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)
%	`10 % 3; // Vaut 1` (Restant division)

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 initialisation
let prixTotal = 5 * 3; // Déclaration et initialisation
let animal; // Déclaration
```

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 !"
```

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
```

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;`
*=	`maVariable *= 4; // Équivalent à : maVariable = maVariable * 4;`
/=	`MaVariable /= 2; // Équivalent à : maVariable = maVariable / 2;`

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)
```

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

```
maFonction();
```

Événements souris

Certains éléments HTML sont associés à un ou plusieurs événements. Ces événements ont un type / déclencheur ("click", "mouseover" et "mouseout") et appellent une fonction spécifique lorsque déclenchés.

Exemple de création d'un événement

```
document.getElementById("monId").addEventListener("mouseover", maFonction);
```

Description des types :
"click" : Déclenchement lorsque l'élément HTML est cliqué.
"mouseover" : Déclenchement lorsque l'élément HTML est survolé.
"mouseout" : Déclenchement lorsque l'élément HTML n'est plus survolé.

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.getElementById("id1").addEventListener("click", colorRed); // Événement 1
    document.getElementById("id2").addEventListener("click", colorRed); // Événement 2
}

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

Dans la function `colorRed()`, *this* remplace `document.getElementById("id1")` lorsque l'événement 1 appelle la fonction et remplace `document.getElementById("id2")` lorsque l'événement 2 appelle la fonction.

Variables globales et locales

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

```
let gMaVariableGlobale = "Accessible partout";

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

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

Alerte

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

```
let nom = "Judith";
alert("Bonjour " + nom);
```

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.

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++){
    a++;
}

alert("a"); // a vaut 3
```

DOM

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

Accéder au contenu textuel d'un élément

```
let texte = document.getElementById("idQuelconque").textContent;
```

Modifier le contenu textuel d'un élément

```
document.getElementById("idQuelconque").textContent = "Nouveau texte";
```

Modifier le style d'un élément

```
document.getElementById("idQuelconque").style.propriété = "valeur";
```

Ajouter une classe à un élément

```
document.getElementById("idQuelconque").classList.add("maClasse");
```

Retirer une classe à un élément

```
document.getElementById("idQuelconque").classList.remove("maClasse");
```

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

```
document.getElementById("idQuelconque").classList.toggle("maClasse");
```

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

```
let maVariable = document.getElementById("idQuelconque").classList.contains("maClasse");
// maVariable contient true ou false
```

Ajouter / Modifier un attribut à un élément

```
document.getElementById("idQuelconque").setAttribute("nomAttribut", "valeur");
```

Retirer un attribut à un élément

```
document.getElementById("idQuelconque").removeAttribute("nomAttribut");
```

Obtenir la valeur d'un attribut pour un élément

```
let valeur = document.getElementById("idQuelconque").getAttribute("nomAttribut");
```

Tableaux

Donnée qui contient plusieurs données accessibles via un index.

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 et, optionnellement, la récupérer. (Dans une variable, par exemple)

```
let valeurRetiree = monTableau.pop();
```

Ce modèle de boucle permet de parcourir un tableau facilement

```
for(let index = 0 ; index < monTableau.length; index++){
     // Faire quelque chose avec monTableau[index]
}
```

Ajouter et / ou retirer une ou plusieurs données dans un tableau

```
monTableau.splice(index, nbRetirer, elem1, elem2, ...);
// La quantité nbRetirer sont retirés à partir de l'index (inclus). De plus,
// 0 à plusieurs éléments (elem1, elem2, ...) prennent leur place et sont ajoutés.
```

Fonctions avec paramètres

Déclaration d'une fonction avec un ou plusieurs paramètres

```
function maFonction (param1, param2, param3, ...){
     // Instructions se servant de param1, param2 et param3 ...
}
```

Appeler une fonction avec paramètres en lui passant des valeurs

```
maFonction("red", 5);
```

Appeler une fonction avec paramètres en lui passant des variables

```
let couleur = "red";
let nombre = 5;
maFonction(couleur, nombre);
```

Attributs data-

Attribut personnalisé qui peut contenir n'importe quelle valeur sous forme de chaîne de caractères. Cet attribut peut être présent dans n'importe quel élément HTML. (div, p, img, h1, etc.)

```
<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**.

Conversion de chaînes 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")
```

Planificateurs

setTimeout : appeler une fonction une fois dans x millisecondes.

```
setTimeout(maFonction, 1000);
// maFonction() sera appelée dans 1
// seconde.
```

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

```
setInterval(maFonction, 2000);
// maFonction() sera appelée toutes les 2
// secondes
```

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);
// Mon planification est arrêté et la fonction ne se répétera plus.
```

Constantes

Une constante est une variable qui est déclarée de manière similaire à une variable, mais avec le mot-clé const.

```
const MA_CONSTANTE = 50;
```

Toutefois, il est interdit de modifier sa valeur après l'avoir initialisée.

```
MA_CONSTANTE = 40; // Erreur
```

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
}
```

Introduction aux objets

Les objets sont des types de données qui possèdent plusieurs propriétés auxquelles on peut associer des valeurs.

Créer un objet : Méthode 1

```
let monChat = new Chat();
monChat.nom = "Sylvestre";
monChat.age = 6;
monChat.affectueux = true;
```

Créer un objet : Méthode 2

```
let monChat = new Chat("Sylvestre", 6, true);
```

Définition d'une classe (représente un type d'objet qui pourra être créé)
Ici, on peut voir que le type de l'objet est nommé « Chat » et il possède les propriétés nom, age et affectueux.

```
class Chat{
    constructor(nom, age, affectueux){
        this.nom = nom;
        this.age = age;
        this.affectueux = affectueux;
    }
}
```

Accéder à une propriété

```
monObjet.nomPropriete
```

Modifier une propriété

```
monObject.nomPropriete = "nouvelleValeur";
```