunc(4.6)); // partie entière : 4
console.log(Math.random());  // Nombre aléatoire entre 0 et 1
```

c. **Conversion entre types :**

```
let str = "42";
let num = Number(str);  // Convertir en nombre
console.log(num); // 42
let str2=  "12.5" ;
console.log(parseInt(str2)) //convertir vers un entier : 12
console.log(parseFloat(str2)) //convertir vers un réel : 12.5
```

2. String (Chaînes de caractères)

Les chaînes de caractères sont des séquences de caractères, délimitées par des guillemets simples ('), doubles (") ou des backticks (` `). Elles sont immuables : toute modification génère une nouvelle chaîne.

Exemple de déclaration :

```
let chaine = "Bonjour, JavaScript!";
```

Propriétés et méthodes des chaînes de caractères :

a. length

Renvoie le nombre de caractères dans une chaîne.

```
let texte = "JavaScript";
console.log(texte.length); // 10
```

b. indexOf

Renvoie l'index de la première occurrence d'une sous-chaîne dans une chaîne. Si la sous-chaîne n'est pas trouvée, retourne -1.

```
let phrase = "Bonjour, JavaScript est génial!";
console.log(phrase.indexOf("JavaScript")); // 9
console.log(phrase.indexOf("Python"));    // -1
```

c. lastIndexOf

Renvoie l'index de la dernière occurrence d'une sous-chaîne dans une chaîne. Si la sous-chaîne n'est pas trouvée, retourne -1.

```
let texte = "JavaScript est génial. JavaScript est puissant.";
console.log(texte.lastIndexOf("JavaScript")); // 27
```

d. substring

Extrait une portion de la chaîne entre deux indices spécifiés (le deuxième indice est exclusif).

```
let texte = "Bonjour, JavaScript!";
console.log(texte.substring(9, 19)); // "JavaScript"
console.log(texte.substring(9));     // "JavaScript!"
```

e. replace

Remplace la première occurrence d'une sous-chaîne par une autre.

```
let texte = "JavaScript est génial. JavaScript est puissant.";
console.log(texte.replace("JavaScript", "Python"));// "Python est
génial. JavaScript est puissant."
```

f. toUpperCase

Convertit tous les caractères d'une chaîne en majuscules.

```
let texte = "bonjour";
console.log(texte.toUpperCase()); // "BONJOUR"
```

g. toLowerCase

Convertit tous les caractères d'une chaîne en minuscules.

```
let texte = "BONJOUR";
console.log(texte.toLowerCase()); // "bonjour"
```

h. trim

Supprime les espaces inutiles au début et à la fin d'une chaîne.

```
let texte = "   Bonjour, JavaScript!   ";
console.log(texte.trim()); // "Bonjour, JavaScript!"
```

i. fromCharCode

Permet de convertir des codes Unicode en caractères.

```
console.log(String.fromCharCode(65, 66, 67)); // "ABC"
```

j. String

String peut être utilisé pour convertir d'autres types en chaînes de caractères.

```
let num = 42;
console.log(String(num)); // "42"
```

k. isNaN

Bien que techniquement associée aux nombres, isNaN est souvent utilisée pour vérifier si une chaîne peut être convertie en un nombre valide.

```
let chaine1 = "123";
let chaine2 = "abc";
console.log(isNaN(chaine1)); // false (convertible en nombre)
console.log(isNaN(chaine2)); // true (non convertible)
```

4. Les booléens :

Un booléen est un type de donnée en JavaScript qui ne peut avoir que deux valeurs :

- true (vrai)
- false (faux)

Les booléens sont souvent le résultat d'une opération de comparaison ou logique.

Exemples simples :

```
let estMajeur = true;
let estEtudiant = false;
console.log(estMajeur); // Affiche true
console.log(estEtudiant); // Affiche false
```

a. Opérateurs de comparaison

Les opérateurs de comparaison permettent de comparer des valeurs. Ils renvoient toujours une valeur booléenne (true ou false).

Opérateur	Description	Exemple	Résultat
==	Égalité (valeur seulement)	5 == "5"	true
!=	Différent (valeur seulement)	5 != "5"	false
>	Supérieur	10 > 5	true
<	Inférieur	5 < 10	true
>=	Supérieur ou égal	10 >= 10	true
<=	Inférieur ou égal	5 <= 10	true

Liste des opérateurs logiques :

Opérateur	Nom	Description
&&	ET logique	Renvoie true si toutes les conditions sont vraies
||	OU logique	Renvoie true si au moins une condition est vraie
!	NON logique (négation)	Renverse la valeur : true devient false et vice-versa

5. Date (Objets Date)

En JavaScript, les dates et heures sont manipulées à l'aide de l'objet Date. Cet objet permet de travailler avec des dates spécifiques et d'extraire ou formater leurs composants.

Pour crée une instance représentant la date et l'heure actuelles :

```
let dateActuelle = new Date();
console.log(dateActuelle); // Exemple : "2025-01-03T12:34:56.789Z"
```

new Date(string) Permet de créer une date à partir d'une chaîne de caractères au format valide. Exemple : YYYY-MM-DD, MM/DD/YYYY, ou une autre représentation standardisée.

```
let dateString = new Date("2025-01-01");
console.log(dateString); // "2025-01-01T00:00:00.000Z"
```

Récupération des composants d'une date :

a. **getDate()**
   Renvoie le jour du mois (1 à 31).

```
let date = new Date("2025-01-03");
console.log(date.getDate()); // 3
```

b. **getMonth()**
   Renvoie le mois de la date (0 pour janvier, 11 pour décembre).

```
let date = new Date("2025-01-03");
console.log(date.getMonth()); // 0 (janvier)
```

c. **getFullYear()**
   Renvoie l'année complète (au format YYYY).

```
let date = new Date("2025-01-03");
console.log(date.getFullYear()); // 2025
```

d. **toString()**
   Convertit une date en une chaîne lisible pour les humains.

```
let date = new Date("2025-01-03"); console.log(date.toString());
// Exemple : "Fri Jan 03 2025 00:00:00 GMT+0000 (Coordinated Universal
Time)"
```

6. Array (Tableaux)

En JavaScript, un tableau (ou Array) est une collection ordonnée de valeurs. Ces valeurs peuvent être de n'importe quel type (numériques, chaînes de caractères, objets, autres tableaux, etc.).

Déclaration et assignation de tableaux :

a. Créer un tableau vide :

```
let tableauVide = [];
```

b. Créer un tableau avec des valeurs initiales :

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

c. accéder à un élément du tableau :

```
let fruits = ["pomme", "banane", "cerise"];
console.log(fruits[2]) // cerise
```

VI. Modification du DOM avec JavaScript

JavaScript permet de manipuler le contenu HTML ou le texte d'une page grâce à l'utilisation de méthodes et propriétés spécifiques.

1. getElementById

La méthode **getElementById** permet de sélectionner un élément HTML grâce à son attribut id.

Syntaxe :

```
document.getElementById(id);
```

⇨ **id** : L'identifiant unique de l'élément.

Exemple d'utilisation :

```
<p id="message">Texte d'origine</p>
<script>
    let element = document.getElementById("message");
    element.innerText = "Texte modifié par JavaScript";
</script>
```

2. innerText

La propriété **innerText** permet de modifier ou de lire uniquement le texte visible d'un élément HTML. Le texte est interprété tel quel (sans balises HTML).

Syntaxe :

```
element.innerText = "Nouveau texte";
```

Exemple d'utilisation :

```
<p id="exemple">Bonjour !</p>
<script>
    let paragraphe = document.getElementById("exemple");
    paragraphe.innerText = "Bonjour tout le monde !";
</script>
```

3. innerHTML

La propriété **innerHTML** permet de modifier ou de lire le contenu HTML d'un élément, y compris les balises HTML. Contrairement à innerText, innerHTML interprète les balises incluses dans la chaîne.

Syntaxe :

```
element.innerHTML = "<b>Nouveau contenu avec balises</b>";
```

Exemple d'utilisation :

```
<div id="contenu">Texte initial</div>
<script>
    let div = document.getElementById("contenu");
    div.innerHTML = "<strong>Texte en gras</strong> et une <a
href='#'>lien</a>";
</script>
```

4. getElementsByName

La méthode **getElementsByName** permet de sélectionner tous les éléments ayant le même attribut name. Elle retourne une **collection d'éléments** (semblable à un tableau).

Syntaxe :

```
document.getElementsByName(name);
```

⇨ **name** : Le nom des éléments HTML à sélectionner.

Exemple d'utilisation :

```
<input type="radio" name="choix" value="Option 1">
<input type="radio" name="choix" value="Option 2">
<button onclick="afficherSelection()">Afficher</button>
<script>
    function afficherSelection() {
        let radios = document.getElementsByName("choix");
        for (let i = 0; i < radios.length; i++) {
            if (radios[i].checked) {
                alert("Vous avez sélectionné : " + radios[i].value);
            }
        }
    }
</script>
```

VII. Les structures de contrôle conditionnelles :

Les structures conditionnelles permettent d'exécuter des instructions spécifiques en fonction de la validité d'une ou plusieurs conditions.

1. La structure if...else

La condition est évaluée, et si elle est vraie, le bloc d'instructions associé est exécuté. Sinon, le bloc suivant (optionnel) est exécuté.

Exemple :

```
let age = 18;
if (age >= 18) {
    console.log("Vous êtes majeur.");
} else {
    console.log("Vous êtes mineur.");
}
```

2. La structure if...else if...else

Elle permet de vérifier plusieurs conditions, en testant chaque condition jusqu'à ce qu'une soit vraie.

Syntaxe :

```
if (condition1) {
    // Code à exécuter si condition1 est vraie
} else if (condition2) {
    // Code à exécuter si condition2 est vraie
} else {
    // Code à exécuter si aucune des conditions n'est vraie
}
```

Exemple :

```
let note = 85;
if (note >= 90) {
    console.log("Excellent");
} else if (note >= 75) {
    console.log("Bien");
} else if (note >= 50) {
    console.log("Passable");
} else {
    console.log("Échec");
}
```

3. La structure switch

Description :

La structure **switch** compare une variable ou une expression à plusieurs valeurs possibles (case) et exécute le bloc de code correspondant.

Syntaxe :

```
switch (expression) {
    case valeur1:
        // Code à exécuter si expression === valeur1
        break;
    case valeur2:
        // Code à exécuter si expression === valeur2
        break;
    default:
        // Code à exécuter si aucune correspondance
}
```

Exemple :

```
let jour = 3;
switch (jour) {
    case 1:
        console.log("Lundi");
        break;
    case 2:
        console.log("Mardi");
        break;
    case 3:
        console.log("Mercredi");
        break;
    default:
        console.log("Jour inconnu");
}
// Résultat : "Mercredi"
```

⚠ *Attention : Sans le mot-clé break, l'exécution continue vers les blocs suivants, même si une condition est satisfaite.*

VIII. Les Boucles en Javascript

Les boucles permettent de répéter des blocs de code plusieurs fois, soit un nombre défini de fois, soit tant qu'une condition reste vraie.

1. La boucle for

La boucle **for** est utilisée lorsque le nombre d'itérations est connu à l'avance. Elle est composée de trois parties :

1. Initialisation.
2. Condition.
3. Incrémentation/Décrémentation.

Syntaxe :

```
for (initialisation; condition; miseÀJour) {
    // Code à exécuter
}
```

Exemple :

```
for (let i = 0; i < 3; i++) {
    console.log("Itération : " + i);
}
// Résultat :
// Itération : 0
// Itération : 1
// Itération : 2
```

2. La boucle while

La boucle **while** exécute un bloc de code tant qu'une condition est vraie. Elle est utilisée lorsque le nombre d'itérations n'est pas connu à l'avance.

Syntaxe :

```
while (condition) {
    // Code à exécuter
}
```

Exemple :

```
let compteur = 0;
while (compteur < 3) {
    console.log("Compteur : " + compteur);
    compteur++;
}
// Résultat :
// Compteur : 0
// Compteur : 1
// Compteur : 2
```

3. La boucle do...while

La boucle **do...while** est similaire à while, sauf qu'elle s'exécute **au moins une fois**, même si la condition est fausse, car la condition est vérifiée après le bloc de code.

Syntaxe :

```
do {
    // Code à exécuter
} while (condition);
```

Exemple :

```
let compteur = 5;
do {
    console.log("Compteur : " + compteur);
    compteur++;
} while (compteur < 3);
// Résultat :
// Compteur : 5 (la boucle s'exécute au moins une fois)
```

IX. Modification du contenu, des attributs ou des styles des éléments HTML

JavaScript permet de manipuler les éléments HTML dynamiquement, notamment pour modifier leur contenu, leurs attributs, leurs styles, ou pour interagir avec des éléments multimédias.

1. Modification des attributs

Syntaxe générale :

```
element.attribut = valeur;
```

Exemples :

➪ **Changer la source d'une image :**

```
let image = document.getElementById("imageExemple");
image.src = "nouvelleImage.jpg";
```

➪ **Modifier l'attribut href d'un lien :**

```
let lien = document.getElementById("monLien");
lien.href = "https://www.exemple.com";
```

➪ **Changer le texte alternatif d'une image (alt) :**

```
image.alt = "Nouvelle description de l'image";
```

2. Modification des styles

Syntaxe générale :

```
element.style.propriétéCSS = valeur;
```

⚠ Les noms de propriétés CSS sont transformés en camelCase dans JavaScript (par exemple, background-color devient backgroundColor).

Exemples :

➪ **Changer la couleur du texte :**

```
let titre = document.getElementById("titreExemple");
titre.style.color = "blue";
```

➪ **Modifier la taille de la police :**

```
titre.style.fontSize = "24px";
```

➪ **Changer l'arrière-plan d'un élément :**

```
titre.style.backgroundColor = "yellow";
```

➪ **Ajouter un bord :**

```
titre.style.border = "2px solid black";
```

3. Interagir avec des éléments multimédias

Les éléments multimédias tels que les vidéos ou les audios peuvent être contrôlés avec les méthodes **play** et **pause**.

Méthodes :

- **play() :** Lance la lecture.
- **pause() :** Met la lecture en pause.

Exemple : Lecture et pause d'une vidéo :

```
let video = document.getElementById("maVideo");
// Démarrer la lecture
video.play();
// Mettre la vidéo en pause
video.pause();
```