Les opérateurs en profondeur

Dans cette lecture, vous apprendrez à connaître les opérateurs supplémentaires, la précédence des opérateurs et l'associativité des opérateurs. Je vous donnerai également quelques exemples d'opérateurs logiques.

1. Opérateurs supplémentaires

- Opérateur logique ET :**&&**
- Opérateur logique OR :**||**
- Opérateur logique NOT :**!**
- L'opérateur de module :**%**
- L'opérateur d'égalité : **==**
- L'opérateur d'égalité stricte : **===**
- L'opérateur d'inégalité : **!=**
- L'opérateur d'inégalité stricte : **!==**
- L'opérateur d'affectation d'addition :**+=**
- L'opérateur d'affectation de concaténation :**+=**(c'est le même que le précédent - nous y reviendrons plus tard)

L'opérateur logique ET en JavaScript :&&

L'**opérateur logique ET** est, par exemple, utilisé pour confirmer que des comparaisons multiples donneront un résultat vrai.

En JavaScript, cet opérateur se compose de deux symboles d'esperluette réunis :**&&**.

Supposons que vous soyez chargé d'élaborer un code qui vérifie si la variable**currentTime** se trouve entre **9 a.m.** et **5 p.m**. Le code doit consigner **true** si **currentTime > 9et** si**currentTime < 17**.

Voici une solution :

```
1   var currentTime = 10;
2   console.log(currentTime > 9 && currentTime < 17);
```

Comment ce code fonctionne-t-il ?

Tout d'abord, à la première ligne, je définis la variable**currentTime** et je lui assigne la valeur de**10**.

Ensuite, à la ligne 2, je consigne deux comparaisons :

currentTime > 9

currentTime < 17

J'utilise également l'opérateur logique**&&** pour joindre les deux comparaisons.

En fait, mon code est interprété comme suit :

```
1   console.log(10 > 9 && 10 < 17);
```

La comparaison de**10 > 9** renvoie**true**.

De même, la comparaison de**10 < 17** renverra**true**.

Cela signifie que je peux réécrire la ligne deux de ma solution comme suit :

```
1   console.log(true && true);
```

Voici, en substance, comment fonctionne mon code.

Maintenant, la question est de savoir quel sera le résultat de **console.log(true && true)**?

Pour comprendre la réponse, vous devez connaître le comportement de l'opérateur logique**&&**.

L'opérateur logique**&&** renvoie une seule valeur : le booléen**true** ou**false**, selon les règles suivantes :

- Il renvoie**true** si les valeurs à sa droite et à sa gauche sont toutes deux évaluées à**true**
- Il renvoie**false** dans tous les autres cas

En d'autres termes :

console.log(true && true)retournera :**true**

console.log(true && false)donnera :**false**

console.log(false && true)sortira :**false**

console.log(false && false)donnera en sortie :**false**

L'opérateur logique OR en JavaScript :||

L'opérateur logique OR en JavaScript se compose de deux symboles de pipe réunis :**||**.

Il est utilisé lorsque vous souhaitez vérifier si au moins l'une des comparaisons données est égale à**true**.

Considérez la tâche suivante : vous devez écrire un programme en JavaScript qui renverra**true** si la valeur de la variable**currentTime** n'est pas comprise entre**9** et**17**. En d'autres termes, votre code doit renvoyer console.log**true** si la valeur de la variable**currentTime** est inférieure à**9** ou supérieure à**17**.

Voici une solution :

```
1   var currentTime = 7;
2   console.log(currentTime < 9 || currentTime > 17);
```

A la première ligne du code, j'affecte le nombre**7** à la variable**currentTime**.

À la ligne deux, je consigne le résultat de la vérification de l'évaluation de**currentTime < 9** ou de**currentTime > 17** à**true**.

C'est la même chose que ceci :

```
1   var currentTime = 7;
2   console.log(true || false);
```

Voici les règles selon lesquelles l'opérateur**||** évalue des valeurs données :

console.log(true || true)produira :**true**

console.log(true || false)donnera :**true**

console.log(false || true)donnera :**true**

console.log(false || false)sortira :**false**

L'opérateur logique OU renvoie toujours**true**, sauf si les deux côtés sont évalués à**false**. En d'autres termes, pour que l'opérateur logique OU renvoie**false**, les résultats des deux comparaisons**doivent être** faux.

Pour revenir à l'exemple de la vérification de**currentTime < 9** ou**currentTime > 17**, cela est logique : vous n'obtiendrez**false** que si la valeur stockée dans la variable**currentTime** est supérieure à**9** et inférieure à**17**.

L'opérateur logique NOT :!

En JavaScript, le symbole de l'opérateur logique NOT est le point d'exclamation :**!**.

Vous pouvez considérer l'opérateur**!** comme un interrupteur, qui fait basculer la valeur booléenne évaluée de**true** à**false** et de**false** à**true**.

Par exemple, si j'affecte la valeur booléenne de **true** à la variable **petHungry**:

var petHungry = true;

...je peux alors consigner le fait que l'animal n'a plus faim en utilisant l'opérateur **!** pour inverser la valeur booléenne stockée à l'intérieur de la variable **petHungry**, comme suit :

console.log('Feeding the pet'); console.log("Pet is hungry: ", !petHungry); console.log(petHungry);

Voici le résultat du code ci-dessus :

```
1   Pet is hungry: true
2   Feeding the pet
3   Pet is hungry: false
4   true
```

La raison de la modification de la sortie dans la console est que vous avez inversé la valeur stockée dans la variable**petHungry**, de**true** à**false**.

Notez toutefois que le code de la ligne cinq de l'exemple ci-dessus affiche toujours**true**- cela est dû au fait que je n'ai pas réaffecté la valeur de la variable**petHungry**.

Voici comment changer de façon permanente la valeur stockée dans la variable**petHungry** de **true** à **false**:

```
1   var petHungry = true;
2   petHungry = !petHungry;
```

Dans cet exemple, j'affecte d'abord la valeur de**true** à la nouvelle variable**petHungry**. Ensuite, à la ligne deux, j'affecte la valeur opposée,**!true**- lire : pas vrai - à la variable existante**petHungry**.

L'opérateur modulus :%

L'opérateur modulus est un autre opérateur mathématique en JavaScript. Il renvoie le reste de la division.

Pour illustrer son fonctionnement, imaginez qu'un petit restaurant disposant de 4 chaises par table et d'un total de 5 tables reçoive soudainement 22 clients.

Combien de clients ne pourront pas s'asseoir dans le restaurant ?

Vous pouvez utiliser l'opérateur modulus pour résoudre ce problème.

```
1   console.log(22 % 5); // 2
```

Le résultat est**2**, ce qui signifie que lorsque je divise**22** et**5**, j'obtiens**4**, et le reste est**2**, ce qui signifie qu'il y a**2** personnes qui n'ont pas pu s'asseoir dans ce restaurant.

L'opérateur d'égalité,==

L'opérateur d'égalité vérifie si deux valeurs sont égales.

Par exemple, cette comparaison renvoie **true**: **5 == 5**. En effet, il est vrai que 5 est égal à 5.

Voici un exemple de l'opérateur d'égalité qui renvoie **false**: **5 == 6**. En effet, il est vrai que 5 n'est pas égal à 6.

De plus, même si l'une des valeurs comparées est du type nombre et l'autre du type chaîne de caractères, la valeur retournée est toujours **true**: **5 == "5"**.

Cela signifie que l'opérateur d'égalité ne compare que les valeurs, mais pas les types.

L'opérateur d'égalité stricte,===

L'opérateur d'égalité stricte compare à la fois les valeurs et les types de données.

Avec l'opérateur d'égalité stricte, la comparaison de **5 === 5** renvoie toujours **true**. Les valeurs de chaque côté de l'opérateur d'égalité stricte ont la même valeur et le même type. Cependant, la comparaison de **5 == "5"** renvoie maintenant **false**, car les valeurs sont égales, mais le type de données est différent.

L'opérateur d'inégalité,!=

L'opérateur d'inégalité vérifie que deux valeurs ne sont pas identiques, mais il ne vérifie pas la différence de type.

Par exemple, **5 != "5"** renvoie un résultat faux, car il est faux d'affirmer que le nombre 5 n'est pas égal au nombre 5, même si cet autre nombre est de type chaîne de caractères.

L'opérateur d'inégalité stricte!==

Pour que l'opérateur d'inégalité stricte renvoie **false**, les valeurs comparées doivent avoir la même valeur et le même type de données.

Par exemple, **5 !== 5** renvoie **false** car il est faux de dire que le nombre 5 n'a pas la même valeur et le même type de données qu'un autre nombre 5.

Cependant, la comparaison du nombre 5 et de la chaîne 5, à l'aide de l'opérateur d'inégalité stricte, renvoie **true**.

```
1   console.log(5 !== "5")
```

2. Utilisation des opérateurs+ sur les chaînes de caractères et les nombres

Combinaison de deux chaînes de caractères à l'aide de l'opérateur+

L'opérateur**+**, lorsqu'il est utilisé avec des données de type numérique, additionne ces valeurs.

Cependant, l'opérateur**+** est également utilisé pour combiner des chaînes de caractères.

Par exemple :

```
"inter" + "net" // "internet"
"note" + "book" // "notebook"
```

Si l'opérateur**+** est utilisé pour joindre des chaînes de caractères, il est alors appelé opérateur de*concaténation*, et vous direz qu'il est utilisé pour*concaténer des* chaînes de caractères.

Lorsqu'il est utilisé avec des nombres, l'opérateur**+** est l'**opérateur d'addition**, et lorsqu'il est utilisé avec des chaînes de caractères, l'opérateur**+** est l'**opérateur de concaténation**.

Combinaison de chaînes et de nombres à l'aide de l'opérateur+

Mais que se passe-t-il lorsque l'on combine une chaîne de caractères et un nombre à l'aide de l'opérateur**+**?

Voici un exemple :

```
365 + " days" // "365 days"
12 + " months" // "12 months"

```

JavaScript tente de vous aider en convertissant les nombres en chaînes de caractères, puis en**concaténant le nombre et la chaîne de caractères**, ce qui donne**une valeur de chaîne de caractères**.

Le processus de cette conversion "sous le capot" des valeurs en JavaScript est appelé "coercition". JavaScript*transforme* une valeur numérique en une valeur de chaîne de caractères afin de pouvoir exécuter l'opérateur**+** sur des types de données disparates.

Le processus de coercition peut parfois être un peu inattendu.

Considérez l'exemple suivant :

```
1 + "2"
```

Quel sera le résultat de**1 + "2"**?

Notez que la valeur de**1** est de type numérique et que la valeur de**"2"** est de type chaîne de caractères. JavaScript va donc contraindre le nombre**1** à devenir une chaîne de caractères**"1"**, puis la concaténer avec la chaîne de caractères**"2"**, de sorte que le résultat est une chaîne de caractères**"12"**.

L'opérateur d'affectation d'addition,+=

L'opérateur d'affectation additionnel est utilisé lorsque l'on souhaite cumuler les valeurs stockées dans une variable.

Voici un exemple : Vous comptez le nombre d'heures supplémentaires effectuées au cours d'une semaine.

Vous ne devez pas spécifier le type de travail, vous voulez simplement compter le nombre total d'heures.

Vous pourriez coder un programme pour en faire le suivi, comme ceci :

```
var mon = 1;
var tue = 2;
var wed = 1;
var thu = 2;
var fri = 3;
console.log(mon + tue + wed + thu + fri); // 9
```

Vous pouvez simplifier le code ci-dessus en utilisant l'opérateur d'affectation d'addition, comme suit :

```
var overtime = 1;
overtime += 2;
overtime += 1;
overtime += 2;
overtime += 3;
console.log(overtime); // 9
```

L'utilisation de l'opérateur d'affectation d'addition réduit le nombre de lignes de votre code.

L'opérateur d'affectation de concaténation,+=

La syntaxe de cet opérateur est exactement la même que celle de l'opérateur d'affectation d'addition. La différence réside dans le type de données utilisé :

```
var longString = "";
longString += "Once";
longString += " upon";
longString += " a";
longString += " time";
longString += "...";
console.log(longString); // "Once upon a time..."
```

Préséance des opérateurs et associativité

La priorité des opérateurs est un ensemble de règles qui détermine quel opérateur doit être évalué en premier.

Considérez l'exemple suivant :

```
1 * 2 + 3
```

Le résultat du code ci-dessus est 5, car l'opérateur de multiplication a la priorité sur l'opérateur d'addition.

L'associativité des opérateurs détermine la manière dont la priorité fonctionne lorsque le code utilise des opérateurs ayant la même priorité.

Il en existe deux types :

- l'associativité de gauche à droite
- l'associativité de droite à gauche

Par exemple, l'opérateur d'affectation est associatif de droite à gauche, tandis que l'opérateur plus grand que est associatif de gauche à droite :

```
var num = 10; // the value on the right is assigned to the variable name on the left
5 > 4 > 3; // the 5 > 4 is evaluated first (to `true`), then true > 3 is evaluated to `fals
```