# Cours № 1 : JavaScript

Pr. Mohamed LACHGAR, ENSA - UCD 10/02/2023

## 1 Introduction à JavaScript

### 1.1 Introduction à JavaScript et à son utilisation sur le Web

JavaScript est un langage de programmation de scripts principalement utilisé pour développer des applications Web interactives. Il a été créé en 1995 par Brendan Eich chez Netscape Communications Corporation, et est maintenant développé par la communauté open-source.

JavaScript peut être utilisé pour ajouter de la dynamicité et de l'interactivité aux pages Web en permettant aux utilisateurs de faire des actions telles que des formulaires de saisie de données, des menus déroulants, des animations et des jeux en ligne, entre autres.

JavaScript s'exécute sur le navigateur Web et peut être inséré directement dans le code HTML ou dans un fichier externe lié à la page Web via une balise <script>. Cela permet aux développeurs de concevoir des applications Web plus riches et plus interactives sans avoir besoin de faire appel à un serveur ou à une base de données.

JavaScript est un langage de programmation orienté objet, ce qui signifie qu'il permet de créer des objets complexes et de les utiliser pour organiser et structurer le code. Il est également un langage de programmation interprété, ce qui signifie qu'il peut être écrit et exécuté sans avoir à le compiler.

### 1.2 Historique de JavaScript

JavaScript a été créé en 1995 par Brendan Eich chez Netscape Communications Corporation. À l'origine, il a été conçu pour ajouter des fonctionnalités interactives aux pages Web, telles que des formulaires de saisie de données et des animations.

En 1996, Netscape a publié la spécification de JavaScript, qui a été adoptée par de nombreux autres navigateurs Web. Au fil des ans, JavaScript est devenu l'un des langages de programmation les plus populaires pour le développement Web, avec une large prise en charge par les navigateurs Web et un écosystème en constante évolution de bibliothèques et de frameworks.

En 1997, Microsoft a introduit JScript, un langage de programmation similaire à JavaScript, pour sa plateforme Internet Explorer. Cependant, JavaScript est resté la norme pour les développeurs Web, et la spécification a continué d'évoluer au fil des ans pour prendre en compte les nouveaux développements et les nouvelles fonctionnalités.

De nos jours, JavaScript est utilisé pour développer des applications Web complexes, des sites Web dynamiques, des applications mobiles hybrides et même des applications de bureau.

Il est devenu un incontournable pour les développeurs Web, et son écosystème en constante évolution permet de concevoir des applications plus riches et plus interactives que jamais.

## 1.3 Installation de l'environnement de développement (IDE) et configuration du navigateur

Pour développer en JavaScript, vous aurez besoin d'un environnement de développement intégré (IDE) et d'un navigateur Web.

1. Installation d'un IDE :
   Il existe plusieurs options gratuites et payantes pour les IDE de JavaScript, y compris Visual Studio Code, Sublime Text, Atom, etc. Vous pouvez télécharger et installer l'IDE de votre choix sur votre ordinateur.

2. Configuration du navigateur :
   Pour tester votre code JavaScript, vous aurez besoin d'un navigateur Web compatible. Les navigateurs les plus couramment utilisés sont Google Chrome, Mozilla Firefox, et Microsoft Edge. Vous pouvez utiliser n'importe lequel de ces navigateurs pour développer et tester votre code.

Pour tester votre code JavaScript, vous pouvez utiliser la console JavaScript intégrée à votre navigateur. Pour accéder à la console, appuyez sur les touches Ctrl + Maj + J (sur Windows) ou Cmd + Option + J (sur Mac) lorsque vous êtes sur une page Web. Vous pouvez également insérer votre code JavaScript directement dans la page Web à l'aide de la balise <script> et de la fonction console.log() pour afficher les résultats dans la console.

## 1.4 Types de données de base de JavaScript (number, string, boolean, null, undefined)

JavaScript prend en charge plusieurs types de données de base, notamment :

- Number: Les nombres peuvent être entiers ou décimaux et sont utilisés pour représenter les nombres réels ou les nombres mathématiques.
- String: Les chaînes de caractères sont utilisées pour représenter des textes et peuvent être définies en utilisant des guillemets simples ou doubles.
- Boolean: Les valeurs booléennes peuvent être soit vraies (true) ou fausses (false), et sont utilisées pour les opérations logiques.
- Null: La valeur null représente l'absence de valeur et est souvent utilisée pour définir une variable sans lui attribuer de valeur.
- Undefined: La valeur undefined signifie que la variable a été déclarée mais n'a pas encore été initialisée ou n'a pas été assignée à une valeur.

En plus de ces types de données de base, JavaScript prend également en charge les objets et les tableaux, qui sont des structures de données plus complexes permettant de stocker et de manipuler des données de différents types. Les objets et les tableaux seront couverts en détail dans les séances ultérieures.

### 1.5 Variables en JavaScript et déclaration

Les variables en JavaScript sont des espaces de mémoire utilisés pour stocker des valeurs. Les variables peuvent être déclarées en utilisant le mot clé "var" ou les mots clés "let" ou "const" introduits en ECMAScript 6. La déclaration de variable consiste à nommer la variable et à lui attribuer une valeur initiale :

```
1 var nomDeVariable = valeur;
2 let nomDeVariable = valeur;
3 const nomDeVariable = valeur;
```

La différence entre les mots clés "var", "let" et "const" est la portée et la réaffectation :

- "var" déclare une variable avec une portée globale ou fonctionnelle, ce qui signifie que la variable est accessible dans tout le code, y compris à l'intérieur des fonctions. De plus, les variables déclarées avec "var" peuvent être réaffectées à tout moment.
- "let" déclare une variable avec une portée de bloc, ce qui signifie que la variable n'est accessible que dans le bloc de code où elle a été déclarée. Les variables déclarées avec "let" peuvent également être réaffectées.
- "const" déclare une constante, ce qui signifie que la valeur de la variable ne peut pas être réaffectée après la première affectation. "const" a la même portée que "let".

En général, il est recommandé d'utiliser "const" pour les valeurs qui ne doivent pas être modifiées et "let" pour les valeurs qui doivent être modifiées au cours de l'exécution du programme.

### 1.6 Opérateurs arithmétiques et logiques

JavaScript prend en charge plusieurs opérateurs arithmétiques et logiques, qui peuvent être utilisés pour effectuer des calculs et des comparaisons :

1. Opérateurs arithmétiques :
    - Addition (+)
    - Soustraction (-)
    - Multiplication (*)
    - Division (/)
    - Modulo (%)

- Incrémentation (++)
- Décrémentation (–)

2. Opérateurs de comparaison :
   - Égalité (==)
   - Inégalité (!=)
   - Strictement égalité (===)
   - Strictement inégalité (!==)
   - Supériorité (> )
   - Infériorité (<)
   - Supériorité ou égalité (>=)
   - Infériorité ou égalité (<=)
3. Opérateurs logiques :
   - ET ()
   - OU (||)
   - NON (!)

Ces opérateurs peuvent être utilisés pour effectuer des calculs simples, des comparaisons et des opérations logiques. Les résultats de ces opérations peuvent être affectés à des variables ou utilisés dans des conditions pour contrôler le flux de programme. Les opérateurs arithmétiques et logiques seront couverts en détail dans les séances ultérieures.

### 1.7 Conditions (if, else, else if)

Les conditions en JavaScript permettent de contrôler le flux de programme en fonction de la valeur de certaines expressions. Les instructions sont exécutées uniquement si la condition est vraie.

La structure de base pour une condition en JavaScript est l'instruction "if". Si la condition est vraie, les instructions définies dans le bloc de code sont exécutées :

```
1 if (condition) {
2   // Instructions à exécuter si la condition est vraie
3 }
```

Vous pouvez également utiliser "else" pour exécuter un autre bloc de code si la condition n'est pas vraie :

```
1 if (condition) {
2   // Instructions à exécuter si la condition est vraie
3 } else {
```

```
4   // Instructions à exécuter si la condition n est pas vraie
5 }
```

Si vous avez plusieurs conditions à vérifier, vous pouvez utiliser "else if" pour les ajouter :

```
1 if (condition1) {
2   // Instructions à exécuter si la condition 1 est vraie
3 } else if (condition2) {
4   // Instructions à exécuter si la condition 2 est vraie
5 } else {
6   // Instructions à exécuter si aucune des conditions n est vraie
7 }
```

## 1.8 Activités pratiques :

### 1.8.1 Créer une application qui demande à l'utilisateur d'entrer son âge et affiche s'il est majeur ou mineur

1. Utilisez la méthode "prompt()" pour demander à l'utilisateur son âge :

```
1     var age = prompt("Entrez votre âge :");
```

2. Convertir la valeur entrée en nombre :

```
1      var age = parseInt(prompt("Entrez votre âge :"));
```

3. Vérifiez si l'âge est supérieur ou égal à 18 ans :

```
1     if (age >= 18) {
2       console.log("Vous êtes majeur.");
3     } else {
4       console.log("Vous êtes mineur.");
5     }
```

4. Exécutez le code dans un navigateur en ouvrant la console développeur (F12 dans la plupart des navigateurs).
   Voici le code complet de l'application :

```
1     var age = parseInt(prompt("Entrez votre âge :"));
2     if (age >= 18) {
3       console.log("Vous êtes majeur.");
```

```
4     } else {
5       console.log("Vous êtes mineur.");
6     }
```

Cet exemple montre comment utiliser une condition "if...else" pour contrôler le flux de programme en fonction de la valeur entrée par l'utilisateur. Les instructions dans le bloc "if" sont exécutées si l'âge est supérieur ou égal à 18 ans, sinon les instructions dans le bloc "else" sont exécutées.

### 1.8.2 Créer une application de calculatrice simple qui effectue des opérations de base (addition, soustraction, multiplication, division).

Créer une application de calculatrice simple en utilisant JavaScript qui effectue des opérations de base (addition, soustraction, multiplication, division).

1. Demandez à l'utilisateur de saisir les nombres à utiliser pour le calcul :

```
1     var nombre1 = parseFloat(prompt("Entrez le premier nombre :"));
2     var nombre2 = parseFloat(prompt("Entrez le deuxi me nombre :"))↩
          ;
```

2. Demandez à l'utilisateur de saisir l'opération à effectuer :

```
1     var operateur = prompt("Entrez l'opération à effectuer (+, -, *,↩
          /) :");
```

3. Effectuez le calcul en utilisant une condition "switch" pour déterminer l'opération à effectuer :

```
1     switch (operateur) {
2       case "+":
3         console.log(nombre1 + nombre2);
4         break;
5       case "-":
6         console.log(nombre1 - nombre2);
7         break;
8       case "*":
9         console.log(nombre1 * nombre2);
10        break;
11      case "/":
12        console.log(nombre1 / nombre2);
13        break;
14      default:
```

```
15          console.log("Opération non valide");
16      }
```

4. Exécutez le code dans un navigateur en ouvrant la console développeur (F12 dans la plupart des navigateurs).

Voici le code complet de l'application :

```
var nombre1 = parseFloat(prompt("Entrez le premier nombre :"));
var nombre2 = parseFloat(prompt("Entrez le deuxi me nombre :"));
var operateur = prompt("Entrez l'opération à effectuer (+, -, *, /) :");

switch (operateur) {
  case "+":
    console.log(nombre1 + nombre2);
    break;
  case "-":
    console.log(nombre1 - nombre2);
    break;
  case "*":
    console.log(nombre1 * nombre2);
    break;
  case "/":
    console.log(nombre1 / nombre2);
    break;
  default:
    console.log("Opération non valide");
}
```

Cet exemple montre comment utiliser une condition "switch" pour contrôler le flux de programme en fonction de la valeur de l'opération entrée par l'utilisateur. Les instructions associées à chaque opération (+, -, *, /) sont exécutées et le résultat est affiché dans la console.

# 2 Boucles et fonctions en JavaScript

## 2.1 Boucles (for, while, do-while)

Les boucles sont un moyen de répéter des instructions plusieurs fois dans un programme JavaScript. Il existe trois types de boucles dans JavaScript : "for", "while" et "do-while".

1. Boucle "for" : La boucle "for" est utilisée pour répéter un nombre spécifique de fois. Elle a la syntaxe suivante :

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

```
3       }
```

Lorsqu'une boucle "for" est exécutée, les étapes suivantes se produisent :

- L'initialisation se produit une seule fois au début de la boucle.
- La condition est vérifiée avant chaque tour de boucle. Si la condition est vraie, le corps de la boucle est exécuté. Sinon, la boucle s'arrête.
- L'incrémentation se produit à la fin de chaque tour de boucle.

2. Boucle "while" : La boucle "while" est utilisée pour répéter un nombre indéterminé de fois. Elle a la syntaxe suivante :

```
1       while (condition) {
2       // instructions à exécuter
3       }
```

Lorsqu'une boucle "while" est exécutée, les étapes suivantes se produisent :

- La condition est vérifiée avant chaque tour de boucle. Si la condition est vraie, le corps de la boucle est exécuté. Sinon, la boucle s'arrête.

3. Boucle "do-while" : La boucle "do-while" est similaire à la boucle "while", sauf que le corps de la boucle est toujours exécuté au moins une fois, même si la condition est fausse. Elle a la syntaxe suivante :

```
1     do {
2       // instructions à exécuter
3     } while (condition);
```

Lorsqu'une boucle "do-while" est exécutée, les étapes suivantes se produisent :

- Le corps de la boucle est exécuté une première fois.
- La condition est vérifiée après chaque tour de boucle. Si la condition est vraie, le corps de la boucle est exécuté à nouveau. Sinon, la boucle s'arrête.

## 2.2 Fonctions en JavaScript (déclaration, appel, retour de valeur)

Les fonctions sont un bloc de code qui peut être exécuté plusieurs fois dans un programme JavaScript. Elles permettent de regrouper du code réutilisable en un seul endroit et peuvent être appelées à partir de n'importe où dans le programme.

1. Déclaration de fonction : Pour déclarer une fonction en JavaScript, vous pouvez utiliser la syntaxe suivante :

```
function nomDeLaFonction(param tres) {
 // corps de la fonction
}
```

Les paramètres sont des variables qui peuvent être passées à la fonction lors de son appel.

2. Appel de fonction : Pour appeler une fonction en JavaScript, vous pouvez utiliser la syntaxe suivante :

```
nomDeLaFonction(arguments);
```

Les arguments sont les valeurs qui sont passées à la fonction lors de son appel.

3. Retour de valeur : Les fonctions peuvent retourner une valeur en utilisant la syntaxe suivante :

```
return valeur;
```

La valeur retournée par la fonction peut être assignée à une variable pour être utilisée plus tard dans le programme.
Exemple :

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

var resultat = addition(5, 3);
console.log(resultat); // affiche 8
```

## 2.3 Paramètres de fonction et arguments

Les paramètres de fonction sont des variables définies dans la déclaration de la fonction qui peuvent être utilisées dans le corps de la fonction. Lorsque la fonction est appelée, des valeurs peuvent être passées à ces paramètres sous forme d'arguments.
Exemple :

```
function addition(a, b) {
   return a + b;
}
var resultat = addition(5, 3);
console.log(resultat); // affiche 8
```

Dans l'exemple ci-dessus, les paramètres a et b sont définis dans la déclaration de la fonction addition. Lorsque la fonction est appelée en lui passant les valeurs 5 et 3 en tant qu'arguments, les paramètres a et b reçoivent ces valeurs. Le corps de la fonction effectue l'opération d'addition et retourne la valeur résultante qui est assignée à la variable resultat.

## 2.4 Scope de variable et contextes d'exécution

Le scope de variable détermine la portée d'une variable dans un programme JavaScript. Il définit l'endroit où une variable peut être accessible dans le code. Il existe deux types de scope de variable en JavaScript : le scope global et le scope local.

1. Scope global : Une variable déclarée en dehors de toutes les fonctions est appelée une variable globale. Elle peut être accessible depuis n'importe où dans le code, y compris à l'intérieur de toutes les fonctions.
   Exemple :

```
1    var nom = "John";
2    function afficherNom() {
3        console.log(nom);
4    }
5    afficherNom(); // affiche "John"
```

2. Scope local : Une variable déclarée à l'intérieur d'une fonction est appelée une variable locale. Elle n'est accessible que à l'intérieur de cette fonction et ne peut pas être accédée en dehors de celle-ci.
   Exemple :

```
1    function afficherNom() {
2        var nom = "John";
3        console.log(nom);
4    }
5    afficherNom(); // affiche "John"
6    console.log(nom); // Uncaught ReferenceError: nom is not defined
```

Le contexte d'exécution est l'environnement dans lequel une fonction est exécutée. Il définit les variables et les fonctions disponibles pour la fonction en cours d'exécution. Chaque fonction a son propre contexte d'exécution qui est créé lorsque la fonction est appelée. Les variables déclarées à l'intérieur de la fonction sont accessibles uniquement dans le contexte d'exécution de cette fonction.

## 2.5 Activités pratiques :

### 2.5.1 Créer une application qui affiche les nombres de 1 à 100 à l'aide de boucles

Créez une application qui affiche les nombres de 1 à 100 dans la console à l'aide d'une boucle. Utilisez la boucle for pour boucler à travers les nombres de 1 à 100 et afficher chaque nombre sur une nouvelle ligne dans la console. Assurez-vous que la boucle s'arrête une fois que les 100 nombres ont été affichés.

Voici les étapes pour créer une application qui affiche les nombres de 1 à 100 à l'aide de boucles :

1. Définir un compteur i avec une valeur initiale de 1.
2. Utiliser la boucle for pour boucler à travers les nombres de 1 à 100. La condition de boucle sera de continuer tant que i est inférieur ou égal à 100.
3. Dans le corps de la boucle for, utiliser la méthode console.log pour afficher la valeur actuelle de i.
4. Incrémenter la valeur de i de 1 à chaque tour de boucle.
5. Une fois la boucle terminée, l'application aura affiché les nombres de 1 à 100 dans la console.

Voici le code JavaScript correspondant :

```
1     for (var i = 1; i <= 100; i++) {
2       console.log(i);
3     }
```

Ce code utilise la boucle for pour boucler à travers les nombres de 1 à 100. À chaque tour de boucle, la valeur actuelle de i est affichée dans la console et la valeur de i est incrémentée de 1. La boucle continue jusqu'à ce que la valeur de i soit supérieure à 100.

### 2.5.2 Créer une application de conversion de devises (par exemple, EUR en USD) qui utilise une fonction de conversion

Créer une application de conversion de devises en utilisant HTML et JavaScript.

- Etape 1: Créez un formulaire HTML qui demande à l'utilisateur de saisir un montant dans une devise spécifique.
- Etape 2: Ajoutez une liste déroulante à votre formulaire qui permet à l'utilisateur de sélectionner la devise d'origine et la devise de conversion.
- Etape 3: Écrivez une fonction JavaScript qui convertit le montant en utilisant des taux de change prédéterminés.

- Etape 4: Ajoutez un événement "submit" à votre formulaire qui appelle votre fonction de conversion de devise lorsque l'utilisateur soumet le formulaire.
- Etape 5: Affichez le résultat de la conversion sur votre page web.
- Etape 6: Testez votre application en convertissant différents montants dans différentes devises pour vous assurer qu'elle fonctionne correctement.

**Solution :**

Voici un exemple de code HTML et JavaScript pour créer une application de conversion de devises :

HTML :

```
<form>
  <label for="amount">Montant:</label>
  <input type="number" id="amount" name="amount" required>

  <label for="from">De:</label>
  <select id="from" name="from">
    <option value="USD">USD - Dollar américain</option>
    <option value="EUR">EUR - Euro</option>
    <option value="GBP">GBP - Livre sterling</option>
  </select>

  <label for="to">  :</label>
  <select id="to" name="to">
    <option value="USD">USD - Dollar américain</option>
    <option value="EUR">EUR - Euro</option>
    <option value="GBP">GBP - Livre sterling</option>
  </select>

  <input type="submit" value="Convertir">
</form>

<div id="result"></div>
```

JavaScript :

```
const exchangeRates = {
  USD: {
    EUR: 0.82,
    GBP: 0.72
  },
  EUR: {
    USD: 1.22,
    GBP: 0.87
  },
```

```
  GBP: {
    USD: 1.39,
    EUR: 1.15
  }
};

function convertCurrency(event) {
  event.preventDefault();

  const amount = Number(document.getElementById("amount").value);
  const from = document.getElementById("from").value;
  const to = document.getElementById("to").value;

  const rate = exchangeRates[from][to];
  const convertedAmount = amount * rate;

  document.getElementById("result").innerHTML = `${amount} ${from} = ${convertedAmount} ${to}`;
}

document.querySelector("form").addEventListener("submit", convertCurrency);
```

Cette application de conversion de devises utilise des taux de change prédéterminés pour convertir le montant saisi par l'utilisateur. Lorsque l'utilisateur soumet le formulaire, la fonction convertCurrency() est appelée pour récupérer les valeurs saisies et calculer le montant converti. Le résultat est affiché sur la page web dans la balise . N'oubliez pas de tester votre application en convertissant différents montants dans différentes devises pour vous assurer qu'elle fonctionne correctement.

# 3 Tableaux et objets en JavaScript

## 3.1 Tableaux en JavaScript (déclaration, accès aux éléments, parcours)

En JavaScript, un tableau est une structure de données qui permet de stocker une collection ordonnée d'éléments, tels que des nombres, des chaînes de caractères, des objets, etc. Les tableaux peuvent être déclarés de la manière suivante :

```
    // Déclaration d un tableau vide
    const tableauVide = [];

    // Déclaration d un tableau avec des éléments
    const tableau = [1, 2, "trois", {quatre: 4}, [5, 6]];
```

Pour accéder aux éléments d'un tableau, on utilise la notation crochet [] en spécifiant l'index de l'élément (l'index du premier élément est 0) :

```
const tableau = ["un", "deux", "trois"];
console.log(tableau[0]); // affiche "un"
console.log(tableau[1]); // affiche "deux"
console.log(tableau[2]); // affiche "trois"
```

On peut également parcourir un tableau à l'aide d'une boucle for :

```
const tableau = ["un", "deux", "trois"];
for (let i = 0; i < tableau.length; i++) {
  console.log(tableau[i]);
}
// affiche "un", "deux", "trois"
```

Il est également possible d'utiliser la méthode forEach() sur un tableau pour exécuter une fonction sur chaque élément :

```
const tableau = ["un", "deux", "trois"];
tableau.forEach(function(element) {
  console.log(element);
});
// affiche "un", "deux", "trois"
```

Enfin, on peut ajouter des éléments à un tableau avec la méthode push() et supprimer le dernier élément avec la méthode pop() :

```
const tableau = ["un", "deux"];
tableau.push("trois");
console.log(tableau); // affiche ["un", "deux", "trois"]

tableau.pop();
console.log(tableau); // affiche ["un", "deux"]
```

### 3.2 Objets en JavaScript (déclaration, accès aux propriétés, modification)

En JavaScript, un objet est une structure de données qui permet de stocker une collection de propriétés, qui peuvent être des variables, des fonctions ou même d'autres objets. Les objets peuvent être déclarés de la manière suivante :

```
// Déclaration d un objet vide
const objetVide = {};

// Déclaration d un objet avec des propriétés
const objet = {
  propriete1: "valeur1",
  propriete2: 2,
  propriete3: function() {
    console.log("Hello world!");
  }
};
```

Pour accéder aux propriétés d'un objet, on utilise la notation point . en spécifiant le nom de la propriété :

```
const objet = {
  propriete1: "valeur1",
  propriete2: 2
};
console.log(objet.propriete1); // affiche "valeur1"
console.log(objet.propriete2); // affiche 2
```

On peut également accéder aux propriétés d'un objet à l'aide de la notation crochet [] en spécifiant le nom de la propriété en tant que chaîne de caractères :

```
const objet = {
  propriete1: "valeur1",
  propriete2: 2
};
console.log(objet["propriete1"]); // affiche "valeur1"
console.log(objet["propriete2"]); // affiche 2
```

Pour modifier la valeur d'une propriété, on utilise simplement la notation point ou crochet suivie de l'opérateur d'assignation = :

```
const objet = {
  propriete1: "valeur1",
  propriete2: 2
};
objet.propriete1 = "nouvelleValeur";
console.log(objet.propriete1); // affiche "nouvelleValeur"

objet["propriete2"] = 3;
console.log(objet.propriete2); // affiche 3
```

On peut également ajouter de nouvelles propriétés à un objet de la même manière :

```
1     const objet = {
2       propriete1: "valeur1",
3       propriete2: 2
4     };
5     objet.propriete3 = "valeur3";
6     console.log(objet.propriete3); // affiche "valeur3"
7
8     objet["propriete4"] = 4;
9     console.log(objet.propriete4); // affiche 4
```

### 3.3 Méthodes d'objet et méthodes de tableau

En JavaScript, les objets et les tableaux disposent de méthodes qui permettent de réaliser des opérations courantes de manière plus simple. Voici quelques exemples :
Méthodes d'objet :

- Object.keys(objet): renvoie un tableau contenant les noms de toutes les propriétés d'un objet.
- Object.values(objet): renvoie un tableau contenant les valeurs de toutes les propriétés d'un objet.
- Object.entries(objet): renvoie un tableau contenant des tableaux de deux éléments correspondant aux paires nom/valeur de toutes les propriétés d'un objet.

Exemple :

```
1     const objet = {
2       propriete1: "valeur1",
3       propriete2: 2
4     };
5
6     console.log(Object.keys(objet)); // affiche ["propriete1", "propriete2↩
         "]
7     console.log(Object.values(objet)); // affiche ["valeur1", 2]
8     console.log(Object.entries(objet)); // affiche [["propriete1", "↩
         valeur1"], ["propriete2", 2]]
```

Méthodes de tableau :

- tableau.forEach(fonction): applique une fonction à chaque élément du tableau.
- tableau.map(fonction): renvoie un nouveau tableau contenant les valeurs renvoyées par une fonction appliquée à chaque élément du tableau.

- tableau.filter(fonction): renvoie un nouveau tableau contenant les éléments pour lesquels une fonction renvoie true.
- tableau.reduce(fonction): applique une fonction à chaque paire d'éléments consécutifs du tableau pour réduire le tableau à une seule valeur.

Exemple :

```
1    const tableau = [1, 2, 3, 4];
2
3    tableau.forEach(element => console.log(element)); // affiche 1, 2, 3, ↩
        4
4    const nouveauTableau = tableau.map(element => element * 2);
5    console.log(nouveauTableau); // affiche [2, 4, 6, 8]
6    const tableauFiltre = tableau.filter(element => element > 2);
7    console.log(tableauFiltre); // affiche [3, 4]
8    const resultat = tableau.reduce((accumulation, element) => ↩
        accumulation + element);
9    console.log(resultat); // affiche 10
```

Ces méthodes sont très utiles pour effectuer des opérations sur des objets ou des tableaux de manière concise et lisible.

### 3.4 JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) est un format de données textuelles utilisé pour échanger des données entre applications. Il est facile à lire et à écrire pour les humains, tout en étant facile à parser et à générer pour les machines. Il est basé sur une syntaxe d'objet en JavaScript, mais peut être utilisé avec de nombreux langages de programmation différents.

La syntaxe de JSON est similaire à celle des objets JavaScript, avec quelques différences. Voici un exemple de JSON :

```
1    {
2      "nom": "Ali",
3      "age": 28,
4      "sexe": "Homme",
5      "adresse": {
6        "rue": "Avenue Hassan II",
7        "ville": "Casablanca",
8        "code_postal": "20000"
9      },
10     "telephone": [
11       {
12         "type": "domicile",
13         "numero": "+212 522 22 22 22"
14       },
```

```
      {
        "type": "mobile",
        "numero": "+212 6 11 11 11 11"
      }
    ]
}
```

Dans cet exemple, nous avons les propriétés suivantes :

- nom: le nom de la personne, de type string.
- age: l'âge de la personne, de type number.
- sexe: le sexe de la personne, de type string.
- adresse: un objet représentant l'adresse de la personne, avec les propriétés rue, ville et code_postal, toutes de type string.
- telephone: un tableau d'objets représentant les numéros de téléphone de la personne, avec les propriétés type et numero, toutes deux de type string.

Dans cet exemple, les numéros de téléphone sont stockés dans un tableau car il est possible que la personne ait plusieurs numéros de téléphone. Le numéro de téléphone est stocké sous forme de chaîne de caractères avec le préfixe +212 qui représente l'indicatif téléphonique du Maroc.

En JavaScript, pour convertir un objet JavaScript en une chaîne JSON, on utilise la fonction JSON.stringify(). Cette fonction prend un objet JavaScript en entrée et retourne une chaîne de caractères contenant une représentation JSON de l'objet. Voici un exemple :

```
const personne = {
  nom: "Ali",
  age: 28,
  sexe: "Homme",
  adresse: {
    rue: "Avenue Hassan II",
    ville: "Casablanca",
    code_postal: "20000"
  },
  telephone: [
    { type: "domicile", numero: "+212 522 22 22 22" },
    { type: "mobile", numero: "+212 6 11 11 11 11" }
  ]
};

const json = JSON.stringify(personne);
console.log(json);
```

Dans cet exemple, la fonction JSON.stringify() prend l'objet personne en entrée et retourne une chaîne de caractères contenant la représentation JSON de cet objet.

Pour convertir une chaîne JSON en un objet JavaScript, on utilise la fonction JSON.parse(). Cette fonction prend une chaîne JSON en entrée et retourne un objet JavaScript. Voici un exemple :

```
1    const json = '{"nom":"Ali","age":28,"sexe":"Homme","adresse":{"rue":"↩
        Avenue Hassan II","ville":"Casablanca","code_postal":"20000"},"↩
        telephone":[{"type":"domicile","numero":"+212 522 22 22 22"},{"↩
        type":"mobile","numero":"+212 6 11 11 11 11"}]}';
2
3    const personne = JSON.parse(json);
4    console.log(personne);
```

Dans cet exemple, la fonction JSON.parse() prend la chaîne JSON json en entrée et retourne l'objet JavaScript correspondant.

Il est important de noter que la syntaxe JSON doit respecter certaines règles, telles que l'utilisation de guillemets doubles pour les noms de propriétés et les chaînes de caractères, et l'absence de commentaires.

## 3.5 Activités pratiques :

### 3.5.1 Créer une application qui stocke des informations sur plusieurs livres (titre, auteur, année de publication) dans un tableau et les affiche

Voici les étapes pour créer une application qui stocke des informations sur plusieurs livres (titre, auteur, année de publication) dans un tableau, avec ajout via un formulaire HTML et affichage dans un tableau HTML :

1. Créez une page HTML avec un formulaire permettant d'ajouter un livre. Le formulaire doit comporter des champs pour le titre, l'auteur et l'année de publication, ainsi qu'un bouton "Ajouter".

2. Dans votre page HTML, ajoutez un tableau pour afficher les livres. Le tableau doit avoir des colonnes pour le titre, l'auteur et l'année de publication.

3. Créez un tableau en JavaScript qui stockera les informations sur les livres. Chaque élément du tableau représente un livre et contient les propriétés suivantes : titre, auteur et année de publication.

4. Créez une fonction en JavaScript pour ajouter un livre au tableau lorsque l'utilisateur soumet le formulaire. Cette fonction doit prendre les données saisies par l'utilisateur dans le formulaire et les ajouter au tableau.

5. Créez une fonction en JavaScript pour afficher les livres dans le tableau HTML. Cette fonction doit parcourir le tableau de livres et ajouter une nouvelle ligne au tableau HTML pour chaque livre, en affichant le titre, l'auteur et l'année de publication.
6. Ajoutez un événement "submit" à votre formulaire HTML qui appelle la fonction d'ajout de livre. Lorsque l'utilisateur soumet le formulaire, la fonction d'ajout de livre doit être appelée pour ajouter le livre au tableau.
7. Appelez la fonction d'affichage de livre pour afficher les livres dans le tableau HTML à chaque fois qu'un livre est ajouté.
8. Testez votre application en ajoutant plusieurs livres et en vérifiant qu'ils sont correctement affichés dans le tableau.

**Indications :** Voici le code pour chaque étape de l'application :

1. Formulaire HTML pour ajouter un livre :

```
<form id="add-book-form">
  <label for="title">Titre :</label>
  <input type="text" id="title" name="title"><br>

  <label for="author">Auteur :</label>
  <input type="text" id="author" name="author"><br>

  <label for="year">Année de publication :</label>
  <input type="text" id="year" name="year"><br>

  <button type="submit">Ajouter</button>
</form>
```

2. Tableau HTML pour afficher les livres

```
<table id="book-table">
  <thead>
    <tr>
      <th>Titre</th>
      <th>Auteur</th>
      <th>Année de publication</th>
    </tr>
  </thead>
  <tbody>
  </tbody>
</table>
```

3. Tableau JavaScript pour stocker les informations sur les livres :

```
let books = [];
```

4. Fonction JavaScript pour ajouter un livre :

```
function addBook(event) {
  event.preventDefault();

  const title = document.getElementById("title").value;
  const author = document.getElementById("author").value;
  const year = document.getElementById("year").value;

  const book = { title, author, year };
  books.push(book);

  document.getElementById("add-book-form").reset();
}
```

5. Fonction JavaScript pour afficher les livres dans le tableau HTML :

```
function displayBooks() {
  const tableBody = document.getElementById("book-table").↩
     getElementsByTagName("tbody")[0];

  tableBody.innerHTML = "";

  for (let i = 0; i < books.length; i++) {
    const book = books[i];

    const row = tableBody.insertRow(i);

    const titleCell = row.insertCell(0);
    titleCell.innerHTML = book.title;

    const authorCell = row.insertCell(1);
    authorCell.innerHTML = book.author;

    const yearCell = row.insertCell(2);
    yearCell.innerHTML = book.year;
  }
}
```

6. Ajout de l'événement "submit" pour appeler la fonction d'ajout de livre :

```
1       const addBookForm = document.getElementById("add-book-form");
2       addBookForm.addEventListener("submit", addBook);
```

7. Appel de la fonction d'affichage de livre pour afficher les livres dans le tableau HTML :

```
1       displayBooks();
```

### 3.5.2 Créer une application de gestion de contact qui stocke les informations de contact (nom, numéro de téléphone, adresse électronique) dans un objet et les affiche

Créez une application de gestion de contacts en utilisant JavaScript, HTML et le localstorage.

L'application doit permettre à l'utilisateur de stocker les informations de contact (nom, numéro de téléphone, adresse électronique) dans le localstorage sous format JSON. L'ajout de contacts doit se faire en utilisant un formulaire HTML et l'affichage de la liste de contacts doit se faire dans un tableau HTML.

Voici les étapes pour la création de l'application :

1. Créer un fichier HTML pour l'interface utilisateur.
2. Inclure une section pour le formulaire d'ajout de contacts et une section pour l'affichage de la liste de contacts.
3. Créer un fichier JavaScript pour la logique de l'application.
4. Écrire une fonction pour ajouter un contact en utilisant les informations du formulaire.
5. Écrire une fonction pour récupérer la liste des contacts depuis le localstorage.
6. Écrire une fonction pour afficher la liste des contacts dans le tableau HTML.
7. Créer le formulaire HTML pour permettre à l'utilisateur de saisir les informations du contact.
8. Ajouter le tableau HTML pour afficher la liste des contacts.

Le résultat final devrait être une application de gestion de contacts fonctionnelle qui stocke les informations de contact dans le localstorage sous format JSON et affiche les contacts dans un tableau HTML.

**Indications :**

1. Créer un fichier HTML contenant un formulaire pour ajouter un nouveau contact et un tableau pour afficher tous les contacts existants. Voici un exemple de code HTML de base pour cela:

```
1       <!DOCTYPE html>
2       <html>
```

```
    <head>
        <title>Ma liste de contacts</title>
    </head>
    <body>
        <h1>Ma liste de contacts</h1>
        <form id="add-contact-form">
            <label for="name">Nom:</label>
            <input type="text" id="name" name="name" required>

            <label for="phone">Numéro de téléphone:</label>
            <input type="tel" id="phone" name="phone" required>

            <label for="email">Adresse électronique:</label>
            <input type="email" id="email" name="email" required>

            <button type="submit">Ajouter un contact</button>
        </form>

        <table>
            <thead>
                <tr>
                    <th>Nom</th>
                    <th>Numéro de téléphone</th>
                    <th>Adresse électronique</th>
                </tr>
            </thead>
            <tbody id="contact-list">
            </tbody>
        </table>

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

2. Ajoutez un fichier JavaScript appelé app.js à votre projet. C'est là que vous allez écrire le code qui gère l'ajout et l'affichage des contacts.

3. Dans app.js, créez une fonction pour ajouter un nouveau contact à la liste. Cette fonction doit récupérer les valeurs des champs de formulaire, les stocker dans un objet JavaScript et ajouter cet objet à une liste de contacts. Ensuite, elle doit convertir la liste de contacts en JSON et la stocker dans le localstorage. Enfin, elle doit appeler une fonction pour afficher la liste de contacts mise à jour dans le tableau HTML.

```
    function addContact() {
        // Récupérer les valeurs des champs de formulaire
```

```
        var name = document.getElementById("name").value;
        var phone = document.getElementById("phone").value;
        var email = document.getElementById("email").value;

        // Créer un objet contact
        var contact = {
            "name": name,
            "phone": phone,
            "email": email
        };

        // Récupérer la liste de contacts existante depuis le localstorage
        var contacts = JSON.parse(localStorage.getItem("contacts")) || [];

        // Ajouter le nouveau contact à la liste
        contacts.push(contact);

        // Stocker la liste mise à jour dans le localstorage
        localStorage.setItem("contacts", JSON.stringify(contacts));

        // Afficher la liste mise à jour dans le tableau HTML
        displayContacts();
    }
```

4. Créez une fonction pour afficher la liste de contacts dans le tableau HTML. Cette fonction doit récupérer la liste de contacts depuis le localstorage, parcourir la liste et générer une ligne de tableau HTML pour chaque contact. Enfin, elle doit insérer ces lignes dans le tableau HTML.

```
function displayContacts() {
    // Récupérer la liste de contacts depuis le localstorage
    var contacts = JSON.parse(localStorage.getItem("contacts")) || [];

    // Générer le HTML pour chaque contact
    var html = "";
    for (var i = 0; i < contacts.length; i++) {
        var contact = contacts[i];
        html += "<tr>";
        html += "<td>" + contact.name + "</td>";
        html += "<td>" + contact.phone + "</td>";
        html += "<td>" + contact.email + "</td>";
        html += "</tr>";
```

```
14          }
15          // Insérer les lignes de tableau HTML dans le tableau HTML
16          document.getElementById("contact-list").innerHTML = html;
17      }
```

5. Ajoutez un écouteur d'événements pour le formulaire de saisie de contact. Ce code doit appeler la fonction 'addContact' lorsqu'un utilisateur soumet le formulaire.

```
1       document.getElementById("add-contact-form").addEventListener("↩
            submit", function(event) {
2           event.preventDefault();
3           addContact();
4       });
```

6. Enfin, appelez la fonction 'displayContacts' lorsque la page se charge pour la première fois, afin d'afficher tous les contacts existants dans le tableau.

```
1       displayContacts();
```

### 3.5.3 Challenge

Voici l'énoncé pour le challenge de l'ajout de fonctionnalités à l'application de gestion de contacts :

1. Ajouter la fonctionnalité de suppression de contacts :
   - Ajouter un bouton "Supprimer" pour chaque contact dans le tableau HTML.
   - Écrire une fonction pour supprimer un contact de la liste et du localstorage lorsque l'utilisateur clique sur le bouton "Supprimer".
2. Ajouter la fonctionnalité de modification de contacts :
   - Ajouter un bouton "Modifier" pour chaque contact dans le tableau HTML.
   - Écrire une fonction pour afficher les informations du contact dans le formulaire d'ajout lorsque l'utilisateur clique sur le bouton "Modifier".
   - Écrire une fonction pour mettre à jour les informations du contact dans la liste et le localstorage lorsque l'utilisateur soumet le formulaire de modification.
3. Ajouter la possibilité de transférer les données vers une base de données MySQL :
   - Écrire un script PHP pour se connecter à une base de données MySQL et insérer les informations de contact.

- Écrire une fonction JavaScript pour récupérer les données de contact du localstorage et les envoyer au script PHP à l'aide d'une requête AJAX.
- Ajouter un bouton "Exporter" pour permettre à l'utilisateur d'exporter les données vers la base de données MySQL.

Le résultat final devrait être une application de gestion de contacts qui permet à l'utilisateur d'ajouter, de modifier, de supprimer et d'exporter des contacts vers une base de données MySQL.