


# Mémento

# JavaScript

---

Version 1.2 (créé le 15/09/2024, modifié le 23/10/2024)



JS

JS (JavaScript) est un langage de programmation côté client qui ajoute de l'interactivité au site web.



**Loric Informatique**

# Table des matières

1. Prise en main.....	5
1.1. Outils nécessaires .....	5
1.2. Où écrire le code JS ? .....	5
1.3. Premier code « Hello World! » .....	5
2. Bases .....	6
2.1. Syntaxe .....	6
2.2. Les variables .....	6
2.2.1. Types de variables .....	6
2.2.2. Opérations sur les variables .....	7
2.3. Commentaires .....	8
2.4. Les tableaux .....	8
2.5. Les objets .....	9
2.6. Conditions.....	10
2.6.1. Opérateurs de comparaison.....	10
2.6.2. Tests de conditions .....	10
2.7. Boucles .....	11
3. Les fonctions .....	13
3.1. Créer une fonction.....	13
3.2. Retourner une valeur de la fonction .....	13
3.3. Faire un appel à la fonction.....	13
3.4. Créer une fonction anonyme .....	14
3.4.1. Manière simple.....	14
3.4.2. Syntaxe fléchée.....	14
4. Les classes.....	14
4.1. Les classes de base.....	14

4.1.1. Créer une classe.....	14
4.1.2. Définir la classe dans un objet.....	15
4.1.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères.....	15
4.1.4. Créer une méthode (dans une classe) .....	15
4.1.5. Créer une méthode statique (dans une classe) .....	15
4.1.6. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable dans une classe.....	15
4.1.7. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable en dehors d'une classe.....	15
4.1.8. Faire un appel d'une méthode ou d'une variable statique en dehors d'une classe.....	16
4.1.9. Créer un getter et un setter.....	16
4.2. Les classes internes.....	16
4.3. L'héritage.....	17
4.3.1. Créer une classe héritée d'une autre classe.....	17
4.3.2. Définir la classe dans un objet.....	17
4.3.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères.....	17
5. Le stockage local.....	18
5.1. Cookies.....	18
5.2. Session Storage.....	18
5.3. Local Storage .....	18
6. Les instructions.....	20
6.1. Instructions de bases .....	20
6.2. Math .....	20
6.3. JSON .....	20
6.4. Exécution asynchrone .....	21
7. Les importations et exportations .....	22

7.1. Exporter une classe (fonctionne aussi avec les fonctions et autres)....	22
7.2. Importer une classe exportée dans un autre fichier .....	22
8. Les éléments HTML (du DOM).....	23
8.1. Récupérer un élément du DOM .....	23
8.2. Modifier un élément du DOM .....	24
8.3. Propriétés sur les éléments.....	24
8.4. Propriétés sur le style des éléments.....	25
8.5. Les événements.....	25
8.5.1. Les différents écouteurs d'événements .....	25
8.5.2. Informations sur les événements .....	26

## 1. Prise en main

### 1.1. Outils nécessaires

- Navigateur Internet à jour (ex : Firefox ou Chrome)
- Un logiciel de codage (Visual Studio Code (recommandé) ou Notepad++)
- Connaissances en HTML et CSS
- Valideur JS : <https://jshint.com/> (faire précéder son code par `// jshint browser:true, eqeqeq:true, undef:true, devel:true, esversion: 6`)



Le memento HTML est disponible en ligne sur le site :

[https://loricaudin.github.io/loric-informatique/mementos/html/memento\\_html.html](https://loricaudin.github.io/loric-informatique/mementos/html/memento_html.html)



Le memento CSS est disponible en ligne sur le site :

[https://loricaudin.github.io/loric-informatique/mementos/css/memento\\_css.html](https://loricaudin.github.io/loric-informatique/mementos/css/memento_css.html)

### 1.2. Où écrire le code JS ?

Soit dans un fichier (ex : *script.js*), mais entre les balises `<body></body>`, il faut rajouter : `<script src="script.js"></script>` au moment où on souhaite appeler le script (ou bien entre les balises `<head></head>`, il faut rajouter : `<script src="script.js" defer></script>` pour l'appeler à la fin du chargement du site).

Soit entre les balises `<script></script>`, placées dans `<body></body>` au moment où on souhaite appeler le script (à éviter).

### 1.3. Premier code « Hello World! »

```
alert("Hello World!");
```

## 2. Bases

### 2.1. Syntaxe

```
instruction1;  
instruction2;  
instruction3;
```

### 2.2. Les variables

#### 2.2.1. Types de variables

Fonction pour connaître le type : `typeof(variable)`

##### 2.2.1.1. Types de base

Type	Description	Fonction pour le convertir
int ou number	Nombre entier	<code>parseInt(variable)</code> ou <code>parseInt(variable, base)</code>
float ou number	Nombre décimal (ex : 0.1)	<code>parseFloat(variable)</code>
string	Chaîne de caractères entre ' ' ou " " (caractère <code>\n</code> : retour à la ligne ; <code>\t</code> : tabulation ; <code>\b</code> : retour en arrière ; <code>\f</code> : nouvelle page)	<code>variable.toString()</code>

### 2.2.1.2. Autres types

Type	Description
boolean	Valeur pouvant être true ou false
object	Autres types (tableau (ou liste), null...)

Il existe 2 types "nulle" : null qui est une valeur vide, et undefined qui est un élément inexistant.

### 2.2.2. Opérations sur les variables

Instruction	Description
1 + 2	Renvoie 3
3 - 1	Renvoie 2
6 * 4	Renvoie 24
5 / 2	Renvoie 2.5
Math.floor(5 / 2)	Renvoie 2 (le quotient sans décimal)
Math.trunc(5 / 2)	Renvoie 2 (le quotient sans décimal), utile si Math.floor() échoue (trop grand nombre)
5 % 2	Renvoie 1 (le reste de la division)
let <i>variable</i> = <i>valeur</i> ;	Déclare une nouvelle variable dont la portée est limitée à celle du bloc dans lequel elle est déclarée
var <i>variable</i> = <i>valeur</i> ;	Déclare une nouvelle variable globale ou locale à une fonction (sans distinction des blocs utilisés dans la fonction)
var <i>a</i> ;	Déclare la variable <i>a</i> sans affecter de valeur (contient null)
<i>a</i> = 5;	Affecte 5 à une variable
<i>a</i> = <i>a</i> + 3;	Ajoute 3 à une variable
<i>a</i> += 3;	Ajoute 3 à une variable

<code>a++;</code>	Ajoute 1 à une variable
<code>b = a++;</code>	Équivalent à : <code>b = a; a = a + 1;</code>
<code>b = ++a;</code>	Équivalent à : <code>a = a + 1; b = a;</code>
<code>const PI = 3.14;</code>	Crée une variable constante (non modifiable)
<code>static variable = valeur;</code>	Déclare une nouvelle variable qui n'est jamais détruite (si déjà exécuté, cette instruction est ignorée)
<code>let texte = 'chaine';</code>	Affecte une chaîne de caractères à une variable (les chaînes peuvent s'additionner avec l'opérateur +)
<code>let texte = "chaine";</code>	Affecte une chaîne de caractères à une variable (les chaînes peuvent s'additionner avec l'opérateur +)

## 2.3. Commentaires

`//Commentaire tenant sur une ligne`

`/*`

Commentaire pouvant être sur une ou plusieurs lignes

`*/`

## 2.4. Les tableaux

Instruction	Description
<code>var tableau = [];</code>	Crée un tableau vide
<code>var tableau = ["Loric", "Informatique", 69];</code>	Crée une liste avec des valeurs (le type de valeurs n'a pas d'importance)
<code>tableau[i]</code>	Renvoie la valeur du tableau à la position <i>i</i> (l'indice de la première valeur est 0), et permet aussi l'écriture d'une autre valeur
<code>tableau.push(valeur);</code>	Ajoute une valeur dans le tableau



<code>tableau.shift();</code>	Enlève et renvoie la première valeur du tableau
<code>tableau.pop();</code>	Enlève et renvoie la dernière valeur du tableau
<code>tableau.splice(indice, n);</code>	Enlève <i>n</i> valeurs à partir de l'indice spécifié
<code>tableau.length</code>	Renvoie la longueur du tableau
<code>tableau.includes(valeur);</code>	Recherche si la valeur est présente dans le tableau
<code>tableau.indexOf(valeur);</code>	Renvoie la position de la valeur recherchée ou -1 si elle n'est pas trouvée
<code>tableau = chaine.split(separateur);</code>	Couper une chaîne de caractères en précisant le séparateur et renvoie un tableau
<code>chaine = tableau.join(separateur);</code>	Convertir un tableau en chaîne de caractères en séparant par un séparateur

## 2.5. Les objets

Instruction	Description
<code>obj = {};</code>	Crée un objet vide
<code>obj = {a: "Loric", b: 2004};</code>	Crée un objet avec des valeurs (le type de valeurs n'a pas d'importance)
<code>obj["a"];</code> ou <code>obj.a;</code>	Récupère la valeur contenue dans une clé
<code>obj["c"] = 3;</code> ou <code>obj.c = 3;</code>	Ajoute 3 à l'objet
<code>delete obj.c;</code>	Supprime une clé de l'objet

## 2.6. Conditions

Une condition renvoie true si elle est respectée et false sinon

### 2.6.1. Opérateurs de comparaison

Condition	Description de ce que vérifie la condition
<code>a == b</code>	<code>a</code> égal à <code>b</code> (seulement en contenu : <code>l="l"</code> )
<code>a === b</code>	<code>a</code> égal à <code>b</code> (en type et en contenu : <code>l≠"l"</code> )
<code>a &lt; b</code>	<code>a</code> strictement inférieur à <code>b</code>
<code>a &gt; b</code>	<code>a</code> strictement supérieur à <code>b</code>
<code>a &lt;= b</code>	<code>a</code> supérieur ou égal à <code>b</code>
<code>a != b</code>	<code>a</code> n'est pas égal à <code>b</code> (seulement en contenu : <code>l="l"</code> )
<code>a !== b</code>	<code>a</code> n'est pas égal à <code>b</code> (en type et en contenu : <code>l≠"l"</code> )
<code>a in b</code>	<code>a</code> est présent dans <code>b</code> (qui peut être un tableau)
<code>  </code>	À mettre entre deux conditions, permet d'avoir une des deux conditions qui doit être vraie
<code>&amp;&amp;</code>	À mettre entre deux conditions, permet d'avoir deux conditions qui doivent être vraies
<code>!condition</code>	Ne doit pas respecter la condition

### 2.6.2. Tests de conditions

Instruction	Description
<pre>if (condition1) {     instruction1; }</pre>	Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i>

<pre>if (condition1) {     instruction1; } else {     instruction2; }</pre>	<p>Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i>, sinon, on exécute <i>instruction2</i></p>
<pre>if (condition1) {     instruction1; } else if (condition2) {     instruction2; } else {     instruction3; }</pre>	<p>Si <i>condition1</i> est vraie, alors on exécute <i>instruction1</i>, sinon, si <i>condition2</i> est vraie, on exécute <i>instruction2</i>, sinon, on exécute <i>instruction3</i></p>
<pre>switch (nombre) {     case a:         instruction1;         break;     case b:     case c:         instruction2;         break;     default:         instruction3; }</pre>	<p>Si <i>nombre</i> == <i>a</i>, alors on exécute <i>instruction1</i>, sinon, si <i>nombre</i> == <i>b</i> ou <i>c</i>, on exécute <i>instruction2</i>, sinon, on exécute <i>instruction3</i></p>
<pre>a = (condition1) ? 1 : 0;</pre>	<p>Si <i>condition1</i> est vraie, <i>a</i> prend la valeur 1, sinon 0.</p>
<pre>try {     instruction1; } catch(erreur) {     instruction2; }</pre>	<p>Si <i>instruction1</i> provoque une erreur, on exécute <i>instruction2</i> (si <code>throw message;</code> est placé dans <i>instruction1</i>, alors <i>instruction2</i> est exécuté avec <code>erreur = message</code>)</p>

## 2.7. Boucles

Instruction	Description
<pre>for (let i = d; i &lt; f; i++) {     instruction1; }</pre>	<p>On répète <math>f-d</math> fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris</p>
<pre>for (let i = d; i &lt; f; i+=p) {     instruction1; }</pre>	<p>On répète <math>(f-d)/p</math> fois l'instruction pour <i>i</i> allant de <i>d</i> compris à <i>f</i> non compris avec pour pas égal à <i>p</i></p>

<code>for (let <i>i</i> in <i>tableau</i>) {     <i>instruction1</i>; }</code>	On parcourt le tableau (ou une chaîne de caractères) pour <i>i</i> allant de 0 à la longueur de tableau
<code>for (let <i>elt</i> of <i>tableau</i>) {     <i>instruction1</i>; }</code>	On parcourt le tableau (ou une chaîne de caractères) pour <i>elt</i> prenant toutes les valeurs du tableau
<code>while (<i>condition</i>) {     <i>instruction1</i>; }</code>	On répète jusqu'à ce que la condition soit fausse (peut ne pas être répété)
<code>do {     <i>instruction1</i>; } while(<i>condition</i>);</code>	On répète jusqu'à ce que la condition soit fausse (est forcément répété une fois)
<code>break;</code>	Permet de sortir d'une boucle sans la terminer (à éviter si possible)

## 3. Les fonctions

Les fonctions peuvent être situées dans le même fichier, mais doivent être définies avant d'être appelées.

### 3.1. Créer une fonction

```
function maFonction(variable1, variable2...) {  
    instructions;  
}
```

### 3.2. Retourner une valeur de la fonction

```
return variable;
```

### 3.3. Faire un appel à la fonction

```
variable = maFonction(valeur1, valeur2...);
```

ou (s'il n'y a pas de variable de retour)

```
maFonction(valeur1, valeur2...);
```

Remarque : il est possible de faire une surcharge de fonctions, c'est-à-dire qu'il est possible de créer deux fonctions identiques avec des paramètres de types différents, ce qui permet à l'interpréteur de choisir la fonction correspondant au type de variables saisies. Idem pour les constructeurs des classes.

Note : Si une valeur n'a pas de valeur renseignée, alors elle prend comme valeur `undefined`

### 3.4. Créer une fonction anonyme

#### 3.4.1. Manière simple

```
function(variable1, variable2...) {  
    instructions;  
}
```

#### 3.4.2. Syntaxe fléchée

```
(variable1, variable2...) => {  
    instructions;  
}
```

Une fonction anonyme peut être affectée comme une variable. Celle-ci devient donc une fonction.

## 4. Les classes

### 4.1. Les classes de base

#### 4.1.1. Créer une classe

```
public class MaClasse {  
  
    static type3 variableStatique3 = valeur3;  
  
    constructor(mVariable1, mVariable2...) {  
        this.variable1 = mVariable1;  
        this.variable2 = mVariable2;  
    }  
}
```

Note : Il est possible de surcharger le constructeur et de créer une variable statique qui reste la même valeur pour tous les objets.

#### 4.1.2. Définir la classe dans un objet

```
var monObjet = new MaClasse(valeur1, valeur2...);
```

#### 4.1.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères

```
toString() {  
    return "message";  
}
```

#### 4.1.4. Créer une méthode (dans une classe)

```
maMethode(variable1, variable2...) {  
    instructions;  
}
```

#### 4.1.5. Créer une méthode statique (dans une classe)

```
static maMethodeStatique(variable1, variable2...) {  
    instructions;  
}
```

#### 4.1.6. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable dans une classe

```
this.maMethode(valeur1, valeur2...)  
this.variable1 = valeur1;
```

#### 4.1.7. Faire un appel d'une méthode autre que le constructeur et toString ou d'une variable en dehors d'une classe

```
monObjet.maMethode(valeur1, valeur2...)  
monObjet.variable1 = valeur1;
```

Il est préférable d'utiliser des méthodes (appelées getter et setter) pour modifier les variables non statiques

#### 4.1.8. Faire un appel d'une méthode ou d'une variable statique en dehors d'une classe

```
MaClasse.maMethodeStatique(valeur1, valeur2...)
MaClasse.variableStatique3 = valeur3;
```

#### 4.1.9. Créer un getter et un setter

```
get variable1() {
    return this.#variable1;
}

set variable1(value) {
    this.#variable1 = value;
}
```

Les getters et les setters peuvent avoir le même nom. Il est cependant préférable de nommer les variables avec au début un #. Les getters et les setters sont par suite appelés comme-ci c'était des variables.

### 4.2. Les classes internes

Les classes internes sont des classes présentes dans une autre classe.

```
class MaClasse {

    class MaClasseInterne {
        ...
    }

}
```

La classe interne est accessible avec `MaClasse.MaClasseInterne`



### 4.3. L'héritage

L'héritage permet à des classes filles de reprendre les mêmes caractéristiques que leur classe mère, et d'ajouter des nouveaux attributs et/ou méthodes qui leur sont propres.

Il est possible de faire un outrepassement d'une méthode de la classe mère, c'est-à-dire de créer une méthode du même nom qu'une méthode de la classe mère pour la remplacer.

Une classe ne peut hériter que d'une seule classe.

#### 4.3.1. Créer une classe héritée d'une autre classe

```
class MaClasseHeritee extends MaClasse {  
  
    public static type6 variableStatique6 = valeur6;  
  
    constructor(mVariable1, mVariable2..., mVariable4, mVariable5...) {  
        super(mVariable1, mVariable2...);  
        this.variable4 = mVariable4;  
        this.variable5 = mVariable5;  
    }  
}
```

La méthode `super()` permet d'appeler le constructeur de la classe initiale.

#### 4.3.2. Définir la classe dans un objet

```
monObjet = new MaClasseHeritee(valeur1, valeur2..., valeur4,  
valeur5...);
```

#### 4.3.3. Ajouter un retour de la classe lors de la conversion en chaîne de caractères

```
toString() {  
    return (super.toString() + "message");  
}
```

## 5. Le stockage local

### 5.1. Cookies

Cookies permet de stocker des variables localement sur l'ordinateur, accessible sur toutes les fenêtres, avec ou sans date d'expiration et avec communication avec un serveur. Sa capacité maximale est de 4 Ko par variables. Les cookies sont créés depuis le serveur web.

Instruction	Description
<code>Document.cookie</code>	Récupérer la liste des cookies

### 5.2. Session Storage

Session Storage permet de stocker des variables localement sur l'ordinateur, expirant à la fermeture de l'onglet et sans communication avec un serveur. Sa capacité maximale est de 5 Mo par variables.

Instruction	Description
<code>sessionStorage.setItem("clé", "valeur");</code>	Créer une variable avec une valeur affectée
<code>variable = sessionStorage.getItem("clé");</code>	Récupérer le contenu d'une variable
<code>sessionStorage.removeItem("clé");</code>	Supprimer une variable
<code>sessionStorage.clear();</code>	Supprimer toutes les variables

### 5.3. Local Storage

Local Storage permet de stocker des variables localement sur l'ordinateur, accessible sur toutes les fenêtres, sans date d'expiration et sans communication avec un serveur. Sa capacité maximale est de 10 Mo par variables.

Instruction	Description
-------------	-------------

<code>window.localStorage</code>	Récupérer toutes les variables stockées dans l'ordinateur localement
<code>localStorage.setItem("clé", "valeur");</code>	Créer une variable avec une valeur affectée
<code>variable = localStorage.getItem("clé");</code>	Récupérer le contenu d'une variable
<code>localStorage.removeItem("clé");</code>	Supprimer une variable
<code>localStorage.clear();</code>	Supprimer toutes les variables

## 6. Les instructions

### 6.1. Instructions de bases

Instruction	Description
<code>console.log("texte");</code>	Affiche un texte dans la console avec un retour à la ligne
<code>console.log(variable);</code> <code>console.log("Valeur : " + variable);</code>	Affiche une variable dans la console
<code>alert(message);</code>	Affiche un message dans une boîte de dialogue
<code>variable = prompt("Entrer une valeur :");</code>	Demande une valeur dans une boîte de dialogue avec le retour dans une variable
<code>variable = confirm("Voulez-vous confirmer ?");</code>	Demande une confirmation dans une boîte de dialogue, renvoyée dans la variable sous forme d'un booléen

### 6.2. Math

Instruction	Description
<code>Math.random();</code>	Créer un nombre aléatoire entre 0 compris et 1 exclu
<code>Math.floor(Math.random() * n);</code>	Créer un nombre entier aléatoire entre 0 et $n$ compris (si fonction importé ci-dessus)

### 6.3. JSON

Instruction	Description
<code>monObjet = JSON.parse(objJSON);</code>	Convertir une chaîne JSON en objet
<code>objJSON = JSON.stringify(monObjet);</code>	Convertir un objet en chaîne JSON

## 6.4. Exécution asynchrone

Instruction	Description
<code>setTimeout(<i>maFonction</i>, <i>temps</i>);</code>	Exécuter en asynchrone une fonction après un certain nombre de millisecondes paramétré
<code>let <i>idInterval</i> = setInterval(<i>maFonction</i>, <i>temps</i>);</code>	Exécuter en asynchrone une fonction de manière répétitive
<code>clearInterval(<i>idInterval</i>);</code>	Arrêter la répétition de la fonction

## 7. Les importations et exportations

Les importations et exportations ne peuvent être effectuées uniquement si l'appel du script dans le HTML soit sous la forme : `<script type="module" src="script.js"></script>`

### 7.1. Exporter une classe (fonctionne aussi avec les fonctions et autres)

```
export { MaClasse1, MaClasse2... };  
ou  
export default MaClasse1;
```

Il est possible de mettre le mot-clé `export` devant `class` pour exporter une classe.

### 7.2. Importer une classe exportée dans un autre fichier

```
import { MaClasse1, MaClasse2... } from "../fichier";  
ou (si c'est une exportation par défaut) :  
import MaClasse1 from "../fichier";
```

Si c'est une exportation par défaut, il est possible de renommer l'objet.

## 8. Les éléments HTML (du DOM)

### 8.1. Récupérer un élément du DOM

Instruction	Description
<pre>let element = document.querySelector("selecteurCSS");</pre>	Récupérer le premier élément respectant le sélecteur CSS
<pre>let listeElements = document.querySelectorAll("selecteurCSS");</pre>	Récupérer les éléments respectant le sélecteur CSS
<pre>let element = document.getElementById("monId");</pre>	Récupérer l'élément ayant pour id = <i>monId</i>
<pre>let listeElements = document.getElementsByClassName("maClasse");</pre>	Récupérer les éléments ayant pour <i>class</i> = <i>maClasse</i>
<pre>let listeElements = document.getElementsByTagName("balise");</pre>	Récupérer les éléments définis par une balise
<pre>let sousElement = element.querySelector("selecteurCSS");</pre>	Récupérer le premier élément respectant le sélecteur CSS
<pre>let listeSousElements = element.querySelectorAll("selecteurCSS");</pre>	Récupérer les éléments depuis un autre élément respectant le sélecteur CSS
<pre>let sousElement = element.getElementById("monId");</pre>	Récupérer l'élément depuis un autre élément ayant pour id = <i>monId</i>
<pre>let listeSousElements = element.getElementsByClassName("maClasse");</pre>	Récupérer les éléments depuis un autre élément ayant pour <i>class</i> = <i>maClasse</i>
<pre>let listeSousElements = element.getElementsByTagName("balise");</pre>	Récupérer les éléments depuis un autre élément définis par une balise

<code>element.childNodes</code>	Récupérer les nœud fils de l'élément
<code>element.children</code>	Récupérer tous les éléments descendants de celui-ci
<code>element.parentNode</code>	Récupérer le parent de l'élément

## 8.2. Modifier un élément du DOM

Instruction	Description
<code>let nouvelElement = document.createElement("balise");</code>	Créer un nouvel élément
<code>let nouvelElement = element.cloneNode(deep);</code>	Créer un clone d'un élément (l'option <i>deep</i> indique par un booléen si on clone également le contenu)
<code>element.insertBefore(nouvelElement);</code>	Insérer avant un autre élément
<code>element.replaceChild(nouvelElement);</code>	Remplacer un élément existant
<code>element.appendChild(nouvelElement);</code>	Ajouter un élément existant à la fin des enfants d'un élément parent
<code>element.removeChild(nouvelElement);</code>	Supprimer un élément existant

## 8.3. Propriétés sur les éléments

Instruction	Description
<code>element.textContent</code>	Récupérer le texte contenu dans l'élément
<code>element.classList.add(maClasse)</code>	Ajouter une classe dans l'élément



<code>element.classList.remove(<i>maClasse</i>)</code>	Retirer une classe dans l'élément
<code>element.classList.contains(<i>maClasse</i>)</code>	Vérifier si l'élément contient cette classe
<code>element.classList.toggle(<i>maClasse</i>)</code>	Si l'élément contient la classe, elle est supprimée, sinon, elle est ajoutée
<code>element.setAttribute(<i>monAttribut</i>, <i>valeur</i>)</code>	Affecter une valeur à un attribut de l'élément (s'il n'existe pas, l'attribut est alors ajouté)
<code>element.removeAttribute(<i>monAttribut</i>)</code>	Supprimer un attribut
<code>element.tagName</code>	Récupérer le nom de la balise

## 8.4. Propriétés sur le style des éléments

Syntaxe : `element.style.monAttribut = "valeur";`

Remarque : Les attributs sous la forme *mon-attribut* en CSS doivent être écrits sous la forme *monAttribut* en JS.

## 8.5. Les événements

### 8.5.1. Les différents écouteurs d'événements

#### 8.5.1.1. Écouteurs simples

Syntaxe : `element.addEventListener("evenement", maFonction);`

Événement	Exécution
"click"	Dès que le bouton de la souris a été cliqué sur l'élément
"change"	Dès que l'élément a changé
"mouseover"	Dès que le bouton de la souris est enfoncé

"mouseout"	Dès que le bouton de la souris est relâché
"keydown"	Dès qu'une touche du clavier a été enfoncée

### 8.5.1.2. Écouteurs dans le code HTML

Syntaxe : `<balise onevenement="instructions;">...`

Événement	Exécution
onclick	Dès que le bouton de la souris a été cliqué sur l'élément
onchange	Dès que l'élément a changé
onmouseover	Dès que le bouton de la souris est enfoncé
onmouseout	Dès que le bouton de la souris est relâché
onkeydown	Dès qu'une touche du clavier a été enfoncée

Note : Il est possible également d'ajouter l'événement directement dans le code JavaScript sous la forme : `element.onevenement = maFonction;`

### 8.5.2. Informations sur les événements

Les fonctions exécutées après un événement doivent obligatoirement avoir en paramètre « e ». Sur cette variable, des informations sur l'événement peuvent être récupérées.

Instruction	Description
<code>e.target</code>	Récupérer l'élément cible
<code>e.currentTarget</code>	Récupérer l'élément qui est