

04 Opérateurs

Dans tous les exemples de ce cours, on considère que la valeur initiale de `x` est égale à 11.

Les opérateurs de calcul

| Signe | Nom | Signification | Exemple | Résultat |
|----------------|---------------|----------------------------|--------------------|----------|
| <code>+</code> | Plus | Addition | <code>x + 3</code> | 14 |
| <code>-</code> | Moins | Soustraction | <code>x - 3</code> | 8 |
| <code>*</code> | Multiplié par | Multiplication | <code>x * 2</code> | 22 |
| <code>/</code> | Divisé par | Division | <code>x / 2</code> | 5.5 |
| <code>%</code> | Modulo | Reste de la division | <code>x % 5</code> | 1 |
| <code>=</code> | Egal | Reçoit la valeur de droite | <code>x = 5</code> | 5 |

L'opérateur `+` sert aussi à concaténer des chaînes de caractères.

Les opérateurs de comparaison

| Signe | Nom | Exemple | Résultat |
|--------------------|-------------------|-------------------------|--------------------|
| <code>==</code> | Egal | <code>x == 11</code> | <code>true</code> |
| <code><</code> | Inférieur | <code>x < 11</code> | <code>false</code> |
| <code><=</code> | Inférieur ou égal | <code>x <= 11</code> | <code>true</code> |
| <code>></code> | Supérieur | <code>x >= 11</code> | <code>false</code> |
| <code>>=</code> | Supérieur ou égal | <code>x > 11</code> | <code>true</code> |
| <code>!=</code> | Different | <code>x != 11</code> | <code>false</code> |
| <code>===</code> | Egal strict | <code>x === 11</code> | <code>true</code> |
| <code>!==</code> | Different strict | <code>x !== 11</code> | <code>false</code> |

On confond souvent `=`, `==` et `===`.

Le `=` est un opérateur d'affectation : il sert à donner une valeur à une variable.

Le `==` est un opérateur de comparaison *faible* : il compare les valeurs après une conversion automatique de type, ce qui peut provoquer des résultats inattendus.

Le `===` est un opérateur de comparaison **strict** : il compare la valeur *et* le type, sans conversion. C'est l'opérateur recommandé en JavaScript moderne.

Par exemple : `"1" == 1` vaut `true`, tandis que `"1" === 1` vaut `false`.

Retenir : utilisez presque toujours `==` et `!=` pour éviter les erreurs.

Les opérateurs d'affectation composés

On appelle ainsi les opérateurs qui réalisent un calcul dans lequel une variable intervient des deux côtés du signe `=` (ce sont donc en quelque sorte des opérateurs d'attribution).

Dans les exemples suivants `x` vaut toujours 11 et `y` aura comme valeur 5.

| Signe | Description | Exemple | Signification | Résultat |
|-----------------|-------------|---------------------|------------------------|----------|
| <code>+=</code> | plus égal | <code>x += y</code> | <code>x = x + y</code> | 16 |
| <code>-=</code> | moins égal | <code>x -= y</code> | <code>x = x - y</code> | 6 |

| Signe | Description | Exemple | Signification | Résultat |
|-----------------|----------------|---------------------|------------------------|----------|
| <code>*=</code> | multiplié égal | <code>x *= y</code> | <code>x = x * y</code> | 55 |
| <code>/=</code> | divisé égal | <code>x /= y</code> | <code>x = x / y</code> | 2.2 |

Les opérateurs logiques

Aussi appelés opérateurs booléens, ils servent à vérifier deux ou plusieurs conditions.

| Signe | Nom | Exemple | Signification |
|-------------------------|-----|--------------------------|--|
| <code>&&</code> | et | condition1 && condition2 | condition1 et condition2 |
| <code> </code> | ou | condition1 condition2 | condition1 ou condition2 |
| <code>!</code> | non | <code>!condition</code> | inverse la valeur vraie/fausse de la condition |

Les opérateurs d'incrémantation

Ces opérateurs vont augmenter ou diminuer la valeur de la variable d'une unité. Ce qui sera fort utile, par exemple, pour mettre en place des boucles.

Dans les exemples ci-dessous, `x` vaut 3.

| Signe | Description | Exemple | Signification | Résultat |
|------------------|----------------------------|----------------------|---|----------------------|
| <code>x++</code> | incrémantation (postfixée) | <code>y = x++</code> | équivaut à <code>y = x;</code> puis <code>x = x + 1;</code> | y vaut 3 et x vaut 4 |
| <code>x--</code> | décrémantation (postfixée) | <code>y = x--</code> | équivaut à <code>y = x;</code> puis <code>x = x - 1;</code> | y vaut 3 et x vaut 2 |

La priorité des opérateurs JavaScript

Les opérations s'effectuent dans l'ordre suivant de priorité (du degré de priorité le plus faible au degré de priorité le plus élevé).

Dans le cas d'opérateurs de priorité égale, de gauche à droite.

| Opération | Opérateur |
|------------------------------------|--|
| , | virgule ou séparateur de liste |
| <code>= += -= *= /= %=</code> | affectation |
| <code>? :</code> | opérateur conditionnel |
| <code> </code> | ou logique |
| <code>&&</code> | et logique |
| <code>== !=</code> | égalité, différence |
| <code>==== !==</code> | égalité stricte, différence stricte |
| <code>< <= >= ></code> | relationnel |
| <code>* / %</code> | multiplication, division, modulo |
| <code>! - ++ --</code> | opérateurs unaires (négation logique, signe, incrémantation, décrémantation) |
| <code>()</code> | parenthèses (forcent la priorité) |
| <code>!- ++ --</code> | parenthèses |

Exercice

Soit les variables suivantes :

- `a` qui contient la chaîne de caractères 100
- `b` = 100

- **c** qui contient la valeur 1,00
- **d** booléen qui vaut vrai

A réaliser :

- Affichez "Ceci est une chaîne de caractères :" et concaténez cette chaîne avec la variable **a** pour afficher "Ceci est une chaîne de caractères : 100".
- Appliquez à **b** l'opérateur de décrémentation
- Ajoutez à **c** la valeur de **a**
- Inversez la valeur de **d**