

JavaScript

▼ Introduction

JavaScript a été créé en **1995**.

Avec JS :

- ✓ Les sites peuvent **réagir** aux actions des utilisateurs.
- ✓ Ils peuvent **afficher des messages** ou **animer** des éléments.
- ✓ Ils peuvent **mémoriser des informations**, rendant l'expérience utilisateur plus fluide.

Aujourd'hui, JavaScript est partout : **sites dynamiques, applications mobiles, jeux vidéo, serveurs, et bien plus encore !** 🚀

1. Front-End (Côté client)

Le **front-end** concerne tout ce que l'utilisateur voit et avec quoi il interagit directement. Avec JavaScript, il est possible de rendre un site **réactif**. 🎮

2. Back-End (Côté serveur) : Le cerveau caché des applications 🧠

Le **back-end**, lui, gère tout ce que l'on ne voit pas : les bases de données, les informations des utilisateurs. Grâce à des outils comme **Node.js**, JavaScript ne se contente plus de vivre côté utilisateur. Il peut désormais exécuter des actions directement sur **les serveurs**. Cela signifie qu'on peut utiliser JavaScript pour gérer tout un site ou une application.

Comprendre ECMAScript (ES5, ES6, et au-delà) 📖🔍

ECMAScript, souvent abrégé en **ES**, est la norme qui définit le fonctionnement du JavaScript. Cela garantit que JavaScript fonctionne de la même manière, quel que soit le navigateur utilisé (Chrome, Firefox, Safari, etc.).

Sans cette norme, chaque navigateur pourrait interpréter JavaScript de manière différente, ce qui entraînerait des bugs et de la confusion pour les développeurs.

Pourquoi avons-nous des mises à jour ?

Comme les applications et les logiciels, JavaScript reçoit des mises à jour pour ajouter de nouvelles fonctionnalités, corriger des problèmes et le rendre plus puissant et plus facile à utiliser. Ces mises à jour sont publiées sous forme de nouvelles versions d'**ECMAScript**.

Voici les deux versions les plus importantes à connaître :

1. ES5 (ECMAScript 5)

- **Date de sortie** : 2009
 - **Pourquoi c'est important** : ES5 a apporté d'**énormes améliorations** qui ont rendu JavaScript plus fiable et plus facile à utiliser. Il a introduit de nouvelles méthodes pour manipuler les tableaux et les chaînes de caractères, ainsi que des outils pour le débogage et la gestion des erreurs.
 - **Principales caractéristiques** :
 - **Nouvelles méthodes pour les tableaux** : Comme **forEach()**, **map()**, et **filter()**, facilitant la manipulation des données.
 - **Support JSON** : A rendu le travail avec **JSON** (format de données utilisé par les API) plus simple.
-

2. ES6 (ECMAScript 6, aussi appelé ES2015)

- **Date de sortie** : 2015
- **Pourquoi c'est important** : **ES6 a révolutionné** JavaScript. Il a introduit de nombreuses fonctionnalités puissantes qui ont rendu le langage plus moderne et plus agréable à utiliser. C'est maintenant le **standard** que la plupart des développeurs utilisent.
- **Principales caractéristiques** :
 - **let et const** : De nouvelles façons de déclarer des variables, plus sûres et plus claires que **var**.
 - **Fonctions fléchées** : Une manière concise d'écrire des fonctions, rendant le code plus lisible.
 - **Template literals** : Permet de créer des chaînes de caractères avec des variables incluses en utilisant des **backticks** (```) au lieu de guillemets.

- **Paramètres par défaut** : Permet d'attribuer des valeurs par défaut aux paramètres des fonctions.
- **Destructuration** : Permet d'extraire facilement des valeurs d'un tableau ou d'un objet et de les affecter à des variables.
- **Promesses** : Une manière moderne de gérer les tâches asynchrones, comme la récupération de données depuis un serveur.


Configurer l'Environnement de Développement JavaScript



Étape 1 : Choisir et Configurer un IDE (VS Code, Sublime, etc.)

IDE : Environnement de Développement Intégré)

Voici deux IDE populaires pour JavaScript :

- **VS Code** : Gratuit, extrêmement populaire et avec de nombreuses fonctionnalités utiles, comme l'auto-complétion du code, des extensions, et un terminal intégré. 
- **Sublime Text** : Léger, facile à utiliser, mais avec moins de fonctionnalités comparé à VS Code. Parfait pour les projets simples. ✨

Étape 2 : Configurer un Environnement JavaScript (Installation de Node.js)

Pour exécuter du JavaScript en dehors du navigateur, vous aurez besoin de **Node.js**. Il vous permet d'écrire et de tester du JavaScript directement sur votre ordinateur, sans avoir besoin d'un navigateur.

Téléchargez la dernière version stable.

Vérification : Ouvrez votre terminal et tapez la commande suivante. Cela vous montrera la version de **Node.js** que vous avez installée. Si vous obtenez un numéro de version, c'est que vous avez tout installé correctement ! Cela installera aussi npm qui est un gestionnaire de paquets permettant d'utiliser des librairies.

```
node -v
```

Étape 3 : Exécuter JavaScript dans la Console du Navigateur

La plupart des navigateurs (comme **Chrome**) disposent d'un outil caché appelé **Console**, qui permet d'exécuter du code JavaScript directement dans le navigateur. Voici comment l'ouvrir :

- **Appuyez sur F12** sur votre clavier (ou faites un clic droit sur la page, sélectionnez "Inspecter", puis allez dans l'onglet "Console").

▼ Les fondamentaux

▼ Les variables

Les Variables en JavaScript (let, const)

Les **variables** permettent de stocker des informations. Ils conservent des informations que vous utiliserez plus tard.

En JavaScript, il existe deux principales façons de déclarer des variables :

1. let

- **Usage** : C'est la méthode recommandée aujourd'hui. Elle est **flexible** et **moderne**, parfaite pour les variables dont la valeur **change** au cours du temps.
- **Particularité** : La variable déclarée avec `let` a une **portée limitée** au bloc dans lequel elle se trouve, ce qui la rend plus sûre à utiliser que `var`.

Exemple :

```
let name= "John";
```

2. const

- **Usage** : Utilisé lorsque vous ne voulez pas que la valeur d'une variable change. C'est parfait pour des constantes qui ne doivent pas être modifiée.
- **Particularité** : Une fois qu'une variable est déclarée avec `const`, vous **ne pouvez pas changer** sa valeur. Cela garantit la stabilité et la

sécurité de vos données.

Exemple :

```
const restaurantName = "The Tasty Spoon"
```

▼ Les types de données

Les Types de Données en JavaScript (Types Primitifs et Référencés) 🧠

En JavaScript, les **types de données** nous indiquent quel type d'information est stocké dans une variable. On les divise en deux catégories principales :

1. Types Primitifs (Types de Base) ⚙️

Les **types primitifs** sont des données simples.

Les types primitifs sont :

- **String** : Texte (chaînes de caractères).
- **Number** : Nombres (entiers, décimaux).
- **Boolean** : Valeurs de vérité (true ou false).
- **Null** : Une valeur vide, assignée intentionnellement.
- **Undefined** : Une variable qui n'a pas encore de valeur.

2. Types Référencés 📁

Les **types référencés** sont plus complexes. Ils **stockent plusieurs valeurs** ou même **d'autres objets** et peuvent être modifiés.

Les types référencés incluent :

- **Objet (Object)** : Contient plusieurs valeurs sous forme de **paires clé-valeur** (comme un carnet d'adresses).
- **Tableau (Array)** : Une **liste ordonnée** d'éléments.

Exemple de Code :

Types Primitifs :

```
let petName = "Fluffy"; // String, utilisé pour du texte
let age = 3; // Number, utilisé pour des nombres
let isHappy = true; // Boolean, valeur vrai ou faux
```

Types Référencés :

```
let shoppingList = ["milk", "eggs", "bread"]; // Array, liste d'articles
let wizard = {name: "Merlin", age: 150, isFriendly: true};
// Object, détails sur le magicien
```

▼ Les opérateurs

Les Opérateurs en JavaScript

Les **opérateurs** sont des symboles qui indiquent à JavaScript quelle action effectuer sur des données. Ils permettent de travailler avec des **nombres**, des **textes**, des **variables**, et plus encore. Voici un aperçu des types d'opérateurs les plus courants.

1. Opérateurs Arithmétiques : Faire des Calculs en Code

Les **opérateurs arithmétiques** sont utilisés pour effectuer des opérations mathématiques simples, comme l'addition ou la soustraction.

Exemple de Code :

```
let a = 1;
a = a * 5; // les 4 principales opérations (+ - * /)
a *= 5; // notation raccourcie
a = a + 1;
a++; // notation raccourcie
console.log(a);
```

2. Opérateurs de Comparaison : Vérifier les Valeurs

Les **opérateurs de comparaison** comparent deux valeurs et retournent **true** (vrai) ou **false** (faux) en fonction du résultat.



Le triple égal vérifie la valeur et le type. Il est à privilégier

Exemple de Code :

```
let b = "4";
let c = 4;
let d = 5;
console.log(b == c); // retourne true
console.log(d === c); // retourne false à privilégier

console.log(d < c); // retourne false
console.log(d >= c); // retourne true
```

3. Opérateurs Logiques : Combiner des Conditions

Les **opérateurs logiques** vous permettent de vérifier plusieurs conditions en même temps.

Exemple de Code :

```
// opérateurs de comparaison ET
console.log(true && true); // retourne true
console.log(true && false); // retourne false
console.log(false && true); // retourne false
console.log(false && false); // retourne false

// opérateurs de comparaison OU
console.log(true || true); // retourne true
console.log(true || false); // retourne true
console.log(false || true); // retourne true
console.log(false || false); // retourne false
```

Explication :

- `&&` (ET) : Les deux conditions doivent être **vraies** pour que le résultat soit **vrai**.
- `||` (OU) : Au moins **une** des conditions doit être **vraie** pour que le résultat soit **vrai**.

▼ Les conditions et boucles

1. Les conditions 🧠

Les instructions `if` , `else if` , et `else` 🏔️

Exemple : Planificateur d'habits basé sur la météo

```
let weather = "rainy";

if (weather === "sunny") {
  console.log("Portez des lunettes de soleil et un chapeau !");
} else if (weather === "rainy") {
  console.log("Prenez un parapluie !");
} else {
  console.log("Portez ce qui vous semble juste !");
}
```

Explication de l'exemple :

- `if` : Vérifie la condition, ici, "Est-ce que la météo est ensoleillée ?"
- `else if` : Si la première condition n'est pas vraie, vérifie une autre condition, ici "Est-ce qu'il pleut ?"
- `else` : Si aucune des conditions précédentes n'est vraie, il s'exécute et fournit une action par défaut.

Et aussi :

- Vous pouvez chaîner plusieurs `else if` pour tester plusieurs conditions, mais `else` fournit une valeur par défaut au cas où toutes les autres conditions échoueraient.

2. Les conditions avec **switch**

C'est plus propre et plus efficace que d'utiliser une pile de **else if**.

Exemple : Décider du jour en fonction du numéro

```
const day = 3;

switch (day) {
  case 1:
    console.log("Lundi");
    break;
  case 2:
    console.log("Mardi");
    break;
  case 3:
    console.log("Mercredi");
    break;
  case 4:
    console.log("Jeudi");
    break;
  default:
    console.log("Jour invalide");
}
```

Explication :

- Le **switch** vérifie la valeur de **day**.
- Dès qu'il trouve une correspondance (ici **case 3**), il exécute le bloc de code correspondant (`console.log("Mercredi")`).
- Le **break** est crucial, car il empêche le code de continuer à tester d'autres cas après une correspondance.
- Si aucun cas ne correspond, le bloc **default** s'exécute.

3. Les conditions avec la ternaire

La ternaire évalue une condition et exécute ce qui se trouve après le point d'interrogation si c'est vrai sinon il exécute ce qui se trouve après les deux points. Il est possible d'imbriquer des ternaires.

Exemple : Décider du droit de parier

```
let age = 17;
let result;
let droit = age >= 18
  ? result = "Vous pouvez parier"
  : result = "Vous ne pouvez pas parier"

console.log(result)
```

4. Répéter des Actions avec des Boucles (Loops)

Les **boucles** vous permettent de répéter des actions jusqu'à ce que certaines conditions soient remplies.

a. La boucle **for** - Répéter un nombre défini d'itérations

Exemple

```
let i = 1 // initialise la boucle
i <= 9 // cela va boucler tant que i n'est pas égal à 9
i = i + 1 // on incrémente de 1 la valeur à chaque tour

for (let i = 1; i <= 9; i = i + 1) {
  console.log(i);
}

// affichage de la table de multiplication de 5
for (let i = 1; i <= 9; i = i + 1) {
  // console.log(i * 5);
  console.log(i + " * 5 = " + i * 5);
}
```

Vous pouvez placer des conditions à l'intérieur d'une boucle et ainsi échapper certains résultats

Exemple avancée :

```
for (let i = 9; i >= 1; i--) {  
  if (i !== 5) {  
    console.log(i);  
  }  
}
```

Ce code affiche les chiffres de 9 à 1 sauf le 5.

Il utilise une boucle inversée, une notation raccourcie ainsi qu'une condition.