Cette page a été traduite à partir de l'anglais par la communauté. Vous pouvez également contribuer en rejoignant la communauté francophone sur MDN Web Docs.

var

L'instruction **`var`** (pour variable) permet de déclarer une variable et éventuellement d'initialiser sa valeur.

JavaScript Demo: Statement - Var

```
var x = 1;

if (x === 1) {
  var x = 2;

  console.log(x);
  // expected output: 2
}

console.log(x);
// expected output: 2

```

Run ›

Reset

Syntaxe

```
var nomVar1 [= valeur1] [, nomVar2 [= valeur2] ... [, nomVarN [= valeurN]]];
```

`nomvarN`

Le nom de la variable, cela peut être n'importe quel identifiant valide.

`valeurN`

La valeur initiale à affecter à la variable, cela peut être n'importe quelle expression valide. S'il n'y a aucune valeur fournie, la variable vaudra `undefined`.

Description

Les déclarations de variables sont traitées avant que le code soit exécuté, quel que soit leur emplacement dans le code. La portée d'une variable déclarée avec `var` est le *contexte d'exécution* courant, c'est-à-dire : **la fonction** qui contient la déclaration **ou le contexte global** si la variable est déclarée en dehors de toute fonction.

Si on affecte une valeur à une variable qui n'a pas été déclarée (le mot-clé `var` n'a pas été utilisé), cela devient une variable globale (une propriété de l'objet global) lorsque l'affectation est exécutée. Les différences entre les variables déclarées et les variables non-déclarées sont :

1. Les variables déclarées sont contraintes dans le contexte d'exécution dans lequel elles sont déclarées. Les variables non-déclarées sont toujours globales.

```
function x() {
  y = 1;   // Lève une exception ReferenceError en mode strict
  var z = 2;
}

x();

console.log(y); // Affiche "1" dans la console
console.log(z); // Lève une exception ReferenceError:
                // z n'est pas définie en dehors de x
```

2. Les variables déclarées sont créées avant que n'importe quel autre code soit exécuté. Les variables non-déclarées n'existent pas tant que leur code n'est pas exécuté.

```
console.log(a);               // Lève une exception ReferenceError
console.log('on continue...'); // N'est jamais exécuté
```

```
var a;
console.log(a);               // Affiche "undefined".
console.log('on continue...'); // Affiche "on continue...".
```

3. Les variables déclarées sont des propriétés non-configurables de leur contexte d'exécution (la fonction courante ou le contexte global). Les variables non-déclarées sont configurables (ce qui signifie qu'elles peuvent être supprimées).

```
var a = 1;
b = 2;
```

```
delete this.a; // Lève une TypeError en mode strict. Échoue silencieuse
delete this.b;

console.log(a, b); // Lève une exception ReferenceError.
// La propriété 'b' a été supprimée et n'existe plus.
```

En raison de ces trois différences, il faut éviter de ne pas déclarer une variable car cela peut provoquer des résultats inattendus. **Il est donc fortement recommandé de toujours déclarer les variables, qu'elles soient dans une fonction ou dans la portée globale.** Le mode strict, introduit avec ECMAScript 5, lève une exception lorsqu'une variable n'est pas déclarée.

La remontée de variables (*hoisting*)

Les déclarations de variables (et les déclarations en général) sont traitées avant que n'importe quel autre code soit exécuté. Ainsi, déclarer une variable n'importe où dans le code équivaut à la déclarer au début de son contexte d'exécution. Cela signifie qu'une variable peut également apparaître dans le code avant d'avoir été déclarée. Ce comportement est appelé « remontée » (*hoisting* en anglais) car la déclaration de la variable est « remontée » au début de la fonction courante ou du contexte global.

```
bla = 2
var bla;
// ...

// est implicitement traité comme :

var bla;
bla = 2;
```

Étant donné ce comportement, il est recommandé de toujours déclarer les variables au début de leurs portées (le début du code global ou le début du corps de la fonction) afin de mieux (sa)voir quelles variables font partie de la fonction et lesquelles proviennent de la chaîne de portées.

Il est important de noter que la remontée des variables affecte uniquement la déclaration et pas l'initialisation de la valeur. La valeur sera affectée lorsque le moteur accèdera à l'instruction d'affectation. Par exemple :

```
function faireQuelqueChose() {
  console.log(truc); // undefined
  var truc = 111;
```

```
  console.log(truc); // 111
}

// Correspond en fait à :
function faireQuelqueChose() {
  var truc;
  console.log(truc); // undefined
  truc = 111;

  console.log(truc); // 111
}
```

Exemples

Déclarer et initialiser deux variables

```
var a = 0, b = 0;
```

Affecter deux variables avec la même chaîne de caractères

```
var a = "A";
var b = a;

// est équivalent à :

var a, b = a = "A";
```

Attention à l'ordre :

```
var x = y, y = 'A';
console.log(x + y); // undefinedA
```

Ici, `x` et `y` sont déclarées avant que n'importe quel code soit exécuté, **les affectations sont réalisées après !** Au moment où `x = y` est évalué, `y` existe donc on n'a pas d'erreur `ReferenceError` mais sa valeur est `undefined`. Ainsi, `x` reçoit la valeur `undefined`. Ensuite, `y` reçoit la valeur `'A'`. Après la première ligne de code, on a donc la situation où `x === undefined && y === 'A'`, ce qui explique le résultat.

Initialiser plusieurs variables

```
var x = 0; // Variable dans la portée globale (le fichier)

function f(){
  var x = y = 1; // x est déclaré localement
                 // ce qui n'est pas le cas de y !
}
```

```
}
f();

console.log(x, y); // 0, 1
// x a bien la valeur globale attendue
// y a été contaminé dans la fonction !
// Une exception ReferenceError sera levée en mode
// strict car y n'est pas défini dans cette portée
```

Les variables globales implicites

Il est possible de faire référence à des variables qui sont des variables globales implicites depuis la portée d'une fonction externe :

```
var x = 0;  // Déclare x comme variable globale du fichier, on lui affe 0

console.log(typeof z); // "undefined", car z n'existe pas encore

function a() {
  var y = 2;   // Déclare y dans la portée de la fonction a
               // Affecte 2 comme valeur à y

  console.log(x, y);   // 0 2

  function b() {
    x = 3;  // Affecte 3 à la variable globale x
            // Ne crée pas une nouvelle variable globale
    y = 4;  // Affecte 4 à la variable externe y,
            // Ne crée pas une nouvelle variable globale
    z = 5;  // Crée une nouvelle variable globale
            // et lui affecte la valeur 5.
  }         // (lève une ReferenceError en mode strict.)

  b();     // Crée z en tant que variable globale
  console.log(x, y, z);  // 3 4 5
}

a();                    // l'appel à a() entraîne un appel à b()
console.log(x, z);      // 3 5
console.log(typeof y); // "undefined" car y est local à la fonction a
```

Spécifications

Spécification	État	Commentaires

Spécification	État	Commentaires
ECMAScript 1st Edition (ECMA-262)	Standard	Définition initiale. Implémentée avec JavaScript 1.0
ECMAScript 5.1 (ECMA-262) La définition de 'instruction var' dans cette spécification.	Standard	
ECMAScript 2015 (6th Edition, ECMA-262) La définition de 'instruction de variable' dans cette spécification.	Standard	
ECMAScript (ECMA-262) La définition de 'variable statement' dans cette spécification.	Standard évolutif	

Compatibilité des navigateurs

Report problems with this compatibility data on GitHub

var	
Chrome	1
Edge	12
Firefox	1
Internet Explorer	3
Opera	3
Safari	1
WebView Android	1
Chrome Android	18
Firefox for Android	4
Opera Android	10.1
Safari on iOS	1
Samsung Internet	1.0

Deno	1.0
Node.js	0.10.0

 Full support

Voir aussi

- `let`
- `const`

Last modified: 28 janv. 2022, by MDN contributors