

# Présentation des boucles et des opérateurs d'incrémentation et de décrémentation en JavaScript

Cours complet JavaScript
INTRODUCTION AU COURS JAVASCRIPT
1. Introduction au JavaScript
2. L'environnement de travail pour ce cours JavaScript
3. Où écrire le code JavaScript ?
4. Commentaires, indentation et syntaxe de base en JavaScript
LES VARIABLES ET TYPES DE VALEURS JAVASCRIPT
5. Présentation des variables JavaScript
6. Les types de données en JavaScript
7. Présentation des opérateurs arithmétiques et d'affectation JavaScript
8. La concaténation et les littéraux de gabarits en JavaScript
9. Les constantes en JavaScript
LES STRUCTURES DE CONTRÔLE JAVASCRIPT
10. Structures de contrôle, conditions et opérateurs de comparaison JavaScript
11. Les conditions if, if...else et if...else if...else en JavaScript
12. Opérateurs logiques, précedence et règles d'associativité des opérateurs en JavaScript
13. Utiliser l'opérateur ternaire pour écrire des conditions JavaScript condensées
14. L'instruction switch en JavaScript
15. Présentation des boucles et des opérateurs d'incrémentation et de décrémentation en JavaScript
16. Les boucles while, do... while, for et for... in et les instructions break et continue en JavaScript
LES FONCTIONS EN JAVASCRIPT
17. Présentation des fonctions JavaScript
18. Portée des variables et valeurs de retour des fonctions en JavaScript
19. Fonctions anonymes, auto-invoquées et récursives en JavaScript
L'ORIENTÉ OBJET EN JAVASCRIPT
20. Introduction à l'orienté objet en JavaScript
21. Création d'un objet JavaScript littéral et manipulation de ses membres
22. Définition et création d'un constructeur d'objets en JavaScript
23. Constructeur Object, prototype et héritage en JavaScript
24. Les classes en JavaScript
VALEURS PRIMITIVES ET OBJETS GLOBAUX JAVASCRIPT
25. Valeurs primitives et objets prédéfinis en JavaScript
26. L'objet global JavaScript String, propriétés et méthodes
27. L'objet global JavaScript Number, propriétés et méthodes
28. L'objet global JavaScript Math, propriétés et méthodes
29. Les tableaux en JavaScript et l'objet global Array
30. Les dates en JavaScript et l'objet global Date
MANIPULATION DU BOM EN JAVASCRIPT
31. JavaScript API, Browser Object Model et interface Window
32. L'interface et l'objet Navigator et la géolocalisation en JavaScript
33. L'interface et l'objet History en JavaScript
34. L'interface et l'objet Location en JavaScript
35. L'interface et l'objet Screen en JavaScript
MANIPULATION DU DOM EN JAVASCRIPT
36. Présentation du DOM HTML et de ses APIs accessibles en JavaScript
37. Accéder aux éléments dans un document avec JavaScript et modifier leur contenu
38. Naviguer ou se déplacer dans le DOM en JavaScript grâce aux noeuds
39. Ajouter, modifier ou supprimer des éléments du DOM avec JavaScript
40. Manipuler les attributs et les styles des éléments via le DOM en JavaScript
41. La gestion d'événements en JavaScript et la méthode addEventListener
42. La propagation des événements en JavaScript
43. Empêcher un événement de se propager et annuler son comportement par défaut en JavaScript
UTILISATION DES EXPRESSIONS RÉGULIÈRES EN JAVASCRIPT
44. Introduction aux expressions régulières ou expressions rationnelles en JavaScript
45. Utiliser les expressions régulières pour effectuer des recherches et remplacements en JavaScript
46. Les classes de caractères et classes abrégées des expressions régulières JavaScript
47. Les métacaractères point, alternatives, ancrs et quantificateurs des expressions régulières JavaScript
48. Créer des sous masques et des assertions dans les expressions régulières JavaScript
49. Les drapeaux, options ou marqueurs des expressions régulières JavaScript
NOTIONS AVANCÉES SUR LES FONCTIONS JAVASCRIPT
50. Paramètres du reste et opérateur de décomposition des fonctions JavaScript
51. Les fonctions fléchées JavaScript
52. Les closures en JavaScript
53. Gestion du délai d'exécution en JavaScript avec setTimeout() et setInterval()
GESTION DES ERREURS ET MODE STRICT EN JAVASCRIPT
54. Gestion des erreurs en JavaScript
55. Le mode strict en JavaScript
L'ASYNCHRONE EN JAVASCRIPT
56. Introduction à l'asynchrone en JavaScript
57. Les promesses en JavaScript
58. Utiliser async et await pour créer des promesses plus lisibles en JavaScript
59. Le chemin critique du rendu et les attributs HTML async et defer
SYMBLES, ITÉRATEURS ET GÉNÉRATEURS EN JAVASCRIPT
60. Les symboles et l'objet Symbol en JavaScript
61. Les protocoles et objets Iterable et Itérateur en JavaScript
62. Les générateurs en Javascript
STOCKAGE DE DONNÉES DANS LE NAVIGATEUR EN JAVASCRIPT
63. Les cookies en JavaScript
64. L'API Web Storage : localStorage et sessionStorage en JavaScript
65. Utiliser l'API de stockage IndexedDB en JavaScript
L'ÉLÉMENT HTML CANVAS ET L'API CANVAS
66. Présentation de l'élément HTML canvas et de l'API Canvas
67. Dessiner des rectangles dans un élément HTML canvas en Javascript
68. Définir des tracés pour dessiner des formes dans un canevas en JavaScript
69. Création de dégradés ou de motifs dans un canevas en JavaScript
70. Ajout d'ombres et utilisation de la transparence dans un canevas en JavaScript
71. Ajouter du texte ou une image dans un canevas en JavaScript
72. Appliquer des transformations sur un canevas en JavaScript
LES MODULES JAVASCRIPT
73. Les modules JavaScript : import et export
JSON, AJAX ET FETCH EN JAVASCRIPT
74. Présentation de JSON et utilisation en JavaScript
75. Introduction à l'Ajax en JavaScript
76. Créer des requêtes Ajax en utilisant l'objet XMLHttpRequest
77. Présentation et utilisation de l'API Fetch en Javascript
CONCLUSION DU COURS COMPLET JAVASCRIPT
78. Conclusion du cours complet

Les boucles sont, avec les conditions, l’une des structures de contrôle de base du JavaScript. Nous allons beaucoup les utiliser et il convient donc de les connaître et de comprendre comment elles fonctionnent.

## Présentation des boucles

Les boucles vont nous permettre d’exécuter plusieurs fois un bloc de code, c’est-à-dire d’exécuter un code « en boucle » tant qu’une condition donnée est vérifiée et donc ainsi nous faire gagner beaucoup de temps dans l’écriture de nos scripts.

Lorsqu’on code, on va en effet souvent devoir exécuter plusieurs fois un même code. Utiliser une boucle nous permet de n’écrire le code qu’on doit exécuter plusieurs fois qu’une seule fois.

Nous disposons de six boucles différentes en JavaScript :

- La boucle **while** (« tant que ») ;
- La boucle **do... while** (« faire... tant que ») ;
- La boucle **for** (« pour ») ;
- La boucle **for... in** (« pour... dans ») ;
- La boucle **for... of** (« pour... parmi ») ;
- La boucle **for await... of** (« pour -en attente-... parmi »).

Le fonctionnement général des boucles est toujours le même : on pose une condition qui sera généralement liée à la valeur d’une variable et on exécute le code de la boucle « en boucle » tant que la condition est vérifiée.

Pour éviter de rester bloqué à l’infini dans une boucle, vous pouvez donc déjà noter qu’il faudra que la condition donnée soit fausse à un moment donné (pour pouvoir sortir de la boucle).

Pour que notre condition devienne fausse à un moment, on pourra par exemple incrémenter ou décrémenter la valeur de notre variable à chaque nouveau passage dans la boucle (ou modifier la valeur de notre variable selon un certain schéma).

Les boucles vont donc être essentiellement composées de trois choses :

- Une valeur de départ qui va nous servir à initialiser notre boucle et nous servir de compteur ;
- Un test ou une condition de sortie qui précise le critère de sortie de la boucle ;
- Un itérateur qui va modifier la valeur de départ de la boucle à chaque nouveau passage jusqu’au moment où la condition de sortie est vérifiée. Bien souvent, on incrémentera la valeur de départ.

## Les opérateurs d'incrémentation et de décrémentation

Incrémenter une valeur signifie ajouter 1 à cette valeur tandis que décrémenter signifie enlever 1.

Les opérations d’incrémentation et de décrémentation vont principalement être utilisées avec les boucles en JavaScript. Elles vont pouvoir être réalisées grâce aux opérateurs d’incrémentaion **++** et de décrémentaion **--**.

Retenez déjà qu’il y a deux façons d’incrémenter ou de décrémenter une variable : on peut soit incrémenter / décrémenter la valeur de la variable puis retourner la valeur de la variable incrémentée ou décrémentée (on parle alors de pré-incrémentation et de pré-décrémentation), soit retourner la valeur de la variable avant incrémentation ou décrémentation puis ensuite l’incrémenter ou la décrémenter (on parle alors de post-incrémentation et de post-décrémentation).

Cette différence d’ordre de traitement des opérations va influencer sur le résultat de nombreux codes et notamment lorsqu’on voudra en même temps incrémenter ou décrémenter la valeur d’une variable et l’afficher ou la manipuler d’une quelconque façon. Tenez-en donc bien compte à chaque fois que vous utilisez les opérateurs d’incrémentaion ou de décrémentaion.

Le tableau ci-dessous présente les différentes façons d’utiliser les opérateurs d’incrémentaion et de décrémentaion avec une variable **let x** ainsi que le résultat associé :

Exemple (opérateur variable)	+ Résultat
<b>++x</b>	Pré-incrémentation : incrémente la valeur contenue dans la variable x, puis retourne la valeur incrémentée
<b>x++</b>	Post-incrémentation : retourne la valeur contenue dans x avant incrémentaion, puis incrémente la valeur de \$x
<b>--x</b>	Pré-décrémentation : décrémente la valeur contenue dans la variable x, puis retourne la valeur décrémentée
<b>x--</b>	Post-décrémentation : retourne la valeur contenue dans x avant décrémentaion, puis décrémente la valeur de \$x

Prenons immédiatement un exemple concret pour illustrer les différences entre pré et post incrémentaion ou décrémentaion.

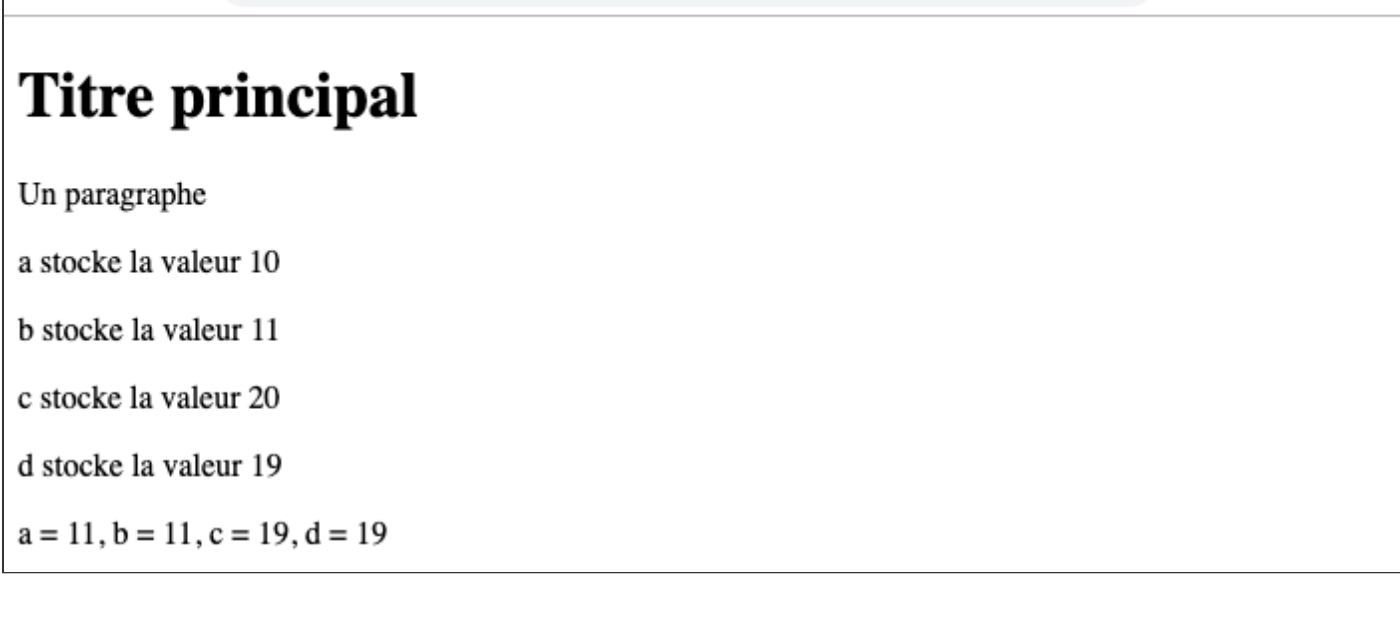
```
<!DOCTYPE html>
<html>
  <head>
    <title>Cours JavaScript</title>
    <meta charset="utf-8">
    <meta name="viewport"
      content="width=device-width, initial-scale=1, user-scalable=no">
    <link rel="stylesheet" href="cours.css">
    <script src="cours.js" async></script>
  </head>

  <body>
    <h1>Titre principal</h1>
    <p>Un paragraphe</p>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
    <p id="p4"></p>
    <p id="p5"></p>
  </body>
</html>
```

```
//On déclare et initialise nos variables sur la même ligne
let a = 10, b = 10, c = 20, d = 20;

//On incrémente / décrémente et affecte le résultat dans un paragraphe.
/*Attention : le premier "+" est un opérateur de concaténation */
document.getElementById("p1").innerHTML = 'a stocke la valeur ' + a++;
document.getElementById("p2").innerHTML = 'b stocke la valeur ' + ++b;
document.getElementById("p3").innerHTML = 'c stocke la valeur ' + c--;
document.getElementById("p4").innerHTML = 'd stocke la valeur ' + --d;

//On affiche ensuite à nouveau le contenu de nos variables
document.getElementById("p5").innerHTML =
'a = ' + a + ', b = ' + b + ', c = ' + c + ', d = ' + d;
```



Il y a plusieurs choses qu’on n’a jamais vues dans ce code. Tout d’abord, vous pouvez constater que j’ai déclaré 4 variables sur la même ligne en utilisant une seule fois le mot **let**. Cette écriture est tout à fait autorisée tant que les différentes variables sont séparées par une virgule et sont bien déclarées sur une seule et même ligne.

Notez que cette syntaxe a des avantages et des inconvénients : elle est un peu plus rapide à écrire, notamment lorsqu’on a beaucoup de variables à déclarer mais est un peu moins claire que la syntaxe de déclaration complète des variables.

Ensuite, on post-incrémente notre variable **let a**, pré-incrémente notre variable **let b**, post-décrémente v notre variable **let c** et pré-décrémente notre variable **let d** puis on affiche le résultat précédé du texte « a/b/c/d stocke la valeur » dans les paragraphes portant les id « p1 », « p2 », « p3 » et « p4 » dans notre fichier HTML appelant le fichier JavaScript.

Notez bien ici que le premier signe **+** entre le texte et les opérations d’incrémentaion ou de décrémentaion est un opérateur de concaténation : il sert à juxtaposer le texte à gauche et la valeur de la variable à droite.


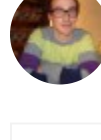
Dans chaque ligne de ce code, on fait donc deux opérations en même temps : on incrémente ou décrémente et on place le résultat dans un paragraphe. Comme vous pouvez le voir, lorsqu’on pré-incrémente ou pré-décérmente, la valeur renvoyée est bien la valeur de base de la variable +/- 1.

En revanche, lorsqu’on post-incrémente ou post-décérmente, la valeur renvoyée est la valeur de base de la variable. Cela est dû au fait que la valeur de base de la variable est ici renvoyée avant l’incrémentaion ou la décrémentaion. Si on affiche plus tard la valeur de nos variables, on peut voir qu’elles ont bien été incrémentées ou décrémentées comme les autres.

[Précédent](#)

[Suivant](#)

## 2 réflexions au sujet de “Présentation des boucles et des opérateurs d'incrémentation et de décrémentation en JavaScript”

 <div><b>claire08</b> 12 septembre 2019 à 18 h 57 min</div>	
	<div>Bonjour,</div> <div>Au début, n’est-ce pas de 6 boucles dont vous parlez et non 4 ? Merci pour vos cours!</div>
	<div>Connectez-vous pour répondre</div>
 <div><b>Pierre GIRAUD</b> 13 septembre 2019 à 11 h 33 min</div>	
	<div>Bonjour,</div> <div>En effet, c’est corrigé, merci !</div>
	<div>Connectez-vous pour répondre</div>

### Laisser un commentaire

Vous devez vous connecter pour publier un commentaire.