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)

Concaténation de chaînes de caractères

Si au moins une donnée est une chaîne de caractères quand on utilise l'opérateur +, on va joindre les données pour former une chaîne de caractères plus longue.

```
let nom = "Sophie";
let message = "Bonjour " + nom + " !";
// message vaut "Bonjour Sophie !"
let mot = "Sal";
mot += "ade";
// mot vaut "Salade"
```

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
```

Expressions

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

```
5 + 2; // Vaut 7
5 * 5 + " fruits"; // Vaut "25 fruits"
5 + 7 > 7 * 2; // Vaut false
```

Conversion d'une chaîne de caractères en valeur numérique

parseInt() : convertit une chaîne de caractères en nombre entier

```
let a = "25";
let b = parseInt(a);
// b contient 25 (et non "25")
```

parseFloat() : convertit une chaîne de caractères en nombre décimal

```
let a = "3.75";
let b = parseFloat(a);
// b contient 3.75 (et non "3.75")
```

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++; // Équivalent à : maVariable = maVariable + 1;`
--	`maVariable--; // Équivalent à : maVariable = maVariable - 1;`
+=	`maVariable += 3; // Équivalent à : maVariable = maVariable + 3;`
-=	`maVariable -= 4; // Équivalent à : maVariable = maVariable - 4;`

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();
```

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("#id").textContent;
```

Modifier le contenu textuel d'un élément (Avec =)

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

Ajouter du texte à un contenu textuel (Avec +=)

```
document.querySelector("#id").textContent += " Texte supplémentaire";
```

Modifier le style d'un élément

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

Ajouter un écouteur d'événements sur un élément HTML

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

Ajouter une classe à un élément

```
document.querySelector("#id").classList.add("maClasse");
```

Retirer une classe à un élément

```
document.querySelector("#id").classList.remove("maClasse");
```

« Basculer » la présence d'une classe pour un élément

```
document.querySelector("#id").classList.toggle("maClasse");
```

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

```
let maVariable = document.querySelector("#id").classList.contains("maClasse");
// maVariable contient true ou false
```

Ajouter (ou modifier) un attribut à un élément

```
document.querySelector("#id").setAttribute("nomAttribut", "valeur");
```

Retirer un attribut à un élément

```
document.querySelector("#id").removeAttribute("nomAttribut");
```

Obtenir la valeur d'un attribut pour un élément

```
let valeur = document.querySelector("#id").getAttribute("nomAttribut");
```

Obtenir un tableau d'éléments HTML avec la même classe

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

Attributs data-

Attribut personnalisé qui peut contenir n'importe quelle valeur sous forme de chaîne de caractères.

```
<img src="images/ex.png" id="image" alt="Une image" data-nom="Chat" data-age="4">
```

Au même titre que n'importe quel attribut, **data-** peut être récupéré ou modifié avec **getAttribute** et **setAttribute**.

Styles

Syntaxe	Ex. de valeur
.color	"red"
.backgroundColor	"blue"
.display	"none" / "block"
.opacity	0.5
.borderColor	"yellow"
.borderWidth	"5px"
.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 + " !");
}
```

Constantes

Une constante est une variable qui est déclarée de manière similaire à une variable, mais avec le mot-clé const. Toutefois, il est interdit de modifier sa valeur après l'avoir initialisée.

```
const MA_CONSTANTE = 50;
MA_CONSTANTE = 40; // Erreur
```

Mot-clé *this*

Le mot-clé *this* représente l'élément HTML qui appelle une fonction à la suite du déclenchement d'un événement.

```
function init(){
    document.querySelector("#id1").addEventListener("click", colorRed); // Événement 1
    document.querySelector("#id2").addEventListener("click", colorRed); // Événement 2
}

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

Opérateurs de comparaison

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

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

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)
```

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
}
```

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

Permet de répéter des instructions

```
for(initialisation ; condition d'exécution ; incrémentation){
    // Instructions
    // Instructions
}
```

Exemple de boucle

```
let a = 0;

for(let index = 1; index < 4; index += 1){
    a += 1;
}

alert(a); // a vaut 3 après la boucle
```

Tableaux

Créer un tableau

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

Accéder à un 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 dans un tableau

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

Ce modèle de boucle permet de parcourir un tableau facilement

```
for(let x of monTableau){
    // Faire quelque chose avec x
}
```

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(evenement){
    // La variable touche contient une chaîne de caractères correspondant à la touche
    // appuyée. (Ex : "a", "z", "ArrowUp", "ArrowLeft", etc.)
    let touche = evenement.key;
    // Faire des opérations avec la touche obtenue
}
```

Fonctions avec paramètre(s)

Déclaration de la fonction

```
function maFonction(param1, param2){
    // Faire quelque chose avec param1 et param2
}
```

Appel de la fonction

```
maFonction(3, "allo"); // Ceci n'est qu'un exemple
```

Fonctions avec valeur de retour

Déclaration de la fonction

```
function maFonction(){
    return 3; // Ceci n'est qu'un exemple
}
```

Appel de la fonction

```
let x = 2 + maFonction(); // Ceci n'est qu'un exemple. x vaut 2 + 3
```