



JS

Formation Javascript

TALIS 2024

Programme

- Jour 1: Introduction au Javascript et bases essentielles, variables, condition, fonction
- Jour 2: Tableaux, objets, dates
- Jour 3: Manipulation du DOM et événements
- Jour 4: Manipulation du DOM et événements
- Jour 5: Introduction à l'asynchrone
- Jour 6: Introduction à Vue.js et Typescript



Introduction au Javascript

- **Javascript:** Langage de programmation pour le web
- **Rôle:** Manipuler le DOM, interagir avec le serveur, ajouter les fonctionnalités interactives aux pages web.
- **Différence avec HTML/CSS:** HTML structure la page, CSS la stylise, Javascript rend la page interactive
- **Langage polyvalent:** développement mobile (React Native, Ionic), développement backend (Node.js, Express.js), développement jeux vidéo, intelligence artificielle, machine learning etc.

Utiliser JavaScript au sein d'une page web

Javascript inline (en ligne)

```
<button onclick="alert('Bonjour !')">Cliquez-moi</button>
```

Javascript dans un `<script>` interne

```
<!DOCTYPE html>
<html>
<head>
  <title>Exemple</title>
</head>
<body>
  <button id="monBouton">Cliquez-moi</button>

  <script>
    document.getElementById('monBouton').onclick = function() {
      alert('Bonjour depuis le script interne !');
    }
  </script>
</body>
</html>
```

Utiliser JavaScript au sein d'une page web

Javascript externe

```
<!DOCTYPE html>
<html>
<head>
  <title>Exemple avec script externe</title>
  <script src="script.js"></script>
</head>
<body>
  <button id="monBouton">Cliquez-moi</button>
</body>
</html>
```

```
document.getElementById('monBouton').onclick = function() {
  alert('Bonjour depuis le fichier externe !');
}
```

Méthodes utiles

- **document.write()** : écrit une chaîne de texte dans le document
- **alert()**: indique au navigateur d'afficher une boîte de dialogue avec un message
- **console.log()**: permet d'afficher un message dans la console (le plus souvent utilisé)

```
1 document.write('Hello World')
2 alert('Hello World')
3 console.log('Hello World')
```

Commentaires dans le code

- Commentaire sur une ligne (syntax : //)

```
// let y = 10; // Cette ligne est désactivée
```

- Commentaire sur plusieurs lignes (syntax : /* */)

```
/*  
Ceci est un commentaire  
sur plusieurs lignes  
*/  
let z = 10;
```

Utilisations:

- Améliorer la visibilité
- Facilite la compréhension pour d'autres développeurs
- Désactiver le code

Les variables c'est quoi?

Une variable est un espace en mémoire réservé pour stocker des informations qui peuvent être référencées et manipulées dans le code.

Comment on déclare une variable en Javascript?

- Déclaration avec `var` : la portée d'une variable peut être globale ou locale (si déclaré dans une fonction)
- Déclaration avec `let`: la portée d'une variable déclarée est limitée au bloc dans lequel elle a été définie. Elle peut être réassignée mais pas redéclaré dans le même bloc
- Déclaration avec `const`: La variable ne peut pas être réassignée. La portée est également limitée au bloc où elle a été définie

L'utilisation de `var` n'est pas recommandée dans le code moderne.

Les types de données primitifs

- **Nombre (Number):** ce type représente des nombres, qu'ils soient entiers ou à virgule flottante (décimaux). JavaScript utilise un seul type pour tous les nombres.
- **Chaîne de caractère (String):** des mots, phrases ou textes
- **Booléen (Boolean):** ce type ne prend que deux valeurs `true` et `false`
- **Undefined:** il indique l'absence de valeur
- **Null:** ce type présente une valeur volontairement vide ou nulle

Exemples:

```
let age = 30;           // Number
let name = "Alice";     // String
let isStudent = true;   // Boolean
let address = null;     // Null
let phoneNumber;        // Undefined
```

(Il existe également des types complexes, on les verra plus tard dans le cours.)

Les opérateurs arithmétiques

Ces opérateurs effectuent des calculs mathématiques sur des nombre

Opérateur	Description	Exemple
<code>+</code>	Addition	<code>5 + 3</code>
<code>-</code>	Soustraction	<code>5 - 3</code>
<code>*</code>	Multiplication	<code>5 * 3</code>
<code>/</code>	Division	<code>5 / 3</code>
<code>%</code>	Modulo (reste de la division)	<code>5 % 3</code>
<code>**</code>	Exponentiation	<code>2 ** 3</code> (2 à la puissance 3)

Les opérateurs de comparaison

Ces opérateurs comparent des valeurs et retournent un booléen

Opérateur	Description	Exemple
<code>==</code>	Égalité (valeurs égales)	<code>5 == "5"</code>
<code>===</code>	Égalité stricte (valeurs et types égaux)	<code>5 === "5"</code>
<code>!=</code>	Inégalité (valeurs différentes)	<code>5 != "5"</code>
<code>!==</code>	Inégalité stricte (valeurs ou types différents)	<code>5 !== "5"</code>
<code>></code>	Plus grand que	<code>5 > 3</code>
<code><</code>	Moins que	<code>5 < 3</code>
<code>>=</code>	Plus grand ou égal à	<code>5 >= 3</code>
<code><=</code>	Moins ou égal à	<code>5 <= 3</code>

Les opérateurs de logiques

Ces opérateurs comparent des valeurs et retournent un booléen

```
let age = 20;
let hasPermission = true;
let isStudent = false;

// Utilisation de l'opérateur && (ET logique)
let accessAllowed = age >= 18 && hasPermission;
console.log("Accès autorisé :", accessAllowed);
// Accès autorisé : true

// Utilisation de l'opérateur || (OU logique)
let accessLimited = age < 18 || !hasPermission;
console.log("Accès limité :", accessLimited);
// Accès limité : false

// Utilisation de l'opérateur ! (NON logique)
let notStudent = !isStudent;
console.log("Pas un étudiant :", notStudent);
// Pas un étudiant : true

// Combinaison des opérateurs
let combinedAccess = (age < 18 || hasPermission) && !isStudent;
console.log("Accès limité pour les mineurs sans statut étudiant :", combinedAccess);
// Accès limité pour les mineurs sans statut étudiant : true
```

Concaténation de chaînes de caractère

- Utilisation de l'opérateur +

```
let str1 = "Bonjour";  
let str2 = "tout le monde";  
let message = str1 + " " + str2; // Ajoute un espace entre les deux chaînes  
console.log(message); // Affiche "Bonjour tout le monde"
```

- Utilisation de l'opérateur +=

```
let greeting = "Bonjour";  
greeting += " tout le monde"; // Concatène et réassigne à greeting  
console.log(greeting); // Affiche "Bonjour tout le monde"
```

- Utilisation des littéraux de gabarits (template literals)

```
let firstName = "Alice";  
let lastName = "Dupont";  
let fullName = `${firstName} ${lastName}`; // Utilise les backticks et `${}` pour l'interp  
console.log(fullName); // Affiche "Alice Dupont"
```

Méthodes utiles pour manipuler des chaînes de caractère

- **length()**: renvoie le nombre de caractères d'une chaîne
- **toUpperCase()**: convertit tous les caractères d'une chaîne en majuscules
- **toLowerCase()**: convertit tous les caractères d'un chaîne en minuscules

```
let message = "Hello World";  
// Renvoie la longueur de la chaîne (nombre de caractères)  
let messageLength = message.length;  
  
console.log(messageLength); // Affiche : 11
```

```
let greeting = "good morning";  
// Convertit la chaîne en majuscules  
let upperGreeting = greeting.toUpperCase();  
  
console.log(upperGreeting); // Affiche : "GOOD MORNING"
```

```
let shout = "HELLO";  
// Convertit la chaîne en minuscules  
let lowerShout = shout.toLowerCase();  
  
console.log(lowerShout); // Affiche : "hello"
```

Les opérations mathématiques

En Javascript l'objet Math offre plusieurs fonctions utiles pour effectuer de opérations mathématiques. Voici quelques exemples:

```
// Renvoie la valeur absolue de -5
console.log(Math.abs(-5)); // 5

// Renvoie le plus petit entier supérieur ou égal à 4.2
console.log(Math.ceil(4.2)); // 5

// Renvoie le plus grand entier inférieur ou égal à 4.8
console.log(Math.floor(4.8)); // 4

// Renvoie la valeur de 4.5 arrondie au nombre entier le plus proche
console.log(Math.round(4.5)); // 5

// Renvoie le plus grand des nombres fournis : 1, 2 et 3
console.log(Math.max(1, 2, 3)); // 3

// Renvoie le plus petit des nombres fournis : 1, 2 et 3
console.log(Math.min(1, 2, 3)); // 1

// Renvoie un nombre flottant aléatoire compris entre 0 (inclus) et 1 (exclus)
console.log(Math.random()); // Par exemple, 0.123456

// Renvoie la valeur de 2 élevée à la puissance 3 (2^3)
console.log(Math.pow(2, 3)); // 8
```

Structures conditionnelles

En JavaScript, les structures conditionnelles permettent d'exécuter différentes portions de code selon certaines conditions

- **If, if ... else** : La structure conditionnelle de base qui exécute un bloc de code si une condition est vraie. Avec 'else' on peut ajouter une alternative si la condition est fausse
- **Else ... if**: permet d'ajouter des conditions supplémentaires

```
if (condition) {  
    // Code à exécuter si la condition est vraie  
} else {  
    // Code à exécuter si la condition est fausse  
}
```

```
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 condition n'est vraie  
}
```


Structures conditionnelles

- **Switch** : Pas très utilisé mais ça peut être utile pour tester une valeur sur plusieurs cas possible
- **Opérateur ternaire**: une forme condensée de l'instruction if..else

```
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 aucun cas n'est satisfait  
}
```

```
let resultat = (condition) ? valeurSiVrai : valeurSiFaux;
```

Les boucles

- **for**: la boucle for peut être utilisée lorsque le nombre de l'itération est connu à l'avance

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

```
for (let i = 0; i < 5; i++) {  
    console.log(i);  
}  
// Résultat: 0, 1, 2, 3, 4
```

Les boucles

- **while:** la boucle `while` exécute un bloc de code tant qu'une condition est vraie.

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

```
let i = 0;  
while (i < 5) {  
    console.log(i);  
    i++;  
}  
// Résultat: 0, 1, 2, 3, 4
```

- **do...while:** la boucle `do...while` est similaire à `while`, sauf qu'elle exécute le bloc de code au moins une fois avant de vérifier la condition

```
let i = 0;  
do {  
    console.log(i);  
    i++;  
} while (i < 5);  
// Résultat: 0, 1, 2, 3, 4
```

Les fonctions

En JavaScript, une fonction est un bloc de code réutilisable qui peut être exécuté lorsqu'il est appelé. Les fonctions permettent d'organiser et de structurer le code, facilitant ainsi sa maintenance et sa réutilisation.

Déclaration d'une fonction:

```
function greet() {  
  console.log("Hello!");  
}
```

Appeler une fonction:

```
greet(); // Affiche "Hello!"
```

Les fonctions

Fonctions avec paramètres:

Les fonctions peuvent accepter des paramètres, qui sont des valeurs passées à la fonction.

```
function greetByName(name) {  
    console.log("Hello, " + name + "!");  
}  
  
greetByName("Alice"); // Hello, Alice!
```

Valeur de retour:

Une fonction peut envoyer une valeur à l'aide du mot-clé return. lorsqu'une fonction atteint return, elle s'arrête immédiatement et renvoi la valeur spécifique

```
function add(a, b) {  
    return a + b;  
}  
  
let result = add(3, 5); // result vaut 8  
console.log(result);
```

Les fonctions

Fonctions anonymes:

Une fonction peut être anonyme, elle n'a pas de nom, elle est souvent affectée à une variable.

```
let greet = function() {  
  console.log("Hello!");  
};  
  
greet(); // Hello!
```

Fonctions fléchée (arrow function):

Une syntaxe plus concise,
introduit avec ES6

```
const greet = () => {  
  console.log("Hello!");  
};  
  
greet(); // Hello!  
  
// Avec des paramètres  
const multiply = (a, b) => a * b;  
console.log(multiply(3, 5)); // 15
```

Les fonctions

Fonctions avec des paramètres par défaut:

```
function greet(name = "friend") {  
    console.log("Hello, " + name + "!");  
}  
  
greet(); // Hello, friend!  
greet("John"); // Hello, John!
```

Fonctions récursives:

Une fonction est récursive lorsqu'elle appelle elle-même

```
function factorial(n) {  
    if (n === 0) {  
        return 1;  
    }  
    return n * factorial(n - 1);  
}  
  
console.log(factorial(5)); // 120
```

Fonctions auto-invoquées:

```
(function() {  
    console.log("Je m'exécute immédiatement !");  
})();
```

Les tableaux (Array)

Les tableaux sont des listes ordonnées de valeurs, indexées par des numéros (les index commencent à 0). Ils peuvent contenir des éléments de n'importe quel type.

Parcourir les éléments itérables d'un tableaux avec la boucle **for...of**:

```
let fruits = ["Pomme", "Banane", "Orange"];  
console.log(fruits[1]); // "Banane"
```

```
for (let element of iterable) {  
  // Code à exécuter  
}
```

```
let fruits = ['apple', 'banana', 'orange'];  
for (let fruit of fruits) {  
  console.log(fruit);  
}  
// Résultat: apple, banana, orange
```


Manipulation des tableaux

- `push()`: ajoute un élément à la fin d'un tableau
- `pop()`: supprime le dernier élément du tableau et il le retourne
- `shift()`: supprime le premier élément du tableau et il le retourne
- `unshift()`: ajoute un ou plusieurs éléments au début du tableau

```
let arr = [1, 2];  
arr.push(3); // arr devient [1, 2, 3]
```

```
let arr = [1, 2, 3];  
arr.pop(); // arr devient [1, 2]
```

```
let arr = [1, 2, 3];  
arr.shift(); // arr devient [2, 3]
```

```
let arr = [2, 3];  
arr.unshift(1); // arr devient [1, 2, 3]
```

Manipulation des tableaux

- `map()`: applique une fonction à chaque élément et renvoie un nouveau tableau
- `filter()`: renvoi un tableau avec les éléments qui passent un test (fonction)
- `reduce()`: applique une fonction à un accumulateur et à chaque élément du tableau
- `find()`: renvoi le premier élément qui passe un test

```
let arr = [1, 2, 3];  
let newArr = arr.map(x => x * 2); // [2, 4, 6]
```

```
let arr = [1, 2, 3, 4];  
let filtered = arr.filter(x => x % 2 === 0); // [2, 4]
```

```
let arr = [1, 2, 3, 4];  
let sum = arr.reduce((acc, val) => acc + val, 0); // 10
```

```
let arr = [1, 2, 3];  
let found = arr.find(x => x > 1); // 2
```

Manipulation des tableaux

- `includes()`: vérifie si un tableau contient un élément spécifique
- `split()`: permet de diviser une chaîne de caractère en plusieurs sous-chaînes en fonction d'un séparateur spécifié et retourne un tableau contenant ces sous-chaînes
- `join()`: permet de combiner tous les éléments d'un tableau en une seule chaîne en insérant un séparateur entre chaque élément

```
let arr = [1, 2, 3];  
arr.includes(2); // true
```

```
let sentence = "I love JavaScript!";  
// Sépare la chaîne à chaque espace et retourne un tableau de mots  
let wordsArray = sentence.split(" ");  
  
console.log(wordsArray); // Affiche : ["I", "love", "JavaScript!"]
```

```
let words = ["I", "love", "JavaScript!"];  
// Combine le tableau en une chaîne, avec des espaces entre chaque mot  
let sentenceJoined = words.join(" ");  
  
console.log(sentenceJoined); // Affiche : "I love JavaScript!"
```

Manipulation des tableaux

- `slice()`: permet de copier une portion d'un tableau ou d'une chaîne de caractères sans modifier l'original. Elle prend deux arguments: l'index de départ et l'index de fin (non inclus)
- `splice()`: permet de modifier un tableau en y ajoutant, supprimant ou remplaçant des éléments. Elle prend plusieurs arguments : l'index de départ, le nombre d'éléments à supprimer et (facultatif) les éléments à ajouter.

```
let text = "Hello World";  
// Copie la partie de la chaîne de l'index 0 à 5  
let slicedText = text.slice(0, 5);  
  
console.log(slicedText); // Affiche : "Hello"
```

```
let colors = ["red", "green", "blue", "yellow"];  
// Supprime 2 éléments à partir de l'index 1  
let removedColors = colors.splice(1, 2);  
  
console.log(removedColors); // Affiche : ["green", "blue"]  
console.log(colors); // Le tableau original est modifié : ["red", "yellow"]
```

Manipulation des tableaux

- `indexOf()`: renvoie l'indice de la première occurrence de l'élément spécifié dans le tableau. Si l'élément n'est pas trouvé, elle retourne -1
- `indexOf()`: permet de trouver l'indice du premier élément d'un tableau qui satisfait une condition donnée par une fonction. Elle est utile quand tu as besoin d'une logique plus complexe pour déterminer quel élément chercher

```
let fruits = ["apple", "banana", "cherry", "apple"];  
let index = fruits.indexOf("banana");  
console.log(index); // Affiche 1
```

```
let fruits = ["apple", "banana", "cherry", "apple"];  
let index = fruits.indexOf("banana");  
console.log(index); // Affiche 1
```

Les objets (Object)

En JavaScript, un objet est une collection de paires clé-valeur. Les clés (ou propriétés) sont des chaînes (ou symboles) qui représentent des noms d'attributs, tandis que les valeurs peuvent être de tout type (nombres, chaînes, fonctions, objets, etc.). Les objets sont une des structures de base les plus couramment utilisées dans JavaScript pour représenter des données complexes.

Création d'un objet:

```
let car = {  
  brand: "Toyota",  
  model: "Corolla",  
  year: 2020  
};  
  
console.log(car.brand); // Affiche "Toyota"  
console.log(car["model"]); // Affiche "Corolla"
```

Avec le constructeur Object:

```
let voiture = new Object();  
voiture.marque = "Toyota";  
voiture.modèle = "Corolla";  
voiture.année = 2020;
```

Les objets (Object)

Accéder aux propriétés d'un objet:

```
console.log(car.year); // Affiche 2020  
console.log(car["brand"]); // Affiche "Toyota"
```

Ajouter, modifier ou supprimer une propriété:

```
// Ajouter une nouvelle propriété  
car.color = "red";  
console.log(car.color); // Affiche "red"  
  
// Modifier une propriété existante  
car.year = 2021;  
console.log(car.year); // Affiche 2021  
  
// Supprimer une propriété  
delete car.model;  
console.log(car.model); // Affiche undefined
```

Les objets (Object)

Méthodes d'un objet: les méthodes sont des fonctions qui sont des propriétés d'un objet

Utilisation de `this` dans les objets: le mot-clé `this` fait référence à l'objet actuel à l'intérieur de ses méthodes

```
let car = {  
  brand: "Toyota",  
  model: "Corolla",  
  year: 2020,  
  start: function() {  
    console.log("The car is starting");  
  }  
};  
  
car.start(); // Affiche "The car is starting"
```

```
let car = {  
  brand: "Toyota",  
  model: "Corolla",  
  year: 2020,  
  getInfo: function() {  
    return this.brand + " " + this.model + ", Year: " + this.year;  
  }  
};  
  
console.log(car.getInfo()); // Affiche "Toyota Corolla, Year: 2020"
```


Boucles sur les propriétés d'un objet:

Méthodes qui permettent d'obtenir les clés, les valeurs ou les paires clé-valeur d'un objet:

- `Object.keys(obj)`
- `Object.values(obj)`
- `Object.entries(obj)`
- `hasOwnProperty()`: vérifie si l'objet possède une propriété spécifique

```
for (let key in car) {  
    console.log(key + ": " + car[key]);  
}  
  
// Affiche :  
// brand: Toyota  
// model: Corolla  
// year: 2020
```

```
let user = {  
    name: "Léa",  
    age: 28,  
    profession: "Developer"  
};  
  
console.log(Object.keys(user)); // ["name", "age", "profession"]  
console.log(Object.values(user)); // ["Léa", 28, "Developer"]  
console.log(Object.entries(user)); // [["name", "Léa"], ["age", 28], ["profession", "Deve
```

```
let obj = {a: 1};  
obj.hasOwnProperty('a'); // true
```

Manipulation des dates

Création d'une date:

- Date actuelle
- Date spécifique
- À partir d'une chaîne de caractère

```
const now = new Date();  
console.log(now);
```

```
const specificDate = new Date(2024, 9, 12); // Octobre 12, 2024 (mois de 0 à 11)  
console.log(specificDate);
```

```
const fromString = new Date('2024-10-12T10:00:00');  
console.log(fromString);
```

Manipulation des dates

Obtenir les composant d'une date:

- L'année
- Le mois (commence par 0)
- Le jour du mois
- L'heure, les minutes et les secondes

```
const year = now.getFullYear();  
console.log(year);
```

```
const month = now.getMonth();  
console.log(month); // 9 pour octobre
```

```
const day = now.getDate();  
console.log(day);
```

```
const hours = now.getHours();  
const minutes = now.getMinutes();  
const seconds = now.getSeconds();  
console.log(`${hours}:${minutes}:${seconds}`);
```

Manipulation des dates

Modifier une date

On peut ajuster les différentes composantes d'une date

```
const newDate = new Date();  
newDate.setFullYear(2025); // Changer l'année à 2025  
newDate.setMonth(11); // Décembre  
newDate.setDate(25); // Jour de Noël  
console.log(newDate);
```

Comparer les dates

```
const date1 = new Date(2024, 9, 12);  
const date2 = new Date(2024, 9, 13);  
console.log(date1 < date2); // true
```

Manipulation des dates

Calculer des différences entre deux dates

- On peut utiliser `getTime()` qui renvoie le nombre de millisecondes écoulées depuis le 1^{er} janvier 1970
- On peut aussi la convertir en jour

```
const diff = date2.getTime() - date1.getTime();  
// Différence en millisecondes  
console.log(diff);  
  
const diffInDays = diff / (1000 * 60 * 60 * 24);  
// ms -> secondes -> minutes -> heures -> jours  
console.log(diffInDays);  
// 1 jour
```

Formater une date

```
const formattedDate = now.toLocaleDateString('fr-FR', {  
  year: 'numeric',  
  month: 'long',  
  day: 'numeric'  
});  
// Exemple : 12 octobre 2024  
console.log(formattedDate);
```

L'utilisation du Javascript dans un site web

Le **DOM** (Document Object Model) est une interface de programmation qui représente la structure d'un document HTML ou XML sous forme d'arbre d'objets. En d'autres termes, le DOM permet à JavaScript (et à d'autres langages de programmation) d'interagir avec et de manipuler le contenu, la structure et le style d'une page web.

Structure du DOM

Le DOM est structuré comme un arbre, où chaque nœud représente un objet correspondant à une partie du document. Par exemple :

- **Document** : Le nœud racine de l'arbre qui représente l'ensemble du document.
- **Éléments** : Chaque balise HTML est un nœud. Par exemple, `<div>`, `<p>`, `<a>`, etc.
- **Attributs** : Les attributs des balises (comme `id`, `class`, `src`, etc.) sont également des nœuds.
- **Texte** : Le texte à l'intérieur des balises est également un nœud.

Sélectionner les éléments du DOM

- Sélectionne un élément par son id
- Sélectionne tous les éléments avec une certaine classe
- Sélectionne tous les éléments d'un certain type de balise
- Sélectionne le premier élément correspondant à un sélecteur CSS
- Sélectionne tous les éléments correspondant à un sélecteur CSS

```
const element = document.getElementById('myElement');
```

```
const elements = document.getElementsByClassName('myClass');
```

```
const divs = document.getElementsByTagName('div');
```

```
const element = document.querySelector('.myClass');
```

```
const elements = document.querySelectorAll('.myClass');
```


Modifier le contenu

- Modifier le contenu HTML d'un élément

```
element.innerHTML = '<p>New content</p>';
```

- Modifier ou récupérer uniquement le texte contenu dans un élément (sans HTML)

```
element.textContent = 'New text';
```



Modifier les attributs

- Ajouter ou modifier un attribut d'un élément
- Récupérer la valeur d'un attribut spécifique
- Supprimer un attribut d'un élément

```
element.setAttribute('class', 'newClass');
```

```
const value = element.getAttribute('href');
```

```
element.removeAttribute('disabled');
```

Modifier le style

- Modifier directement une propriété CSS d'un élément

```
element.style.color = 'blue';
```

Ajouter ou supprimer les classes

- Ajouter une classe à un élément
- Supprimer une classe d'un élément
- Ajouter ou modifier une classe selon sa présence
- Vérifier si un élément a une classe spécifique

```
element.classList.add('newClass');
```

```
element.classList.remove('oldClass');
```

```
element.classList.toggle('active');
```

```
const hasClass = element.classList.contains('myClass');
```

Ajouter ou supprimer les classes

- Créer un nouvel élément HTML
- Ajouter un élément en tant que dernier enfant d'un autre élément
- Insérer un nouvel élément avant un autre élément
- Supprimer un élément enfant

```
const newDiv = document.createElement('div');
```

```
parentElement.appendChild(newDiv);
```

```
parentElement.insertBefore(newDiv, referenceDiv);
```

```
parentElement.removeChild(childDiv);
```

Les événements

- `addEventListener()`: un écouteur

d'événement à un élément

(ex. clic, survol)

```
element.addEventListener('click', function() {  
  console.log('Element clicked');  
});
```

```
function handleClick() {  
  console.log('Clicked!');  
}  
  
element.addEventListener('click', handleClick);
```

Paramètre de `addEventListener()`

1. Event(obligatoire) : Le type d'événement à écouter sous forme de chaîne de caractère.
2. Listener (obligatoire): La fonction callback qui sera exécutée lorsque l'événement se produira
3. Options facultatives:
 - Capture: gère l'écoute pendant la phase de capture si true, sinon pendant la phase de propagation
 - Once: L'événement est supprimé après avoir été déclenché une fois
 - Passive: si true, indique que l'événement n'appellera pas `preventDefault()`

L'utilisation du Javascript dans un site web

Événements de souris

- **click** : Se produit lorsque l'utilisateur clique sur un élément.
- **dblclick** : Se produit lorsque l'utilisateur double-clique sur un élément.
- **mouseover** : Se produit lorsque le curseur de la souris passe au-dessus d'un élément.
- **mouseout** : Se produit lorsque le curseur de la souris quitte un élément.
- **mousemove** : Se produit lorsque le curseur de la souris se déplace sur un élément.
- **mousedown** : Se produit lorsque l'utilisateur appuie sur un bouton de la souris.
- **mouseup** : Se produit lorsque l'utilisateur relâche un bouton de la souris.

L'utilisation du Javascript dans un site web


Événements de clavier

- **keydown** : Se produit lorsqu'une touche du clavier est enfoncée.
- **keyup** : Se produit lorsqu'une touche du clavier est relâchée.
- **keypress** : Se produit lorsqu'une touche est enfoncée (plus utilisé pour les touches de caractères).



L'utilisation du Javascript dans un site web

Événements de formulaire

- **submit** : Se produit lorsqu'un formulaire est soumis.
 - **change** : Se produit lorsque la valeur d'un élément de formulaire change (par exemple, un champ de texte ou un menu déroulant).
 - **input** : Se produit lorsque la valeur d'un champ de texte est modifiée.
 - **focus** : Se produit lorsqu'un élément (comme un champ de texte) reçoit le focus.
 - **blur** : Se produit lorsqu'un élément perd le focus.
- 

L'utilisation du Javascript dans un site web

Événements de fenêtre

- **load** : Se produit lorsque la page ou un élément (comme une image) est entièrement chargé.
- **resize** : Se produit lorsque la fenêtre du navigateur est redimensionnée.
- **scroll** : Se produit lorsque la fenêtre ou un élément défile.
- **unload** : Se produit lorsque la page est fermée ou déchargée.

Événements tactiles (pour les appareils mobiles)

- **touchstart** : Se produit lorsqu'un utilisateur touche un élément.
- **touchend** : Se produit lorsqu'un utilisateur relâche un élément.
- **touchmove** : Se produit lorsque l'utilisateur déplace son doigt sur l'écran.

L'utilisation du Javascript dans un site web

Événements de médias

- **play** : Se produit lorsque la lecture d'un élément multimédia commence.
- **pause** : Se produit lorsque la lecture d'un élément multimédia est interrompue.
- **ended** : Se produit lorsque la lecture d'un élément multimédia est terminée.



L'utilisation du Javascript dans un site web

Événements personnalisés

Vous pouvez également créer des événements personnalisés en utilisant le constructeur **CustomEvent**, puis les déclencher avec **dispatchEvent**.

Exemple d'utilisation

Voici un exemple montrant comment utiliser **addEventListener** avec différents types d'événements :

```
<!DOCTYPE html>
<html>
<head>
  <title>Événements</title>
</head>
<body>
  <button id="clickButton">Cliquez-moi</button>
  <input type="text" id="inputField" placeholder="Tapez quelque chose">

  <script>
    // Événement de clic sur le bouton
    const button = document.getElementById('clickButton');
    button.addEventListener('click', function() {
      alert('Bouton cliqué !');
    });

    // Événement de changement sur le champ de texte
    const input = document.getElementById('inputField');
    input.addEventListener('input', function() {
      console.log('Texte changé :', input.value);
    });
  </script>
</body>
</html>
```

L'utilisation du Javascript dans un site web

removeEventListener est une méthode en JavaScript utilisée pour retirer un gestionnaire d'événements précédemment ajouté à un élément avec **addEventListener**. Cela permet de désactiver la réaction à un événement spécifique, ce qui peut être utile pour optimiser les performances ou éviter des actions indésirables.

```
element.removeEventListener(event, function, useCapture);
```

- **event** : Une chaîne de caractères qui spécifie le type d'événement (par exemple, 'click', 'keydown', etc.).
- **function** : La fonction qui était associée à cet événement. **Il est important que cette fonction soit exactement la même référence que celle passée à addEventListener**, sinon **removeEventListener** ne fonctionnera pas.
- **useCapture** (optionnel) : Un booléen qui indique si la fonction doit être retirée de la phase de capture. Doit être le même que celui utilisé lors de l'ajout de l'événement.

L'utilisation du Javascript dans un site web

Exemple d'utilisation avec removeEventListener

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>removeEventListener Example</title>
</head>
<body>
  <button id="myButton">Clique ici</button>

  <script src="script.js"></script>
</body>
</html>
```

```
// Sélection du bouton
const button = document.getElementById('myButton');

// Fonction qui sera exécutée lors du clic
function handleClick() {
  console.log('Bouton cliqué !');

  // Retirer l'écoute de l'événement après 3 clics
  clickCount++;
  if (clickCount >= 3) {
    button.removeEventListener('click', handleClick);
    console.log('L\'écoute du clic est désactivée.');
```

Méthodes utiles pour la gestion des événements

setTimeout(): permet d'exécuter une fonction ou un code après un délai spécifié (en millisecondes).

Syntax:

- **function:** La fonction à exécuter une fois le délai écoulé
- **delay:** Le délai en millisecondes avant l'exécution de la fonction

Annulation d'un **setTimeout()**:

```
setTimeout(function, delay);
```

```
// Affiche un message après 2 secondes (2000 millisecondes)
setTimeout(function() {
  console.log('Hello, after 2 seconds!');
}, 2000);
```

```
let timeoutId = setTimeout(function() {
  console.log('This will not run');
}, 5000);

// Annule l'exécution de setTimeout
clearTimeout(timeoutId);
```

Méthodes utiles pour la gestion des événements

stopPropagation(): La méthode est utilisée dans la gestion des événements pour empêcher la propagation d'un événement à travers le DOM. Cela signifie qu'une fois appelée, l'événement ne se propagera pas aux éléments parents.

Pourquoi l'utiliser ?

En JavaScript, un événement (comme un clic) sur un élément peut "remonter" (propagation ascendante) jusqu'aux éléments parents via le flux de propagation (ou "bubbling"). Si vous souhaitez empêcher cet événement d'atteindre les éléments parents, **stopPropagation()** est la solution.

```
// Événement sur l'élément enfant
childElement.addEventListener('click', function(event) {
  console.log('Child clicked');
  event.stopPropagation(); // Empêche l'événement de se propager au parent
});

// Événement sur l'élément parent
parentElement.addEventListener('click', function() {
  console.log('Parent clicked');
});
```

Dans cet exemple :

- Si stopPropagation() n'est pas utilisé, cliquer sur l'élément enfant afficherait les deux messages : "Child clicked" et "Parent clicked", car l'événement remonte jusqu'au parent.
- Avec stopPropagation(), seul "Child clicked" sera affiché.

Méthodes utiles pour la gestion des événements

preventDefault(): la méthode est utilisée dans les gestionnaires d'événements pour empêcher le comportement par défaut de l'événement.

Exemple: intercepter le clic sur un lien sans que la page ne soit redirigée

```
event.preventDefault();
```

```
document.querySelector('a').addEventListener('click', function(event) {  
  event.preventDefault(); // Empêche la redirection  
  console.log('Le lien a été cliqué, mais sans redirection.');
```

preventDefault() vs stopPropagation() :

- **preventDefault()** : Empêche le **comportement par défaut** de l'élément sans stopper la propagation de l'événement. Par exemple, vous pouvez empêcher la soumission d'un formulaire tout en laissant l'événement continuer à être propagé aux éléments parents.
- **stopPropagation()** : Empêche la **propagation** de l'événement à travers le DOM, mais ne bloque pas le comportement par défaut.

Fonctions asynchrones

En JavaScript, une fonction asynchrone est une fonction qui peut s'exécuter en arrière-plan, sans bloquer le reste du programme. Cela permet de faire des choses comme attendre des données d'un serveur sans que l'application ne "gèle" pendant ce temps.

Pourquoi c'est utile ?

- Imagine que tu demandes des données à un serveur (par exemple, les informations météo). Cela peut prendre quelques secondes. Pendant ce temps, tu ne veux pas que ton programme arrête de fonctionner. Une fonction asynchrone permet d'attendre cette réponse sans bloquer l'application.

```
async function getWeather() {  
  const weatherData = await fetch('https://api.weather.com/data');  
  console.log(weatherData);  
}
```

Dans cet exemple, la fonction `getWeather` attend (`await`) que les données de l'API météo soient récupérées, mais pendant ce temps, le reste de ton code peut continuer à fonctionner.

Les Promesses (Promise)

Une promesse est un objet qui représente quelque chose qui va se passer dans le futur. Cette chose peut réussir ou échouer, et la promesse nous permet de savoir ce qui s'est passé.

3 états d'une promesse :

- Pending (en attente) : L'opération est en cours.
- Resolved (Fulfilled) : L'opération a réussi.
- Rejected : L'opération a échoué.

Dans cet exemple :

- `resolve` est appelé quand l'opération réussit.
- `reject` est appelé si l'opération échoue.

```
const myPromise = new Promise((resolve, reject) => {  
  let operationSuccess = true; // Disons que l'opération réussit  
  
  if (operationSuccess) {  
    resolve("L'opération a réussi !");  
  } else {  
    reject("L'opération a échoué.");  
  }  
});
```

Utiliser une promesse

Une fois que tu as une promesse, tu peux savoir si elle a réussi ou échoué en utilisant `then()` et `catch()`:

```
myPromise
  .then(result => {
    console.log(result); // Si l'opération a réussi, affiche "L'opération a réussi !"
  })
  .catch(error => {
    console.error(error); // Si l'opération échoue, affiche "L'opération a échoué."
  });
```

API (Application Programming Interface)

Une **API** est comme un serveur qui te donne des informations ou des services. Par exemple, tu peux demander des données sur la météo, des traductions, ou encore des informations sur des utilisateurs.

Pour **appeler une API**, on utilise souvent la fonction `fetch()` en JavaScript, qui retourne une promesse (on attend donc une réponse qui pourrait réussir ou échouer).

```
fetch('https://api.weather.com/data') // Appel de l'API météo
  .then(response => response.json()) // Convertit la réponse en format JSON
  .then(data => console.log(data)) // Affiche les données météo
  .catch(error => console.error('Erreur lors de la récupération des données :', error));
```

Combiner Promesse et API

Voici un exemple où tout est réuni : on appelle une API pour récupérer des données, et on utilise une promesse pour gérer le succès ou l'échec de cette opération :

```
async function fetchWeather() {  
  try {  
    const response = await fetch('https://api.weather.com/data'); // Appel API  
    const weatherData = await response.json(); // Attendre et transformer en JSON  
    console.log(weatherData); // Affiche les données météo  
  } catch (error) {  
    console.error('Erreur lors de la récupération des données : ', error); // Gérer les erreurs  
  }  
}  
  
fetchWeather(); // On appelle la fonction
```

Ressources en ligne pour apprendre JavaScript

- [MDN Web Docs \(Mozilla\)](#) Une ressource officielle et complète pour apprendre JavaScript, allant des bases aux concepts avancés. La documentation est bien expliquée et contient des exemples pratiques.
- [JavaScript.info](#) Un tutoriel détaillé qui couvre JavaScript moderne. Il est bien structuré et convient aux débutants comme aux développeurs plus expérimentés qui veulent approfondir certains concepts.
- [Codecademy – JavaScript](#) Un cours interactif qui te permet de pratiquer en direct tout en apprenant. Le cours couvre des sujets comme les fonctions, les objets et la manipulation du DOM.
- [FreeCodeCamp](#) FreeCodeCamp propose un parcours complet de JavaScript avec des projets pratiques et des exercices sur les algorithmes. Le contenu est très apprécié par la communauté des développeurs.
- [W3Schools – JavaScript](#) Un guide accessible et simple pour les débutants avec des exemples de code que tu peux tester directement dans le navigateur.