Javascipt

Introduction

- JavaScript permet d'ajouter de l'interactivité aux sites Web
- Code JavaScript est intégré directement dans les pages HTML
- Code interprété par le navigateur côté client, contrairement au PHP, qui est interprété côté serveur.
- JavaScript est un langage événementiel : il permet d'associer des actions aux événements déclenchés par l'utilisateur (comme le survol de la souris, les clics ou la saisie au clavier).

Introduction

- JavaScript est pris en charge par les principaux navigateurs sans nécessiter de plug-in spécifique.
- Il permet un accès direct aux objets d'un document HTML.
- Offre la possibilité de créer des animations légères, sans les temps de chargement prolongés souvent associés aux contenus multimédia.
- JavaScript est un langage relativement sécurisé : il ne peut ni lire ni écrire sur le disque dur de l'utilisateur, évitant ainsi les risques de transmission de virus.

Introduction

- Il y a deux manières principales pour insérer un code **JavaScript** dans une page HTML :

JavaScript intégré directement dans le fichier HTML (avec une balise <script>)

```
<html>
    <head>
        <title>Page HTML</title>
    </head>
    <body>
        <script>
            alert('bonjour');
        </script>
    </body>
</html>
```

JavaScript externe (via un fichier .js séparé)

```
<html>
    <head>
        <title>Page HTML</title>
        <script src="monScript.js">
        </script>
    </head>
    <body>

    </body>
</html>
```

Boîte de dialogue

C'est quoi, une boîte de dialogue ?

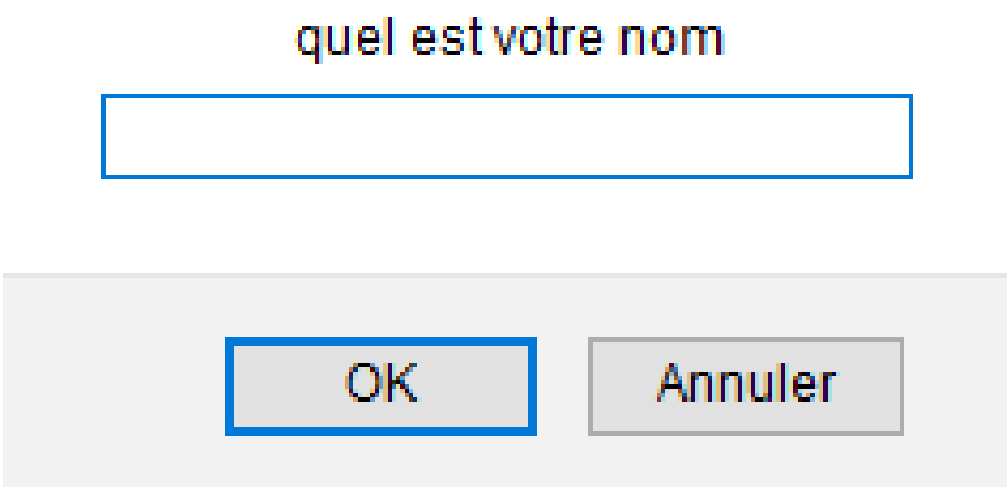
C'est une petite fenêtre (une boîte) qui sert à communiquer (dialoguer) avec l'utilisateur (lui afficher ou lui demander quelque chose). Elle permet :

- D'afficher un message ou une alerte à l'utilisateur.
- De demander une saisie ou une confirmation
- D'interagir directement avec l'utilisateur pour recevoir des données d'entrée et afficher des données de sortie.

- JavaScript permet d'interagir avec les utilisateurs via différentes méthodes pour recevoir des données d'entrées et afficher des données en sortie.

Entrée et sortie de données avec JavaScript

- **Méthode prompt() :** permet d'afficher une boîte de dialogue dans laquelle l'utilisateur peut saisir des données.

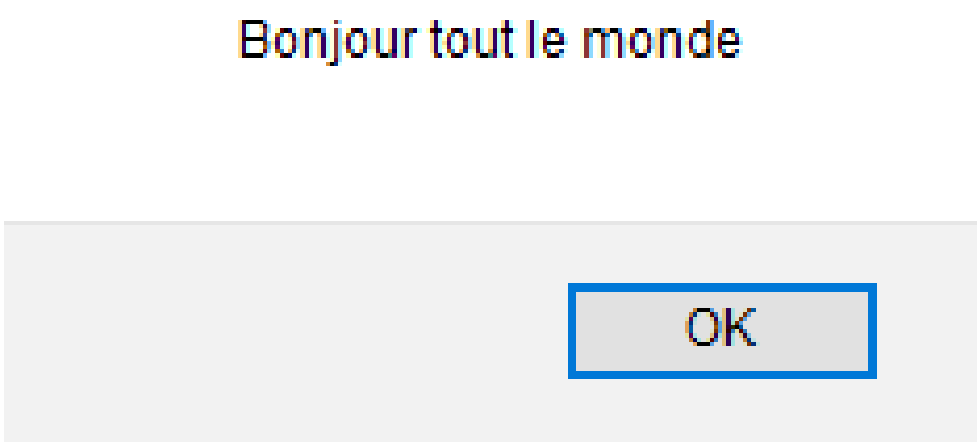


- **Code : JavaScript**

```
let nom = prompt ('Quel est votre nom ?', 'Entrez votre nom ici')
```

Entrée et sortie de données avec JavaScript

- **Méthode alert() :** sert à afficher à l'utilisateur des informations simples de type texte. Une fois que ce dernier a lu le message, il doit cliquer sur OK pour faire disparaître la boîte.

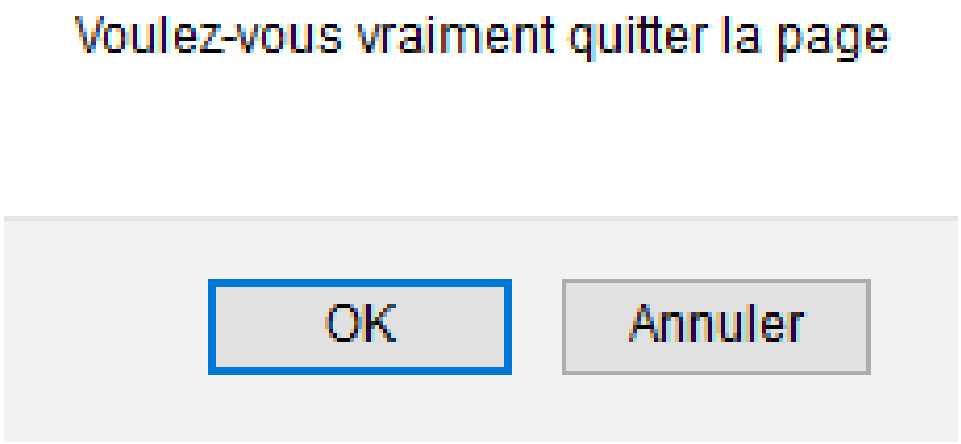


- **Code : JavaScript**

```
alert ( ' Bonjour tout le monde ' )
```

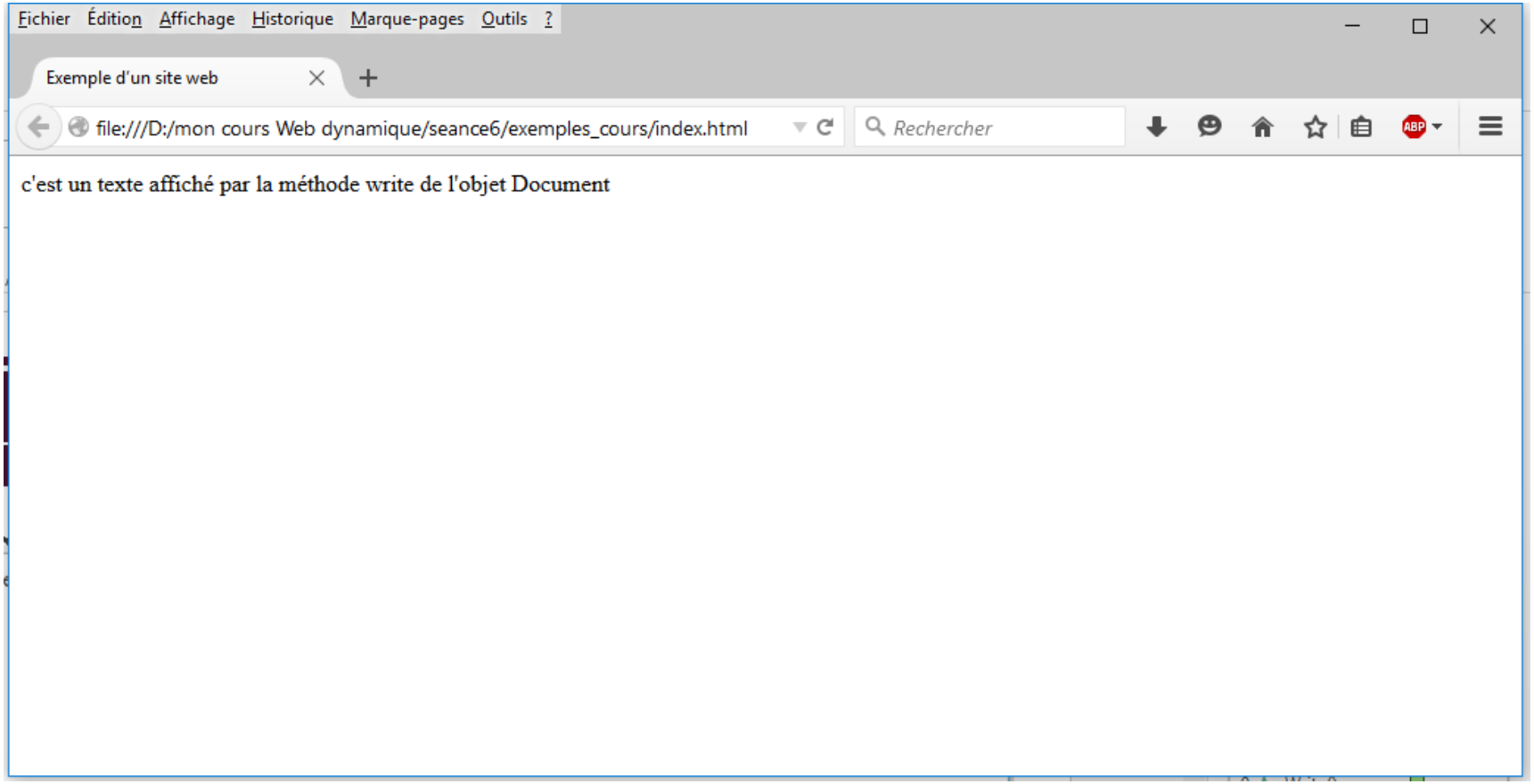
Entrée et sortie de données avec JavaScript

- **Méthode confirm() :** est utilisé pour afficher une boîte de dialogue modale avec un message et deux boutons « OK » et « Annuler ». Elle permet d'obtenir une confirmation de l'utilisateur, par exemple pour valider ou annuler une action.



Entrée et sortie de données avec JavaScript

- **Méthode document.write() :** permet d'écrire du code HTML directement dans le contenu de la page WEB.



Entrée et sortie de données avec JavaScript


```
<html>
    <head>
        <title> une page simple </title>
    </head>
    <body>
        Bonjour
        <script>
        alert('bonjour');
        document.write (prompt('quel est votre nom ?','Indiquer votre
        nom ici'));
        confirm('quel bouton allez-vous choisir ?');
        </script>
    </body>
</html>
```

Identifiants JavaScript

- Un identifiant en JS correspond au nom d'une variable. Ce nom doit être **unique**.
- Ces identifiants peuvent être des noms courts (comme x et y) ou bien des noms plus descriptifs (comme note, total, NomComplet, etc.).

Les règles à respecter lors du choix des noms de variables

- Un identifiant peut contenir des lettres, des chiffres, des traits de soulignement(_) et des signes de dollar ($).
- Un identifiant peut commencer par une lettre, un signe $ ou bien un signe _
- JS est sensible à la casse (y et Y sont considérés comme des variables différentes)
- Un identifiant ne peut pas être un mot réservé du langage

Déclaration de variables

Les variables permettent de stocker des données pour les utiliser et les manipuler dans un programme. JavaScript offre plusieurs façons de déclarer des variables.

- **var** (**ancienne méthode**): Introduit dans les premières versions de JavaScript (1995 à 2015).
- **let** (**méthode moderne**) : Introduit avec ES6(ECMAScript 6 2015), permet de déclarer des variables dont le contenu peut changé.
- **const** : Introduit avec ES6, est utilisé pour déclarer des variables constantes dont la valeur ne peut pas être modifiée après leur initialisation.
- **Déclaration sans mot-clé** (ne rien utilisé) : Peut causer des conflits de variables dans un contexte global.

Déclaration de variables

Utilisation de l'instruction **let variable = valeur;**

- Permet de déclarer une variable dont la valeur peut changer.
- Pas de typage explicite : la détection du typage est automatique selon la valeur affectée par l'interpréteur.

```
let x = 10; // Nombre
let message = "Bonjour"; // Chaîne de caractères
let estVrai = true; //Booléen
```

Porté :

- Déclaration en dehors de fonction → globale
- Déclaration dans une fonction → locale

Déclaration de variables


```
<html>
<head>
        <title>Exemple</title>
</head>
<body>
        <script>
                let nom;
                let texte;
                nom=prompt("quel est votre nom");
                texte="je connais votre nom, c'est "+"<i>"+nom+"</i>";
                document.write(texte);
        </script>
</body>
</html>
```

Structure de controle: if …… else

Les structures de contrôle **if...else** en programmation permettent d'exécuter des blocs de code en fonction de conditions logiques. Elles sont essentielles pour prendre des décisions dans un programme.

Syntaxe :

```
if (condition) {
        // Bloc de code à exécuter si la condition est VRAIE
} else {
        // Bloc de code à exécuter si la condition est FAUSSE
```

```
<body>
    <script>
            let age=prompt("quel est votre age");
            if (age>=18){
                alert("vous etes Majeur...");
            }else{
                alert("vous etes Mineur...");
            }
    </script>
</body>
```

Structure de contrôle : boucles itérative

- Les boucles itératives permettent de répéter un ensemble d'instructions tant qu'une condition est vraie ou pour un nombre spécifique d'itérations.
- Elles sont utilisées pour automatiser des tâches répétitives.
- **Boucle for** : utilisé quand on connaît à l'avance le nombre d'itérations.
- **Boucle while** : utilisée quand on veut répéter tant qu'une condition est vraie
- **Boucle do...while** : comme while, mais garantit qu'au moins une itération sera effectuée

Boucles itérative : **FOR**

Initialisation: Déclaration de la variable de contrôle de boucle.
Condition: La boucle continue tant que cette condition est vraie.
Incrément: Mise à jour de la variable de contrôle à chaque itération.

Syntaxe :

```
for (initialisation; condition; incrément) {
  // Bloc de code à exécuter à chaque itération
}
```

```
<body>
        <script>
                let nb=prompt("Donnez un nombre");
                let somme=0;
                for(let i=1;i<=nb;i++){
                somme+=i;
                }
                alert("La somme des nombres entre 0 est "+ nb +" est "+somme);
        </script>
</body>
```

Boucles itérative : **While**

Syntaxe :

```
while (condition) {
        // Bloc de code à exécuter tant que la condition est vraie
}
```

```
<body>
    <script>
        let nb=prompt("Donnez un nombre");
        let somme=0, i=0;
        while(i<=nb){
        somme+=i; i++;
        }
        alert("la somme des nombres entre 0 est "+nb+" est "+somme);
    </script>
</body>
```

Boucles itérative : do.....while

Syntaxe :

```
do{
        // instruction
} while (condition)
```

```
<body>
    <script>
        let nb=prompt("Donnez un nombre");
        let somme=0, i=0;
        do{
        somme+=i; i++;
        }while(i<=nb);
        alert("la somme des nombres entre 0 est "+nb+" est "+somme);
    </script>
</body>
```

Exercice:

- Écrire un programme JavaScript qui permet de saisir trois entiers au clavier et d'afficher le plus grand d'entre eux. Le programme doit effectuer les actions suivantes :

  - Demander à l'utilisateur de saisir trois **nombres entiers**.
  - Comparer les trois nombres entre eux pour déterminer lequel est le plus grand.
  - Afficher le plus grand nombre à l'utilisateur.
  - Si deux ou trois nombres sont égaux, le programme doit afficher le plus grand nombre parmi ces valeurs égales.

Fonctions:

- Une fonction est définie comme un bloc de code organisé et réutilisable, créé pour effectuer une action unique.
- Le rôle des fonctions est d'offrir une meilleure modularité au code et un haut degré de réutilisation.
- Une fonction JavaScript est exécutée au moment de son appel

```
function nom_fonction ([param1, ...]){
        //Corps de la fonction
        return ...;
}
```

- L'instruction **return** est utilisé pour renvoyer une valeur (souvent calculé) au programme appelant.
- Lorsque l'instruction **return** est atteinte, la fonction arrête son exécution

Fonctions et portée des variables :

- La portée d'une variable détermine dans quelle partie du code cette variable est accessible. Elle dépend de l'endroit où elle est déclarée et du mot-clé utilisé pour la déclarer (var, let, ou const).
- **Variable globale :**
  - Déclaré en dehors de toute fonction ou bloc.
  - Elle peut être utilisée et modifiée n'importe où dans le programme, à la fois à l'intérieur et à l'extérieur des fonctions.
- **Variable locale :**
  - Déclaré à l'intérieur d'une fonction ou d'un bloc
  - Elle n'est accessible que dans cette fonction ou ce bloc, et pas à l'extérieur

Fonctions et protée des variables :

```
<script>
  let nom=" Soukaina";//variable globale
  function afficher(){
            alert("Votre nom est: "+nom);
         }
         alert("Votre nom est: "+nom);
</script>
```

```
<script>
  var nom=" Soukaina"; //variable globale
      function saisir(){
            var nom; //variable locale
            nom=prompt("Quel est votre nom:");
            alert(nom);
            return nom;
      }
      alert("Votre nom est: "+saisir());
      alert("Votre nom est: "+nom);
</script>
```

Fonctions anonymes:

- Une fonction anonyme est une fonction qui n'a pas de nom. Elle est souvent utilisée dans des contextes où elle n'a pas besoin d'être réutilisée.

```
function ([param1, ...]){
   //Corps de la fonction

}
```

Exemple :

```
function ( ) {
            console.log("Ceci est une fonction anonyme !"); }
```

```
let x= function ([param1, ...]){
      //Corps de la fonction
}
```

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

```
let nom_fonction=([param1,...])=>{
            //Corps de la fonction
}
```

```
let addition =  (a, b) => {
return a + b; };
Console.log ( addition (3, 5)); //affiche 8
```

→ Pour que cette fonction s'appelle automatiquement:

```
(()=>{
        //Corps de la fonction
    })();
```

Objets prédéfinis :

- En JavaScript, les objets prédéfinis (ou intégrés) sont des outils puissants fournis par le langage pour simplifier les manipulations courantes des données, des structures ou des calculs. Ces objets sont disponibles dès le démarrage d'un programme JavaScript.

- Plusieurs objets prédéfinis en JavaScript :
  - Math, Date, String, Number, Array, Console, Boolean, Function ou image

- L'opérateur **Typeof** :
  - L'opérateur **Typeof** est utilisé pour déterminer le type de données d'une variable ou d'une valeur en JavaScript. Il renvoie une chaîne de caractères décrivant le type de l'opérande.

```
let p = " Bonjour ";
typeof p; //retourne string
```

Tableau de données (OBJET ARRAY) :

- En JavaScript, un tableau (ou Array) est un objet qui permet de stocker une collection ordonnée de données.
- Chaque élément a un index, qui commence à 0.
- Peut contenir des éléments de différents types (nombres, chaînes, objets, fonctions…).
- Déclaration par l'utilisation de let
- Il est possible de déclarer un tableau sans dimension fixée: Sa taille s'adapte en fonction du contenu
- Création d'un Tableau :

- **Avec des crochets [ ] :**

```
let fruits = ["Pomme", "Banane", "Orange"] ;
```

- **Avec le constructeur Array :**

```
let numeros = new Array ( 1, 2, 3 , 4 ) ;
```

Utilisation de Tableau :

- **La propriété length** : Renvoie la taille du tableau (le nombre d'éléments).
  - **let** fruits = **["Pomme", "Banane"]** ;
  - **console.log(fruits.length)**; **//2**
- **Ajouter ou supprimer des éléments :**
  - **push ( ) :** Ajoute un élément à la fin**.**
  - **fruits.push("Orange"); //fruits = ["Pomme", "Banane", "Orange"];**
  - **pop ( ) :** Supprime le dernier élément.
  - **fruits.pop( ); //fruits = ["Pomme", "Banane "]**
  - **unshift ( ) :** Ajoute un élément au début.
  - **fruits.unshift("Mangue"); //fruits=["Mangue","Pomme","Banane"];**
  - **shift ( ) :** supprime le premier élément.
  - **fruits.shift( ); //fruits = ["Pomme", "Banane "]**

Parcourir un Tableau :

- Utilisation d'une boucle **for** classique :

```
let tableau = [1, 20, 30, 40];
for (let i = 0; i < tableau.length; i++) {
console.log(`Élément à l'indice ${i} : ${tableau[i]}`); }
```

- Utilisation d'une boucle **for ... of** : Permet de parcourir directement les valeurs d'un tableau, sans gérer les indices.

```
let tableau = [10, "a", 9, 12];
for (let valeur of tableau ) {
console.log(`valeur : ${valeur}`); }
```

- Utilisation d'une boucle **for ... in** : Permet de parcourir uniquement les indices définis

```
let tableau = new Array ( 1, "a", 9 );
tableau [200] = 12; //ajout d'un élément à un indice éloigné
for (let i in tableau ) {
console.log(`tableau [${i}] = ${tableau[i]}`); }
```

→La propriété **lenght** inclus les trous(case vide)

Tableau associatifs :

- Un **tableau associatif** est une structure de données qui associe des clés à des valeurs.
- Les indices (clés) d'un tableau associatif peuvent être des chaînes de caractères.

```
let tab = { };
tab["nom"] = "hajji";
tab["prenom"] = "hayat";
tab["age"] = 20;
tab["adresse"] = "Fes";
alert (" Votre nom est : "+tab["nom"] +
        "\n Votre prenom est : "+tab["prenom"] +
        "\n Votre adresse est : "+tab["adresse"]
        );
```

Tableau multidimensionnels:

- Un **tableau multidimensionnel** est un tableau contenant d'autres tableaux en tant qu'éléments.

```
let planning = new Array (2);
planning [0] = new Array("Maths", "Anglais", "Physique " );
planning [1] = new Array("Info", "Français", "Sport" );
console.log("Cours du lundi, 3ème créneau :", planning[0][2]); // Physique
planning[1][1] = "Biologie";
planning[0].push("Chimie");
console.log("Planning complet :");
for (let i = 0; i < planning.length; i++) {
        console.log("Jour", i + 1, ":", planning[i]);
}
```

```
Cours du lundi, 3ème créneau : Physique
Planning complet :
Jour 1 : [ 'Maths', 'Anglais', 'Physique', 'Chimie' ]
Jour 2 : [ 'Info', 'Biologie', 'Sport' ]
```

L'objet Array(1) :

Il y a des méthodes pour manipuler l'objet **Array** :

- **concat ( )** : permet de fusionner deux ou plusieurs tableaux.

Exemple :

```
let fruits1 = ["pomme", "banane"];
let fruits2 = ["orange", "kiwi"];
let allFruits = fruits1.concat(fruits2);
console.log(allFruits);
```

→**Affiche :** ["pomme", "banane", "orange", "kiwi"]

- **join ( )** : converti un tableau en chaîne de caractères.

Exemple :

```
let fruits = ["pomme", "banane", "orange"];
let fruitsString = fruits.join(", ");
console.log(fruitsString);
```

→**Affiche :** "pomme, banane, orange"

L'objet Array(2) :

- **slice(startIndex, endIndex)** : extrait une partie du tableau, commençant à partir de l'indice startIndex et allant jusqu'à (mais sans inclure) l'indice endIndex. Elle peut prendre 2 arguments:
  - **startIndex : l'indice à partir duquel commencer (inclus)**
  - **endIndex : l'indice à partir duquel arrêter (non inclus)**

Exemple :

```
let fruits = ["pomme", "banane", "orange", "kiwi"] ;
let sousFruits = fruits.slice(1, 3) ;
console.log(sousFruits);
```

→**Affiche : ["banane", "orange"]**

- **sort ( )** : trie les éléments d'un tableau.

Exemple :

```
let fruits = ["pomme", "banane", "orange", "kiwi"] ;
fruits.sort ( );
console.log(fruits);
```

→**Affiche : ["banane", "kiwi", "orange", "pomme"]**

L'objet Array(3) :

- **reverse ( )** : inverse l'ordre des éléments dans le tableau

Exemple :

```
let fruits = ["pomme", "banane", "orange", "kiwi"] ;
fruits.reverse;
console.log(fruits);
```

→**Affiche : ["kiwi", "orange", "banane", "pomme"]**

- **indexOf ( )** : renvoie l'indice du premier élément trouvé dans le tableau qui correspond à l'élément passé en argument.

Exemple :

```
let fruits = ["pomme", "banane", "orange", "kiwi"] ;
let indice = fruits.indexOf ("banane");
console.log(indice);
```

→**Affiche : 1**

L'objet String :

L'objet **String** possède de nombreuses méthodes intégrées qui permettent de manipuler les chaînes de caractères.

- La propriété **lenght** renvoie le nombre de caractères dans une chaîne.

Exemple :

```
let str = "Bonjour";
console.log(str.length);
```

→ **Affiche:** 7

Méthodes :

- **Opérations sur les chaînes**
- **Opérations sur les caractères**

L'objet String :

Opérations sur les chaînes :

- **concat(str)** : retourne la chaîne concaténée avec **str**
- **split(str)** : retourne, sous forme de tableau, les portions de la chaînes délimitées par **str**
- **substring(debut,fin)**: extrait une sous-chaîne, depuis la position **debut** (incluse) à **fin** (exclue).
- **substr(debut,i)** : extrait une sous-chaîne, depuis la position **début**, en prenant **i** caractères

```
let phrase = "Bonjour tout le monde";
let phraseConcat = phrase.concat(" et bienvenue !");
console.log(phraseConcat);
let mots = phrase.split(" ");
console.log(mots);
let sousChaine = phrase.substring(8, 12);
console.log(sousChaine);
let sousChaine2 = phrase.substr(8, 4);
console.log(sousChaine2);
```

Concat() :
→ **"Bonjour tout le monde et bienvenue !"**
Split() :
→ **["Bonjour", "tout", "le", "monde"]**
Substring(8,12):
→ **"tout"**
Substr(8, 4) :
→ **"tout"**

L'objet String :

Opérations sur les caractères :

- **charAt(i)** : retourne le iième caractère
- **indexOf(str)** : retourne la position de str dans la chaîne (-1 si elle n'est pas trouvée)
- **lastIndexOf(str)** : idem, mais renvoie la position de la dernière occurrence de str
- **toLowerCase()** : retourne la chaîne en minuscules
- **toUpperCase()** : retourne la chaîne en majuscules

```
let phrase = "Bonjour tout le monde, bonjour encore";
let caractere = phrase.charAt(8);
console.log(caractere);
let indexPremier = phrase.indexOf("bonjour");
console.log(indexPremier);
let indexDernier = phrase.lastIndexOf("bonjour");
console.log(indexDernier);
let minuscules = phrase.toLowerCase();
console.log(minuscules);
let majuscules = phrase.toUpperCase();
console.log(majuscules);
```

charAT() :
→ " t"
indexOf("bonjour") :
→ 23
lastIndexOf("bonjour"):
→ 23

L’objet Math :

L’objet **Math** en JavaScript est un objet intégré qui fournit des propriétés et des méthodes pour effectuer des calculs mathématiques.

Propriétés:

- **E** : renvoie la valeur de la constante d'Euler (~2.718);
- **LN2** : renvoie le logarithme népérien de 2 (~0.693);
- **LN10** : renvoie le logarithme népérien de 10 (~2.302);
- **LOG2E** : renvoie le logarithme en base 2 de e (~1.442);
- **LOG10E** : renvoie le logarithme en base 10 de e (~0.434);
- **PI** : renvoie la valeur du nombre pi (~3.14159);
- **SQRT1_2** : renvoie 1 sur racine carrée de 2 (~0.707);
- **SQRT2** : renvoie la racine carrée de 2 (~1.414);

L’objet Math :

Méthodes :

- **abs( ), exp( ), log(), sin( ), cos( ), tan( ), asin( ), acos( ), atan( ), max( ), min( ), sqrt( )** sont les opérations mathématiques habituelles;
- **atan2( )** : retourne la valeur radian de l'angle entre l'axe des abscisses et un point;
- **ceil( )** : retourne le plus petit entier supérieur à un nombre;
- **floor( )** : retourne le plus grand entier inférieur à un nombre;
- **pow(base, exponent)** : retourne le résultat d'un nombre mis à une certaine puissance;
- **random( )** : retourne un nombre aléatoire entre 0 et 1;
- **round( )** : arrondi un nombre à l'entier le plus proche.

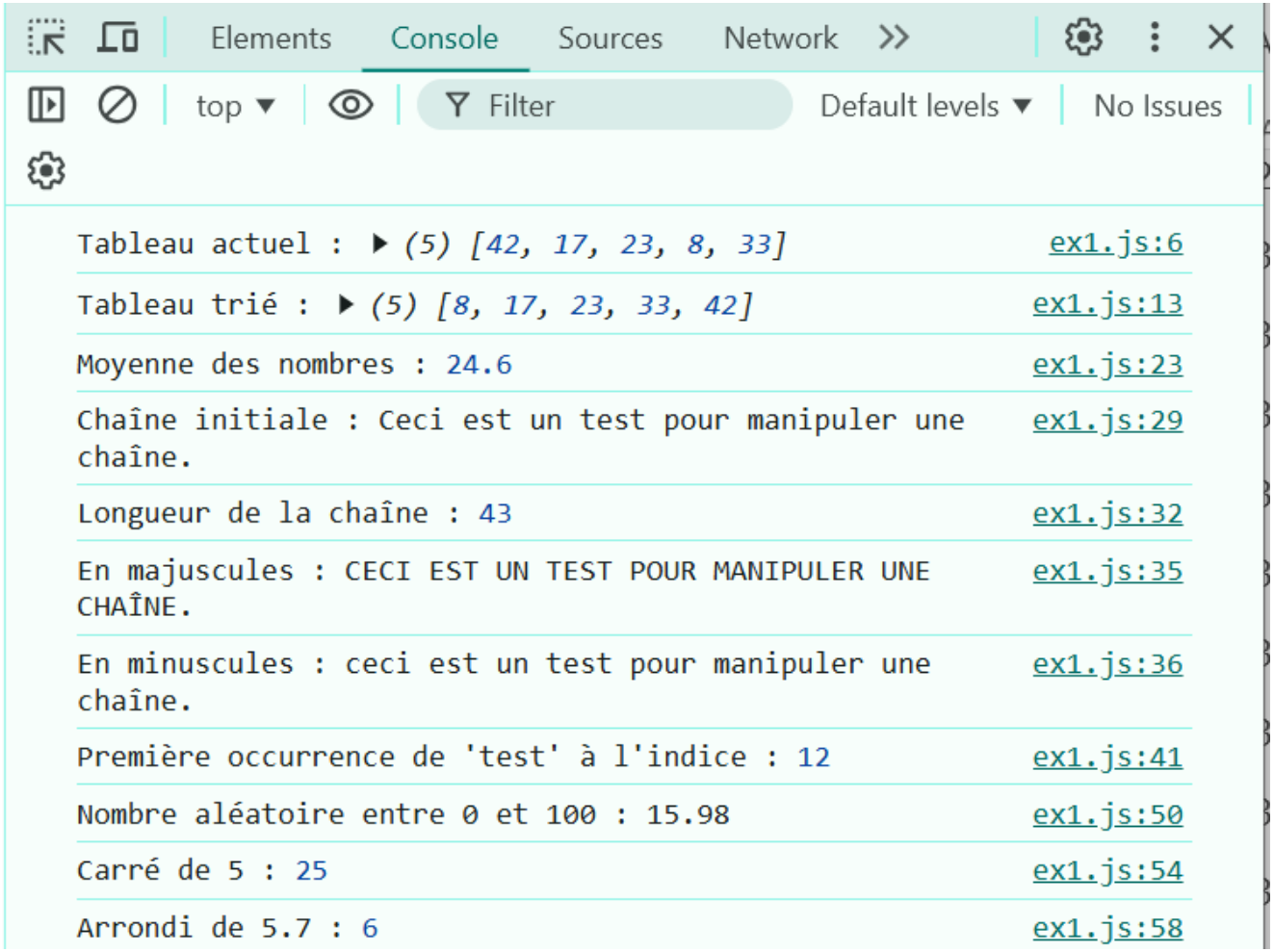
Exercice Récapitulatif:

- Déclarez un tableau nombres contenant 5 nombres au hasard
- Créez une fonction **afficherTableau( )** qui affiche dans la console tous les éléments du tableau nombres.
- Créez une fonction anonyme qui trie le tableau nombres par ordre croissant en utilisant la méthode sort (). Affichez le tableau trié après l'appel de cette fonction.
- Créez une fonction **calculerMoyenne( )** qui calcule la moyenne des nombres dans le tableau nombres.
- Créez une fonction **stringManipulation(chaine)** qui prend une chaîne de caractères en argument et effectue les opérations suivantes :
  - Affichez la longueur de la chaîne.
  - Convertissez la chaîne en majuscule et en minuscule, puis affichez ces résultats.
  - Trouvez la première occurrence du mot "test" dans la chaîne.
- Générer un nombre aléatoire entre 0 et 100 en utilisant la méthode Math.random().
  - Affichez-le dans la console.
  - Affichez le carré de 5 en utilisant Math.pow().
  - Arrondissez un nombre décimal (par exemple 5.7) au nombre entier le plus proche en utilisant Math.round().

Exercice Récapitulatif:

Exercice Récapitulatif

Ouvrez la console pour voir les résultats.



L'objet Image:

L'objet **Image** est utilisé pour manipuler des images dans une page web. C'est une manière pratique de créer, charger et gérer des images dynamiquement en utilisant JavaScript.

- Pour créer un objet image, on utilise le constructeur **new Image( ) .**

Exemple : `let img = new Image ( );`

Propriétés :

- **src :** Définit ou obtient l'URL de l'image.
- **alt** : Définit ou obtient le texte alternatif pour l'image.
- **width** et **height** : Permettent de définir ou d'obtenir la largeur et la hauteur de l'image (en pixels).
- **complete** : Renvoie **true** si l'image est complètement chargée.

Méthodes :

Constructeur :

- **Image ( )** : créer une nouvelle instance de l'objet image sans dimensions.
- **Image (largeur, hauteur)** : créer une nouvelle image avec des dimensions initiales spécifiques (en pixels).

L'objet Date:

Le constructeur **Date** est utilisé pour créer des objets représentant une date et une heure.

Création d'un objet Date :

let now = new Date( ) ; → Renvoie la date et l'heure actuelles.

let specificDate = new Date(2024, 11, 30) ; → 30 Décembre 2024.

- **Les mois vont de 0 (janvier) à 11 (décembre).**

let fromString = new Date("2024-12-01") ; → 1er décembre 2024.

Méthodes :

Obtenir des informations sur une date :

- **getFullYear ()** : renvoie l'année complète.
- **getMonth ()** : renvoie le mois (0=janvier, 11=décembre).
- **getDate ()** : renvoie le jour du mois.
- **getDay ()** : renvoie le jours de la semaine (0=dimanche, 6=samedi).
- **getHours ()** : renvoie l'heure (0-23)
- **getMinutes()** : les minutes(0-59) /**getSeconde()** : les secondes(0-59).
- **getTime() :** obtenir le nombre de millisecondes écoulées depuis le 1er janvier 1970 (UTC).

L’objet Date:

Méthodes :

Modifier une date :

- **setFullYear (année)** : Définit l'année de la date.//Définit les éléments de la date
- **setMonth (mois)** : Définit le mois de la date..
- **setDate (jour)** : Définit le jour du mois de la date.
- **getHours (hours)** : récupère l’heure//récupère //lit
- **getMinutes(minutes)** : récupère les minutes.
- **getSeconds()** : récupère les secondes.

Conversion de la date en chaine de caractères:

- **toString ()** : Renvoie une représentation complète et lisible de la date (jour, mois, année, heure, etc.).
- **toDateString()** : renvoie uniquement la date sous forme lisible.
- **toTimeString()** : renvoie uniquement l’heure.
- **toISOString()** : renvoie la date en format ISO (UTC).
- **toLocaleDateString()** : renvoie la date eu format local.
- **toLocaleTimeString()** : renvoie l’heure au format local.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Exemple - Objet Date</title>
</head>
<body>
    <h1>Exemple de l'objet Date</h1>
    <p>Ouvrez la console pour voir les résultats !</p>
    <!-- Lien vers le fichier JavaScript -->
    <script src="ex1.js"></script>
</body>
</html>
```

```javascript
// Création d'une date spécifique
let myDate = new Date(2023, 10, 30, 14, 45); // 30 novembre 2023, 14:45

// Obtenir des informations
console.log("Année :", myDate.getFullYear());
console.log("Mois :", myDate.getMonth());
console.log("Jour du mois :", myDate.getDate());
console.log("Heure :", myDate.getHours());
console.log("Minutes :", myDate.getMinutes());

// Modifier la date
myDate.setFullYear(2025);
myDate.setMonth(5); // Juin
myDate.setDate(15);
console.log("Date modifiée :", myDate);

// Calculer la différence entre deux dates
let now = new Date();
let diff = myDate - now; // En millisecondes
console.log("Jours jusqu'à la nouvelle date :", diff / (1000 * 60 * 60 * 24));

// Formater la date
console.log("Format lisible :", myDate.toDateString());
console.log("Format ISO :", myDate.toISOString());
```

Exemple de l'objet Date

Ouvrez la console pour voir les résultats !

```javascript
// Création d'une date spécifique
let myDate = new Date(2023, 10, 30, 14, 45); // 30 novembre 2023, 14:45

// Obtenir des informations
console.log("Année :", myDate.getFullYear());
console.log("Mois :", myDate.getMonth());
console.log("Jour du mois :", myDate.getDate());
console.log("Heure :", myDate.getHours());
console.log("Minutes :", myDate.getMinutes());

// Modifier la date
myDate.setFullYear(2025);
myDate.setMonth(5); // Juin
myDate.setDate(15);
console.log("Date modifiée :", myDate);

// Calculer la différence entre deux dates
let now = new Date();
let diff = myDate - now; // En millisecondes
console.log("Jours jusqu'à la nouvelle date :", diff / (1000 * 60 * 60 * 24));

// Formater la date
console.log("Format lisible :", myDate.toDateString());
console.log("Format ISO :", myDate.toISOString());
```

Fonctions supérieures:

Les **fonctions supérieures** permettent d'effectuer des opérations avancées sur des données comme la conversion, la validation, ou l'évaluation de données:

- **eval(chaine)** : évalue une chaîne de caractères comme du codeScript et l'exécute.

```
let x = 10;
let result = eval("x * 2");     //évalue l'expression "x * 2"
console.log(result);  → affiche 20
```

- **isFinite(nombre)** : détermine si la valeur d'un nombre est finie, retourne **true** si la valeur est un nombre fini, sinon **false**.

```
console.log(isFinite(10));          → true
console.log(isFinite(Infinity));  → false
console.log(isFinite(NaN));        →false
```

Fonctions supérieures:

- **isNaN(objet)** : permet de déterminer si la valeur donnée n'est pas un nombre (NaN=Not a Number), retourne true si l'objet n'est pas un nombre.

```
console.log(isNaN(10));       → false(10 est un nombre)
console.log(isNaN( "Hello")); → true
console.log(isNaN(NaN));      →true
```

- **parseFloat(chaine)** : permet de convertir une chaîne de caractère en un nombre à virgule flottante(décimal).

```
console.log(parseFloat(10.5));   → 10.5
console.log(parseFloat("10.5abc")); → 10.5
console.log(parseFloat("abc"));    → NaN
```

- **parseInt(chaine)** : permet de convertir une chaîne de caractère en un nombre entier.

```
console.log(parseInt("10.5"));   → 10
console.log(parseFloat("10abc")); → 10
console.log(parseFloat("abc"));    → NaN
```

Programmation événementielle:

- Programmation événementielle
  - Un paradigme où l'exécution du code dépend d'événements déclenchés par l'utilisateur ou le système (par exemple, un clic, une saisie, ou une action du navigateur).
  - JavaScript utilise des gestionnaires d'événements pour détecter et répondre aux interactions
    - **JavaScript = langage réactif**
    - **L'interaction avec l'utilisateur est gérée via des événements**
- Événements JavaScript :
  - **blur** : le focus est enlevé d'un objet(par exemple, quitter un champ de saisie)
  - **focus** : le focus est donné à un objet (par exemple, cliquer sur un champ)
  - **change** : la valeur d'un champ de formulaire à été modifiée par l'utilisateur
  - **mouseover** : la souris est déplacée sur un objet
  - **click** : un clic souris est déclenché sur un objet
  - **select** : un champ de formulaire est sélectionné (par tabulation)
  - **submit** : un formulaire est soumis
  - **load** : la page est chargée par le navigateur
  - **unload** : l'utilisateur quitte la page

Programmation événementielle:

- Il est possible de baser l'exécution de fonctions sur des événements dans une page web.
- Les événements sont déclenchés par des actions ou des changements dans l'état du navigateur ou de l'utilisateur.

Événement détectables :

- Nom de l'événement précédé de on : onBlur, onChange, onClick, onFocus, onLoad, onMouseover, onSelect, onSubmit,. onUnload

Association événement – action :

- Dans le code HTML, l'événement est attaché à un élément via un attribut.
- L'attribut **événement** est associé à une **fonction** ou **action**.

```
<nom_élément attribut1="valeur1" attribut2="valeur2" événement="action">

<button onClick="envoyer()">Envoyer</button>
```

Déclenchement d'instruction JavaScript:

Exemple :

```
<html>
    <head>
       <title>Exemples de déclenchements</title>
       <script>
            function saluer() {
                          alert("Bonjour tout le monde");
                          }
       </script>
     </head>
     <body onLoad="saluer()">
        <h1 onMouseover="saluer()">Survoler le pointeur pour exécuter
                                        l'événement</h1>
        <form>
            <input type="button" name="bouton"  value="salut" onClick="saluer()">
        </form>
            <h1>Exécution sur protocole javascript:</h1>
            <a href="javascript:saluer()">pour saluer</a>
     </body>
</html>
```

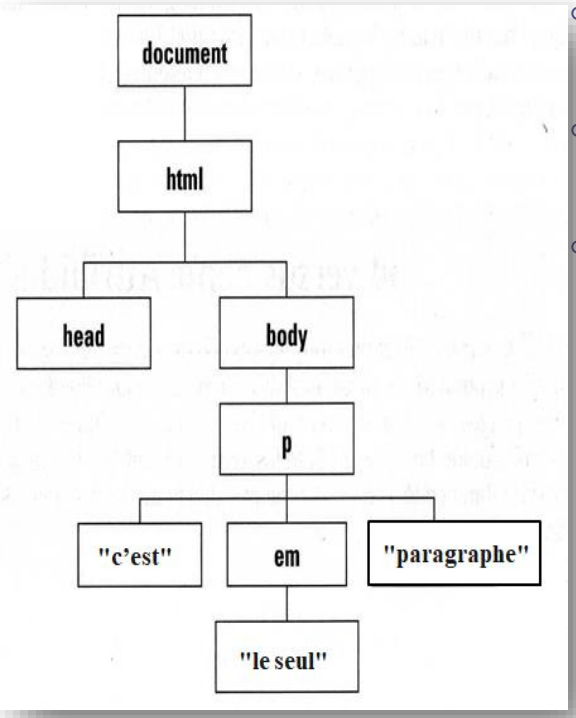
DOM : DOCUMENT OBJECT MODEL:

Qu'est-ce que le DOM ?

- Ensemble d'interfaces normalisées permettant d'exposer la structure d'une page HTML sous forme d'objets organisés en hiérarchie.
- Un objet est un élément HTML
- Le **DOM** agit comme une adresse par laquelle vous pouvez localiser un objet de la page HTML.
- Une fois qu'un objet HTML est récupéré, vous pouvez interagir avec lui en JavaScript pour :
  - Modifier son contenu
  - Changer son style
  - Réagir aux actions de l'utilisateur (comme les clics ou survols)

DOM : DOCUMENT OBJECT MODEL:

Le **DOM** est une structure hiérarchique qui représente les éléments d'une page sous forme d'un arbre.



- Le **DOM** décrit le chemin partant de la fenêtre du navigateur pour descendre jusqu'aux objet de la page Web.
- Le **DOM** est organisé comme un arbre qui reflète la structure imbriquée du code HTML.
- L'arbre contient des nœuds, les nœuds peuvent avoir des fils, et tous les nœuds ont un parent (sauf la racine).

```
<html>
  <head></head>
  <body>
      <p>This is the <em>one and
        only</em> paragraph.</p>
  </body>
</html>
```

Utilisation du DOM:

À l'aide de JavaScript :

- **Modifier le style d'un élément** : On peut sélectionner un élément (<p> par exemple), et modifier sa couleur **(DOM document + DOM element).**
- **Associer un événement à un élément** : On peut sélectionner un élément et lui assigner un événement **(DOM document + DOM events).**
- **Changer les attributs d'un élément** : On peut sélectionner les attributs ("title" par exemple) et changer leur contenu (je remplace title="image2" par title="beau tigre") **(DOM document + DOM attribute).**

Accéder aux élément du DOM « DOCUMENT »

Le DOM « document » fournit plusieurs méthodes pour sélectionner des éléments dans une page HTML, selon différents critères :

- **Sélection par ID** : permet de sélectionner un élément en utilisant son identifiant unique (id).

```
document.getElemntById("Id");
```

- **Sélectionner par attribut name** : retourne une liste de tous les éléments ayant l'attribut **name** spécifié.

```
document.getElemntsByName("Nom");
```

- **Sélection par balise HTML** : retourne une collection de tous les éléments ayant un nom de balise spécifique(ex. div, p, span, etc.).

```
document.getElemntsByTagName("div");
```

- **Sélection par classe** : retourne une collection de tous les éléments ayant un nom de classe donné.

```
document.getElemntsByClassName("NomClasse");
```

DOM « DOCUMENT » : Exemple

Supposons une page HTML contenant ces éléments :

```
<div id="section1" class="contenu">Texte 1</div>

<div id="section2" class="contenu">Texte 2</div>

<input name="champTexte" value="Test">
```

Sélection avec DOM :

```
let elementParId = document.getElementById("section1");
let elementsParName = document.getElementsByName("champTexte");
let elementsParTag = document.getElementsByTagName("div");
let elementsParClasse = document.getElementsByClassName("contenu");
```

Accéder aux élément du DOM « DOCUMENT »

Le **DOM** « document » offre deux méthodes très puissantes pour sélectionner des éléments HTML en utilisant les **sélecteurs CSS** :

- **querySelector(selector)** : **retourne le premier élément** qui correspond au sélecteur CSS spécifié. Si aucun élément ne correspond, elle retourne null. On l'utilise lorsqu'on veut sélectionner un seul élément spécifique.
- **querySelectorAll(selector)** : retourne tous les éléments correspondant au sélecteur CSS sous forme d'une **NodeList**.

```
<div class="contenu" >Texte 1</div>
<div id="section1" class="contenu">Texte 2</div>
<span class="contenu"> Texte 2</ span>
```

```
let premierContenu = document.querySelector(".contenu");

Console.log(premierContenu); //Résultat : <div class="contenu">Texte 1</div>

let Contenus = document.querySelectorAll(".contenu");

let deuxiemeContenu = document.querySelectorAll(".contenu")[1];

Console.log(deuxiemeContenu); //<div id="section1" class="contenu">Texte 2</div>
```

DOM «ELEMENT» & DOM «ATTRIBUTE »

Le **DOM «ELEMNT»** et le **DOM «ATTRIBUTE»** permettent d'interagir dynamiquement avec les éléments HTML d'une page web et de modifier leurs propriétés ou leurs attributs via JavaScript.

- **DOM «ELEMNT»** : représente les éléments HTML (comme , <p>, <img>, etc.) de la page web.
  - Modifier leur contenu: utiliser **innerHTML, textContent.**
  - Changer leurs styles: utiliser la propriété style pour appliquer des styles directement.
  - Gérer les classes CSS : **classList.add()** pour ajouter une classe et **classList.remove()** pour supprimer une classe.

- **DOM «ATTRIBUTE»** : permet de manipuler les attributs des éléments HTML (comme id, class, src, alt, etc.).
  - Récupérer la valeur d'un attribut : **getAttribute()**.
  - Modifier ou ajouter un attribut : **setAttribute()**.
  - Supprimer un attribut : **removeAttribute()**.

DOM «ELEMENT» & DOM «ATTRIBUTE »: exemple


```html
<h2 id="titre" >Recherche</h2>

<input type="search" name="recherche" id ="rechercher" >

<button id="modifier" >Modifier</button>
```

```javascript
// DOM "Element" : Modifier le contenu du <h2>
let titreElement = document.getElementById("titre");
titreElement.textContent = "Nouveau Titre"; //Change le texte du <h2> en
"Nouveau Titre"
// DOM "Attribute" : Modifier et récupérer les attributs de l'input
let inputElement = document.getElementById("rechercher");
// Modifier l'attribut 'name' de l'input
inputElement.setAttribute("name", "nouvelleRecherche");
// Récupérer et afficher l'attribut 'name' dans la console
let inputName = inputElement.getAttribute("name");
console.log("Nom de l'input :", inputName);
```

Modifier le DOM : innerHTML

innerHTML est une propriété en JavaScript utilisée pour accéder ou modifier le contenu HTML d'un élément. Elle permet d'ajouter, remplacer ou supprimer des balises HTML et du texte directement dans un élément sélectionné dans le DOM.

Syntaxe : **element.innerHTML**

- Pour obtenir le contenu HTML d'un élément :

```
let contenu = document.getElementById("monElement").innerHTML;
console.log(contenu); // Affiche le contenu HTML
```

- Pour modifier le contenu HTML d'un élément :

```
document.getElementById("monElement").innerHTML="<p>Nouveau contenu</p>";
```

```
<div id="conteneur">Bonjour !</div>
<button onclick="changerTexte()">Changer le texte</button>
<script>
   function changerTexte() {
          document.getElementById("conteneur").innerHTML = "<p>Nouveau
          texte inséré avec HTML</p>"; }
</script>
```

DOM «EVENTS»

Le **DOM «events»** permet de faire une action lors d'un événement (exemple au clic de la souris).

Principaux événements du DOM:

Événements liés à la souris :

- **click** : un élément est cliqué.
- **dblclick** : un élément est double-cliqué.
- **mouseover** : la souris survole un élément.
- **mouseout** : la souris quitte un élément.

Événements liés au clavier :

- **keydown** : un touche est enfoncé.
- **keyup** : un touche est relâchée.
- **keypress** : une touche est pressés.

Événements de formulaire :

- **submit** : quand un formulaire est envoyé.
- **change** : la valeur d'un champ change.
- **input** : l'utilisateur saisit du texte.

Événements liés au chargement:

- **Load** : quand la page ou une ressource (image, script) est complétement chargée.
- **DOMContentLoad** : quand le document HTML est chargé et analysé

Il existe deux principales méthodes pour ajouter des événements en JavaScript :

→ **Méthode onevent (ancienne)**

→ **Méthode addEventListner (moderne)**

DOM «EVENTS»: : addEventListner

La méthode **addEventListner()** permet effectivement **d'abonner** une **fonction** à un **événement spécifique** sur un élément du DOM. Cela signifie que lorsque l'événement se déclenche, la fonction associée sera exécutée automatiquement.

Syntaxe :

```
objet.addEventListener(eventType, listenerFunction);
```

- **objet** : l'élément ou l'objet sur lequel l'événement est écouté. Cela peut être un élément HTML (button, input, etc.), document ou window.
- **eventType** : une chaîne de caractère qui désigne le type d'événement à écouter:
  - **« click », « mouseover », « keypress », « change », « load »…..**
- **listnerFunction** : la fonction (ou le code à exécuter) qui sera appelée chaque fois que l'événement spécifié de déclenche. Elle peut être une fonction définie précédemment ou une fonction anonyme.

Exemple:

```html
<h2 id="titre">Exemple EventListener</h2>
<p id="message">Passez la souris sur le bouton pour voir l'effet.</p>
<button id="actionButton">Cliquez-moi</button>
```

```javascript
// Écouter le chargement complet de la page
  window.addEventListener('load', function () {
        console.log('La page est complètement chargée.');
        // Sélectionner le bouton et le paragraphe
        const button = document.getElementById('actionButton');
        const message = document.getElementById('message');
        // Ajouter un événement 'mouseover' au bouton
        button.addEventListener('mouseover', function () {
           message.textContent = "Vous survolez le bouton !";
           button.style.backgroundColor = "lightgreen";
        });
        // Ajouter un événement 'mouseout' au bouton
        button.addEventListener('mouseout', function () {
           message.textContent = "Passez la souris sur le bouton pour voir l'effet.";
           button.style.backgroundColor = "";
        });

        // Ajouter un événement 'click' au bouton
        button.addEventListener('click', function () {
           message.textContent = "Vous avez cliqué sur le bouton !";
           message.style.color = "red";
        });
    });
```

Exemple EventListener

Passez la souris sur le bouton pour voir l'effet.

Cliquez-moi

Exemple EventListener

Vous survolez le bouton !

Cliquez-moi

Exemple EventListener

Passez la souris sur le bouton pour voir l'effet.

Cliquez-moi

Exemple EventListener

Vous avez cliqué sur le bouton !

Cliquez-moi