

## Opérateurs arithmétiques

Utilisables avec des nombres entiers, des nombres décimaux et des variables ayant une valeur numérique.

Opérateur	Exemple et commentaire
+	5 + 5; // Vaut 10 (Addition)
-	20 - 7; // Vaut 13 (Soustraction)
*	3 * 4; // Vaut 12 (Multiplication)
/	5 / 2; // Vaut 2.5 (Division)

## Expressions

Plusieurs variables de types différents peuvent faire partie d'une expression. Cette expression peut employer plusieurs opérateurs différents.

```
5 + 2; // Vaut 7
5 + 7 > 7 * 2; // Vaut false
```

## Déclaration et initialisation de variables

Déclarer une variable permet de lui donner un nom et d'y stocker une donnée. L'initialisation est optionnelle et permet de lui affecter une valeur.

```
let nom = "Hugo"; // Déclaration et affectation
let prixTotal = 5 * 3; // Déclaration et affectation
let animal; // Déclaration
```

## Littéraux de gabarits (Template Strings)

Permettent de construire des chaînes de caractères en y insérant les valeurs contenues dans des variables.

```
let nom = "Judith";
let objet = "chaises";
let phrase = `Je suis ${nom} et j'aime les ${objet}`;
// phrase vaut "Je suis Judith et j'aime les chaises"

// On peut même faire des calculs :
let article = "rhododendron";
let qte = 3;
let prix = 3.99;
let phrase = `${qte} ${article} coûtent ${qte * prix} dollars.`;
// phrase vaut "3 rhododendron coûtent 11.97 dollars."
```

## Opérateurs d'affectation

Permettent de modifier la valeur d'une variable (et potentiellement son type de données) à l'aide d'une expression.

Opérateur	Exemple et commentaires
=	maVariable = 25; // Affecte la valeur 25 à maVariable
+=	maVariable += 3; // Équivalent à : maVariable = maVariable + 3;
-=	maVariable -= 4; // Équivalent à : maVariable = maVariable - 4;

## Alerte

Crée une alerte (Pop-up) avec un message personnalisé

```
let nom = "Judith";
alert(`Bonjour ${nom}`);
```

## Message dans la console

Imprime un message dans la console

```
console.log("Voici mon message");
```

## DOM

Le DOM permet à JavaScript d'interagir avec les éléments HTML et le CSS dans une page Web.

Accéder au contenu textuel d'un élément (et le ranger dans une variable, par exemple)

```
let texte = document.querySelector(".classe").textContent;
```

Modifier le contenu textuel d'un élément (Avec = on remplace, avec += on ajoute)

```
document.querySelector(".classe").textContent = "Nouveau texte";
```

Modifier le style d'un élément

```
document.querySelector(".classe").style.propriété = "valeur";
```

Ajouter un écouteur d'événements sur un élément HTML

```
document.querySelector(".classe").addEventListener("type", maFonction);
```

Modifier les classes d'un élément (**add** pour ajouter, **remove** pour retirer et **toggle** pour basculer)

```
document.querySelector(".classe1").classList.add("classe2");
```

Vérifier la présence d'une classe (Résulte en un booléen) pour un élément

```
let maVariable = document.querySelector(".classe1").classList.contains("classe2");  
// maVariable contient true ou false
```

Ajouter (ou modifier) un attribut à un élément

```
document.querySelector(".classe").nomAttribut = "valeur";
```

Obtenir la valeur d'un attribut pour un élément

```
let valeur = document.querySelector(".classe").nomAttribut;
```

Obtenir un tableau d'éléments HTML avec la même classe

```
let elements = document.querySelectorAll(".classe");
```

### Styles

Syntaxe	Ex. de valeur
.color	"red"
.backgroundColor	"blue"
.display	"none" / "block"
.opacity	0.5
.width	"500px"
.height	"200px"

## Variables globales et locales

Une variable globale (déclarée à l'extérieur des fonctions) peut être utilisée n'importe où dans le code.

Une variable locale (déclarée dans une fonction) ne peut être utilisée que dans la fonction en question.

```
let gMaVariableGlobale = "global";

function message() {
    alert(gMaVariableGlobale + " !");
}
```

## currentTarget (l'élément HTML qui a déclenché un événement)

*currentTarget* peut être utilisé pour cibler l'élément HTML même qui a appelé une fonction à la suite du déclenchement d'un événement.

```
function init() {
    document.querySelector(".classe1").addEventListener("click", colorRed); //Événement 1
    document.querySelector(".classe2").addEventListener("click", colorRed); //Événement 2
}

function colorRed(event) {
    event.currentTarget.style.color = "red";
}
```

## Planificateurs

**setTimeout()** : appeler une fonction une seule fois dans x millisecondes.

```
setTimeout(maFonction, 1000);
```

**setInterval()** : appeler une fonction à intervalle régulier toutes les x millisecondes.

```
setInterval(maFonction, 2000);
```

**clearInterval()** : permet de mettre fin à un planificateur, mais exige que le planificateur en question ait été stocké dans une variable au préalable.

```
let monPlanificateur = setInterval(maFonction, 500);
// Appelle ma fonction toutes les 0.5 seconde
clearInterval(monPlanificateur);
// monPlanificateur est arrêté et la fonction ne se répétera plus.
```

## Événements clavier

Permet de détecter l'appui de touches sur le clavier. L'écouteur d'événements clavier doit être créé comme ceci :

```
// Doit être intégré dans la fonction init()
document.addEventListener("keydown", maFonction);

// Fonction qui est appelée lors d'un événement clavier
function maFonction(event) {
    // La variable touche contient une chaîne de caractères correspondant à la touche
    // appuyée. (Ex : "a", "z", "ArrowUp", "ArrowLeft", etc.)
    let touche = event.key;
    // Faire des opérations avec la touche obtenue
}
```

Opérateurs de comparaison

Permettent de comparer deux expressions. Le résultat est toujours un booléen. (true ou false)

Opérateurs	Exemple et commentaire
<code>==</code>	<code>5 + 2 == 7 // Vaut true (Identiques)</code> <code>"salut" == "allo" // Vaut false</code>
<code>!=</code>	<code>5 != 2 + 3 // Vaut false (Différents)</code> <code>"salut" != "allo" // Vaut true</code>
<code>&lt;</code>	<code>5 &lt; 6; // Vaut true (Plus petit)</code>
<code>&lt;=</code>	<code>5 &lt;= 5 // Vaut true (Plus petit ou égal)</code>
<code>&gt;</code>	<code>6 &gt; 5 // Vaut true (Plus grand)</code>
<code>&gt;=</code>	<code>6 &gt;= 6 // Vaut true (Plus grand ou égal)</code>

Instructions conditionnelles

Permettent de rendre conditionnelle l’exécution de blocs d’instructions. La ou les conditions doivent être des expressions qui résultent en une valeur booléenne. (true ou false)

Simple if

```
if (/* condition */) {  
    // Instructions à exécuter  
    // si la condition est vraie  
}
```

if ... else

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

Opérateurs logiques

Combinent plusieurs comparaisons

Opérateur **ET** (Tout doit être true pour donner true)

```
(age >= 8) && (age < 65)
```

Opérateur **OU** (Une seule expression doit être true pour donner true)

```
(age < 12) || (age >= 65)
```

Opérateur **NON** (Inverse le résultat)

```
!(age < 18)
```

if ... else if ... else

```
if (/* condition 1 */) {  
    // Instructions à exécuter  
    // si la condition 1 est vraie  
}  
else if (/* condition 2 */) {  
    // Instructions à exécuter  
    // si la condition 2 est vraie  
    // et que la condition 1 est fausse  
}  
else {  
    // Instructions à exécuter si  
    // les deux conditions sont fausses  
}
```

## Boucles

Permettent de répéter du code.

### Exemple de boucle **while**

```
let i = 1;

while(i < 4){
    console.log(i);
    i += 1;
}

// Cette boucle imprime 1, puis 2, puis 3
// dans la console.
```

### Exemple de boucle **do while**

```
let i = 1;

do{
    console.log(i);
    i += 1;
}while(i < 0);

// Cette boucle imprime 1 dans la console
// et fait une seule itération.
```

## Tableaux

### Créer un tableau

```
let monTableau = [valeur, valeur, valeur, valeur, ...];
```

### Accéder à une donnée d'un tableau

```
monTableau[index] // index est un nombre situé entre 0 et monTableau.length - 1
```

### Obtenir la taille d'un tableau

```
monTableau.length
```

### Ajouter une donnée supplémentaire à la fin d'un tableau

```
monTableau.push(nouvelleValeur);
```

### Modifier une donnée dans un tableau

```
monTableau[index] = nouvelleValeur;
```

### Retirer la dernière donnée d'un tableau

```
monTableau.pop();
```

### Retirer une ou plusieurs données n'importe où dans un tableau

```
monTableau.splice(index, nbRetirer);
```

## Fonctions

Permettent d'exécuter du code encapsulé dans une fonction.

Exemple de déclaration d'une fonction

```
function maFonction(){  
    // Instruction à exécuter  
    // Instruction à exécuter  
}
```

Appeler une fonction pour l'exécuter (Dans la console ou dans une autre fonction)

```
maFonction();
```

## Fonctions avec paramètre(s)

Déclaration de la fonction

```
function maFonction(p1, p2){  
    // Faire quelque chose avec p1 et p2  
}
```

Appel de la fonction

```
maFonction(3, "allo");
```

## Fonctions avec valeur de retour

Déclaration de la fonction

```
function maFonction(){  
    return 3;  
}
```

Appel de la fonction

```
let x = 2 + maFonction(); // x vaut 2 + 3
```

## Fonctions de texte

Diverses fonctions à utiliser avec des chaînes de caractères.

```
let y = "35px";  
parseInt(y) // Retourne 35 (Converti en nombre)  
let z = "-14.32";  
parseFloat(z) // Retourne -14.32 (Converti en nombre à virgule)
```

## Fonctions mathématiques

Diverses fonctions à utiliser avec des nombres.

```
Math.random() // Retourne un nombre aléatoire entre 0 et 0.999999999...
```