

# CheatSheet JavaScript

conçue par Jérémy Mouzin, créateur de la formation JavaScript de Zéro

Pour toute question, contactez-moi :

[twitter.com/jeremymouzin](https://twitter.com/jeremymouzin)  
jeremy@javascriptdezero.com

Formez-vous au JavaScript sur [www.javascriptdezero.com](http://www.javascriptdezero.com)

## OBJECTIF DE CE DOCUMENT

L'objectif de ces fiches est d'avoir une vue d'ensemble des propriétés et méthodes disponibles sur chaque type de donnée de base en JavaScript.

Les propriétés et méthodes ont été groupées par affinités et similitudes pour faciliter la lecture et l'apprentissage.

Le but est de vous permettre d'avoir une référence simple et facile à consulter pour choisir ou découvrir la propriété ou méthode dont vous avez besoin rapidement.

La définition des propriétés et méthodes est la plus succincte possible. Vous pourrez consulter la documentation MDN de celles-ci en cliquant sur le nom de votre choix.

N'hésitez pas à imprimer ces fiches et à les relire régulièrement pour apprendre toutes les méthodes qui existent. Le but final n'est pas de connaître chaque détail de toutes les méthodes et propriétés mais de simplement savoir qu'elles existent !

*N'hésitez pas à partager ce document autour de vous.  
Et bonne lecture ! 🚀*

# Number

## PROPRIÉTÉS STATIQUES (8)

Number. <u>MIN_VALUE</u>	$5^{-324}$ — Valeur <u>positive</u> minimale d'un nombre
Number. <u>MAX_VALUE</u>	$2^{1024}$ — Valeur maximale d'un nombre
Number. <u>MIN_SAFE_INTEGER</u>	$-(2^{53} - 1)$ — Valeur minimale d'un nombre entier
Number. <u>MAX_SAFE_INTEGER</u>	$2^{53} - 1$ — Valeur maximale d'un nombre entier
Number. <u>NaN</u>	<b>Not a Number</b> — Valeur qui n'est pas un nombre
Number. <u>EPSILON</u>	$2^{-52}$ — Différence entre 1 et la plus petite valeur supérieure à 1
Number. <u>NEGATIVE_INFINITY</u>	Infini négatif
Number. <u>POSITIVE_INFINITY</u>	Infini positif

## MÉTHODES STATIQUES (6)

Number. <u>isFinite(n)</u>	Détermine si <i>n</i> est un nombre fini
Number. <u>isInteger(n)</u>	Détermine si <i>n</i> est un nombre entier
Number. <u>isSafeInteger(n)</u>	Détermine si <i>n</i> est un entier stockable avec 100% de précision ?
Number. <u>isNaN(n)</u>	Détermine si <i>n</i> vaut NaN
Number. <u>parseFloat(chaîne)</u>	Convertit la <i>chaîne</i> en un nombre flottant
Number. <u>parseInt(chaîne)</u>	Convertit la <i>chaîne</i> en un nombre entier

## MÉTHODES NON STATIQUES (6)

<u>toExponential([v])</u>	Formatte en notation exponentielle, <i>v</i> chiffres après la virgule
<u>toFixed([v])</u>	Formatte en notation à point-fixe, <i>v</i> chiffres après la virgule
<u>toPrecision([n])</u>	Renvoie une chaîne de caractères avec la précision donnée
<u>valueOf()</u>	Renvoie la valeur primitive stockée dans l'objet de type Number
<u>toString([base])</u>	Renvoie le nombre en utilisant <i>base</i> pour représenter la valeur
<u>toLocaleString([locales, [opt]])</u>	Renvoie une chaîne de caractères en tenant compte des <i>locales</i>

## ASTUCES

```
const n = 0x2A // (ou 0x2a)
const n = 0b101010
(42).toString(2) → "101010"
(42).toString(8) → "52"
(42).toString(16) → "2a"
+"42" → 42
+"0x2A" → 42
```

Notation hexadécimale  $(0x2A)_{16} = (42)_{10}$   
Notation binaire  $(0b101010)_2 = (42)_{10}$   
Conversion base décimale → binaire  
Conversion base décimale → octale  
Conversion base décimale → hexadécimale  
Conversion en un nombre avec l'opérateur unaire "+"

- ⚠ `NaN == NaN → false`
- ⚠ `NaN === NaN → false`
- ⚠ `NaN != NaN → true`
- ✓ `Number.isNaN(NaN) → true`

La valeur NaN n'est **jamais** égale à elle-même. Pour tester si une variable *valeur* contient NaN on peut aussi faire :  
`valeur != valeur → true` uniquement pour NaN

`Number.parseInt("42 km") → 42` Les lettres et espaces après les chiffres sont ignorés  
`Number.parseFloat("1.6m") → 1.6` C'est le point "." qui fait office de virgule ","

# String

## PROPRIÉTÉS NON STATIQUES (2)

length	Renvoie la longueur de la chaîne
N	Renvoie le caractère présent à la position N — <span style="color: yellow;">⚠</span> Le 1er caractère est à la position 0

## MÉTHODES STATIQUES (3)

String. <u>fromCharCode(n1, n2, ...)</u>	Renvoie la chaîne de caractères représentant les points de code UTF-16 <i>n1, n2, ... sans</i> gestion des caractères supplémentaires
String. <u>fromCodePoint(n1, n2, ...)</u>	Renvoie la chaîne de caractères représentant les points de code UTF-16 <i>n1, n2, ... avec</i> gestion les caractères supplémentaires (paires de substitution) — Cette méthode est préférée !
String. <u>raw()</u>	Renvoie une chaîne de caractères basée sur une template string : les substitutions sont faites mais les séquences d'échappement (\n, etc.) ne sont pas interprétées

## MÉTHODES NON STATIQUES (31)

<u>charAt(i)</u>	Renvoie le caractère à la position <i>i sans</i> gestion des caractères supplémentaires (paires de substitution)
<u>charCodeAt(i)</u>	Renvoie un entier entre 0 et 65535 correspondant au code UTF-16 du caractère à la position <i>i sans</i> gestion des caractères suppl.
<u>codePointAt(i)</u>	Renvoie un entier représentant le caractère à la position <i>i avec</i> gestion des caractères supplémentaires
<u>concat(ch1 [, ch2, ...])</u>	Concatène les chaînes de caractères
<u>includes(rech [, position])</u>	Détermine si <i>rech</i> est inclue dans la chaîne depuis la <i>position</i>
<u>indexOf(rech [, position])</u>	Renvoie la position de la <u>première</u> occurrence de <i>rech</i> dans la chaîne à partir de <i>position</i>
<u>lastIndexOf(rech [, position])</u>	Renvoie la position de la <u>dernière</u> occurrence de <i>rech</i> depuis le début de la chaîne jusqu'à <i>position</i>
<u>search(regexp)</u>	Renvoie la position de la première correspondance de <i>regexp</i>
<u>match(regexp)</u>	Renvoie un tableau contenant les informations de correspondances de la chaîne avec l'expression régulière <i>regexp</i>
<u>matchAll(regexp)</u>	Renvoie un itérateur contenant toutes les correspondances de la chaîne avec l'expression régulière <i>regexp</i>
<u>padStart(long [, motif])</u>	Renvoie la chaîne avec le <i>motif</i> répété au début de celle-ci pour atteindre une longueur de <i>long</i> caractères
<u>padEnd(long [, motif])</u>	Renvoie la chaîne avec le <i>motif</i> répété à la fin de celle-ci pour atteindre une longueur de <i>long</i> caractères
<u>repeat(long)</u>	Renvoie la chaîne répétée plusieurs fois de suite pour atteindre une longueur de <i>long</i> caractères
<u>trim()</u>	Renvoie la chaîne après avoir retiré les espaces blancs au début et à la fin de celle-ci
<u>trimStart()   trimLeft()</u>	Renvoie la chaîne après avoir retiré les espaces blancs au début de celle-ci
<u>trimEnd()   trimRight()</u>	Renvoie la chaîne après avoir retiré les espaces blancs à la fin de celle-ci

# String

## MÉTHODES NON STATIQUES (31)

replace(recherche, remplacer)  
    regexp      fnRemplacer

Renvoie la chaîne dont on aura remplacé :

- la 1ère occurrence de la chaîne *recherche* par *remplacer*
- la 1ère occurrence de la *regexp* (si **sans** /g) par *remplacer*
- toutes les occurrences de la *regexp* (si **avec** /g) par *remplacer*

On peut également remplacer dynamiquement des valeurs via une fonction *fnRemplacer*

replaceAll(recherche, remplacer)  
    regexp      fnRemplacer

Renvoie la chaîne dont on aura remplacé :

- toutes les occurrences de la chaîne *recherche* par *remplacer*
- toutes les occurrences de la *regexp* (/g obligatoire) par *remplacer*

On peut également remplacer dynamiquement des valeurs via une fonction *fnRemplacer*

slice(debut [, fin])

Renvoie la chaîne entre la position *debut* **inclus** jusqu'à la position *fin* **exclue**. Les positions *debut* et *fin* **supportent** les nombres négatifs (on compte à partir de la fin de la chaîne)

substring(debut [, fin])

Renvoie la chaîne entre la position *debut* **inclus** jusqu'à la position *fin* **exclue**. Les positions *debut* et *fin* **ne supportent pas** les nombres négatifs : ils seront remplacés par 0.

Si *fin < debut* alors leurs valeurs sont échangées

startsWith(rech [, position])

Découpe la chaîne selon le *separateur* ou la *regexp* et renvoie un tableau avec les sous-chaînes dans la limite de *limite* éléments

endsWith(rech [, longueur])

Détermine si la chaîne commence par *rech* en partant de la *position*

toLowerCase()

Détermine si la chaîne se termine par *rech* en prenant en compte les *longueur* premiers caractères de cette chaîne

toUpperCase()

Renvoie la chaîne convertie en minuscules

toLocaleLowerCase([locale, ...locales])

Renvoie la chaîne convertie en majuscules

toLocaleUpperCase([locale, ...locales])

Renvoie la chaîne en minuscules en respectant la *locale*

valueOf()

Renvoie la chaîne en majuscules en respectant la *locale*

toString()

Renvoie la valeur primitive stockée dans l'objet de type String

localeCompare(ch [, loc, [opt]])

Renvoie la représentation textuelle de l'objet String

normalize([form])

Compare l'ordre lexicographique de 2 chaînes de caractères

Renvoie un nombre < 0 si la chaîne vient avant *ch*

Renvoie un nombre > 0 si la chaîne vient après *ch*

Renvoie 0 si la chaîne est la même que *ch*

Renvoie la forme normalisée Unicode de la chaîne de caractères

# Object

## MÉTHODES STATIQUES (21)

Object.assign(cible, ...sources)

Renvoie l'objet *cible* après avoir fusionné toutes ses propriétés avec celles des objets *sources*

Object.create(proto [, propsObj])

Crée un objet avec le prototype *proto* dont les propriétés peuvent être configurées grâce à l'objet *propsObj*

Object.entries(obj)

Retourne un tableau dont chaque élément est un tableau [“clé”, valeur] représentant chaque propriétés de l'objet *obj*

Object.fromEntries(iterable)

Crée un objet à partir d'un ensemble de paires [“clé”, valeur] d'un objet *iterable* (Array, Map etc.). Fonction inverse de *entries()*

Object.keys(obj)

Renvoie un tableau des noms des propriétés de l'objet *obj*

⚠ Les propriétés non-énumérables sont **exclues**

Object.getOwnPropertyNames(obj)

Renvoie un tableau des noms des propriétés de l'objet *obj*

⚠ Les propriétés non-énumérables sont **incluses**

Object.is(valeur1, valeur2)

Détermine si *valeur1* et *valeur2* sont identiques. Différences avec l'opérateur d'égalité stricte === :

Object.is(NaN, NaN) → **true**      NaN === NaN → **false**

Object.is(-0, 0) → **false**      -0 === 0 → **true**

Object.preventExtensions(obj)

Interdit l'ajout de nouvelles propriétés sur l'objet *obj*

Object.freeze(obj)

Gèle l'objet *obj* : on ne peut plus modifier celui-ci

Object.seal(obj)

Scelle l'objet *obj* : on ne peut plus ajouter ni supprimer de propriétés mais on peut encore modifier leurs valeurs

Object.isExtensible(obj)

Détermine si on peut ajouter des propriétés à l'objet *obj*

Object.isFrozen(obj)

Détermine si l'objet *obj* est gelé

Object.isSealed(obj)

Détermine si l'objet *obj* est scellé

Object.defineProperty(obj, prop, descriptor)

Définit la propriété *prop* sur l'objet *obj* en utilisant le *descripteur* pour configurer les attributs de cette propriété

Object.defineProperties(obj, props)

Même chose mais permet de définir plusieurs propriétés d'un coup au travers de l'objet *props*

Renvoie la configuration de la propriété *prop* de l'objet *obj*

Object.getOwnPropertyDescriptor(obj, prop)

Renvoie toutes les configurations des propriétés de l'objet *obj*

(obj)

Object.getOwnPropertyDescriptors(obj)

Renvoie un tableau des symboles présents sur l'objet *obj*

(obj)

Object.getOwnPropertySymbols(obj)

Renvoie le prototype de l'objet *obj*

Object.setPrototypeOf(obj, proto)

Définit le prototype *proto* comme prototype de l'objet *obj*

Object.values(obj)

Renvoie un tableau des valeurs des propriétés de l'objet *obj*

# Object

## MÉTHODES NON STATIQUES (6)

hasOwnProperty(prop)

isPrototypeOf(obj)

propertyIsEnumerable(prop)

• toLocaleString(locale)

• toString()

• valueOf()

Détermine si l'objet possède la propriété *prop* (héritage **exclu**)

Détermine si l'objet existe dans la chaîne de prototypes de *obj*

Détermine si la propriété *prop* est énumérable

Renvoie une chaîne de caractères représentant l'objet en tenant compte de la *locale*

Renvoie une chaîne de caractères représentant l'objet

Renvoie la valeur primitive de l'objet

- Ces méthodes ont pour objectif d'être surchargées par des types de données dérivées d'*Object* comme *String*, *Number*, *Date* etc.

## ASTUCES

`Object.assign({}, obj1)`

ou

`clone = { ...obj1 }`

Clone l'objet *obj1* dans un nouvel objet. Effectue un clonage superficiel (🇬🇧 shallow clone)

Même chose en utilisant l'opérateur spread ...

`Object.assign({}, obj1, obj2)`

ou

`clone = { ...obj1, ...obj2 }`

Fusionne *obj1* et *obj2* dans un nouvel objet. Effectue un clonage superficiel (🇬🇧 shallow clone)

Même chose en utilisant l'opérateur spread ...

`JSON.parse(JSON.stringify(obj1))`

Clone l'objet *obj1* dans un nouvel objet. Effectue un clonage profond (🇬🇧 deep clone). ⚠ Fonctionne seulement si *obj1* peut être transformé en JSON correctement

`delete obj.prop`

Supprime la propriété *prop* de l'objet *obj*

# Array

## MÉTHODES STATIQUES (3)

Array. <u>from(iterable [, fnMap[, this]])</u>	Crée un tableau à partir d'un <i>iterable</i> ou d'un objet ressemblant à un tableau ( <i>array-like</i> ). La fonction <i>fnMap</i> sera appliquée à chaque élément lors de la création du tableau (cf. <i>map()</i> )
Array. <u>isArray(obj)</u>	Détermine si <i>obj</i> est un tableau
Array. <u>of(elem1 [, elem2[, ... , elemN]])</u>	Crée un tableau à partir d'un nombre variable d'arguments

## PROPRIÉTÉS NON STATIQUES (2)

length	Renvoie la taille du tableau (nombre d'éléments)
N	Renvoie l'élément présent à la position N —  Le 1er élément est à la position 0

## ASTUCES

Array.from(Array(N), (v,i) => i)	Génère un tableau de <i>N</i> nombres de 0 à <i>N</i> -1
Array.from(Array(N), (v,i) => i+1)	Génère un tableau de <i>N</i> nombres de 1 à <i>N</i>
[a, b] = [b, a]	Échange les valeurs des variables <i>a</i> et <i>b</i>
Array.from(new Set([0,0,1,2,2,3])) ou [...new Set([0,0,1,2,2,3])]	Filtre les doublons dans un tableau. Renverra un nouveau tableau : [0,1,2,3]
tableau.slice(-1)	Renvoie le dernier élément du <i>tableau</i>
chaine.split('')	Crée un tableau depuis une chaîne de caractères
ou	
Array.from(chaine)	
ou	
[...chaine]	
const tableau = [0,1,2,3,4]; tableau.length = 2; // → [0,1] tableau.length = 0; // → []	Tronque le tableau à une certaine taille Vide le tableau

# Array

## MÉTHODES NON STATIQUES (31)

<u>concat(val1 [, val2 [, ..., valN]])</u>	Renvoie un nouveau tableau qui sera la concaténation du <i>tableau</i> initial avec <i>val1</i> , <i>val2</i> , <i>valN</i> dans cet ordre
<u>copyWithin(cible [, debut [, fin]])</u>	Copie <b>en place</b> le sous-ensemble du <i>tableau</i> depuis la position <i>debut</i> jusqu'à <i>fin exclude</i> à la position <i>cible</i> de ce même tableau
<u>fill(valeur [, debut [, fin]])</u>	Affecte <b>en place</b> <i>valeur</i> à tous les éléments du <i>tableau</i> à partir de la position <i>debut</i> jusqu'à la position <i>fin exclude</i>
<u>entries()</u> <u>keys()</u> <u>values()</u>	Renvoie un itérateur des paires clé-valeur des éléments du <i>tableau</i> Renvoie un itérateur avec les <u>clés</u> de chaque élément du <i>tableau</i> Renvoie un itérateur avec les <u>valeurs</u> de chaque élément du <i>tableau</i>
<u>join(separateur)</u>	Retourne une chaîne de caractères contenant tous les éléments du <i>tableau</i> séparés par le <i>separateur</i>
<u>reverse()</u> <u>slice([debut [, fin]])</u> <u>splice(debut [, supp [, el1, ... elN]])</u>	Retourne le <i>tableau</i> à l'envers. Modifie le <i>tableau</i> original <b>en place</b> Retourne un sous-ensemble du <i>tableau</i> de <i>debut</i> à <i>fin exclude</i> Supprime ou ajoute <b>en place</b> de nouveaux éléments au <i>tableau</i>
<u>includes(element [, position])</u> <u>find(fonction [, this])</u> <u>findIndex(fonction [, this])</u>	Détermine si le <i>tableau</i> contient l' <i>element</i> à partir de la <i>position</i> Renvoie le 1er élément passant le test implémenté par <i>fonction</i> Renvoie la <u>position</u> du 1er élément passant le test implémenté par <i>fonction</i>
<u>indexOf(element [, position])</u> <u>lastIndexOf(element [, position])</u>	Retourne la <u>position</u> de l' <i>element</i> à partir de la <i>position</i> Renvoie la <u>dernière</u> position de l' <i>element</i> à partir du début du <i>tableau</i> jusqu'à <i>position include</i>
<u>forEach(fonction [, this])</u>	Exécute la <i>fonction</i> pour chaque élément du <i>tableau</i> ⚠ Il est impossible d'arrêter une boucle <i>forEach</i> via l'instruction <i>break</i> par exemple : il faut lancer une exception ⚠ La <i>fonction</i> doit être <u>synchrone</u> utiliser <i>async</i> et les promesses ne fonctionnera pas comme prévu
<u>map(fonction [, this])</u> <u>filter(fonction [, this])</u> <u>reduce(fonction [, this])</u> <u>reduceRight(fonction [, this])</u>	Renvoie un tableau contenant les résultats de l'appel de <i>fonction</i> sur chacun des éléments du <i>tableau</i> Renvoie un tableau ne contenant que les éléments qui passent le test implémenté par <i>fonction</i> Exécute la fonction de réduction <i>fonction</i> sur chaque élément du <i>tableau</i> , dans l'ordre, et renvoie une unique valeur Même chose que <i>reduce()</i> mais dans l'ordre inverse
<u>some(fonction [,this])</u> <u>every(fonction [, this])</u>	Détermine si <u>au moins un</u> élément du <i>tableau</i> passe le test implémenté dans la <i>fonction</i> Détermine si <u>tous</u> les éléments du <i>tableau</i> passent le test implémenté dans la <i>fonction</i>

# Array

## MÉTHODES NON STATIQUES (31)

<u>push(elem1 [, elem2 [, ..., elemN]])</u>	Ajoute <b>en place</b> les éléments <i>elem1, elem2, elemN</i> à la fin du <i>tableau</i> et renvoie sa nouvelle taille
<u>pop()</u>	Supprime <b>en place</b> et renvoie le <u>dernier</u> élément du <i>tableau</i>
<u>shift()</u>	Supprime <b>en place</b> et renvoie le <u>premier</u> élément du <i>tableau</i>
<u>unshift()</u>	Ajoute <b>en place</b> les éléments <i>elem1, elem2, elemN</i> au début du <i>tableau</i> et renvoie sa nouvelle taille
<u>sort(fonction)</u>	Trie <b>en place</b> les éléments du <i>tableau</i> grâce à la <i>fonction</i>
<u>flat([profondeur])</u>	Renvoie un tableau aplati c'est-à-dire dont les sous-tableaux du <i>tableau</i> ont été concaténés selon la <i>profondeur</i> donnée
<u>flatMap(fonction)</u>	Exécute la fonction <i>map(fonction)</i> puis <i>flat(1)</i> sur le <i>tableau</i>
<u>toString()</u>	Retourne une chaîne représentant les éléments du <i>tableau</i>
<u>toLocaleString([locales [, options]])</u>	Retourne une chaîne représentant les éléments du <i>tableau</i> en tenant compte des <i>locales</i> et des <i>options</i>