CheatSheet JavaScript

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

Pour toute question, contactez-moi :

twitter.com/jeremymouzin
jeremy@javascriptdezero.com

Formez-vous au JavaScript sur 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 succinte 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.MIN_VALUE	$5x10^{-324}$ — Valeur positive minimale d'un nombre
Number.MAX_VALUE	2^{1024} — Valeur maximale d'un nombre
Number.MIN_SAFE_INTEGER	$-(2^{53} - 1)$ — Valeur minimale d'un nombre entier
Number.MAX_SAFE_INTEGER	$2^{53} - 1$ — Valeur maximale d'un nombre entier
Number.NaN	**N**ot **a** **N**umber — Valeur qui n'est pas un nombre
Number.EPSILON	2^{-52} — Différence entre 1 et la plus petite valeur supérieure à 1
Number.NEGATIVE_INFINITY	Infini négatif
Number.POSITIVE_INFINITY	Infini positif

MÉTHODES STATIQUES (6)

Number.isFinite(n)	Détermine si *n* est différent de `+infini`, `-infini` et NaN
Number.isInteger(n)	Détermine si *n* est un nombre entier
Number.isSafeInteger(n)	Détermine si *n* est un entier stockable avec 100% de précision
Number.isNaN(n)	Détermine si *n* vaut NaN
Number.parseFloat(chaîne)	Convertit la *chaîne* en un nombre flottant
Number.parseInt(chaîne)	Convertit la *chaîne* en un nombre entier

MÉTHODES NON STATIQUES (6)

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

ASTUCES

`const n = 0x2A // (ou 0x2a)`	Notation hexadécimale $(0x2A)_{16} = (42)_{10}$
`const n = 0b101010`	Notation binaire $(0b101010)_2 = (42)_{10}$
`(42).toString(2) → "101010"`	Conversion base décimale → binaire
`(42).toString(8) → "52"`	Conversion base décimale → octale
`(42).toString(16) → "2a"`	Conversion base décimale → hexadécimale
`+"42" → 42`	Conversion en un nombre avec l'opérateur unaire "+"
`+"0x2A" → 42`	
⚠️ `NaN == NaN → false`	La valeur NaN n'est **jamais** égale à elle-même. Pour
⚠️ `NaN === NaN → false`	tester si une variable *valeur* contient NaN on peut aussi
⚠️ `NaN !== NaN → true`	faire :
✅ `Number.isNaN(NaN) → true`	`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 — ⚠️ Le 1er caractère est à la position 0

MÉTHODES STATIQUES (3)

String.fromCharCode(n1, n2, ...)	Renvoie la chaîne de caractères représentant les points de code UTF-16 *n1*, *n2*, ... **sans** gestion des caractères supplémentaires
String.fromCodePoint(n1, n2, ...)	Renvoie la chaîne de caractères représentant les points de code UTF-16 *n1*, *n2*, ... **avec** gestion les caractères supplémentaires (paires de substitution) — Cette méthode est préférée !
String.raw()	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)

charAt(i)	Renvoie le caractère à la position *i* **sans** gestion des caractères supplémentaires (paires de substitution)
charCodeAt(i)	Renvoie un entier entre 0 et 65535 correspondant au code UTF-16 du caractère à la position *i* **sans** gestion des caractères suppl.
codePointAt(i)	Renvoie un entier représentant le caractère à la position *i* **avec** gestion des caractères supplémentaires
concat(ch1 [, ch2, ...])	Concatène les chaînes de caractères
includes(rech [, position])	Détermine si *rech* est inclue dans la chaîne depuis la *position*
indexOf(rech [, position])	Renvoie la position de la première occurrence de *rech* dans la chaîne à partir de *position*
lastIndexOf(rech [, position])	Renvoie la position de la dernière occurrence de *rech* depuis le début de la chaîne jusqu'à *position*
search(regexp)	Renvoie la position de la première correspondance de *regexp*
match(regexp)	Renvoie un tableau contenant les informations de correspondances de la chaîne avec l'expression régulière *regexp*
matchAll(regexp)	Renvoie un itérateur contenant toutes les correspondances de la chaîne avec l'expression régulière *regexp*
padStart(long [, motif])	Renvoie la chaîne avec le *motif* répété au début de celle-ci pour atteindre une longueur de *long* caractères
padEnd(long [, motif])	Renvoie la chaîne avec le *motif* répété à la fin de celle-ci pour atteindre une longueur de *long* caractères
repeat(long)	Renvoie la chaîne répétée plusieurs fois de suite pour atteindre une longueur de *long* caractères
trim()	Renvoie la chaîne après avoir retiré les espaces blancs au début et à la fin de celle-ci
trimStart() | trimLeft()	Renvoie la chaîne après avoir retiré les espaces blancs au début de celle-ci
trimEnd() | trimRight()	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 occurence de la chaîne *recherche* par *remplacer* • la 1ère occurence de la *regexp* (si **sans** */g*) par *remplacer* • toutes les occurences 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 occurences de la chaîne *recherche* par *remplacer* • toutes les occurences 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* **incluse** 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* **incluse** 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
split([separateur [, limite]]) regexp	Découpe la chaîne selon le *separateur* ou la *regexp* et renvoie un tableau avec les sous-chaînes dans la limite de *limite* éléments
startsWith(rech [, position])	Détermine si la chaîne commence par *rech* en partant de la *position*
endsWith(rech [, longueur])	Détermine si la chaîne se termine par *rech* en prenant en compte les *longueur* premiers caractères de cette chaîne
toLowerCase()	Renvoie la chaîne convertie en minuscules
toUpperCase()	Renvoie la chaîne convertie en majuscules
toLocaleLowerCase([locale, ...locales])	Renvoie la chaîne en minuscules en respectant la *locale*
toLocaleUpperCase([locale, ...locales])	Renvoie la chaîne en majuscules en respectant la *locale*
valueOf()	Renvoie la valeur primitive stockée dans l'objet de type String
toString()	Renvoie la représentation textuelle de l'objet String
localeCompare(ch [, loc, [opt]])	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*
normalize([form])	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, descripteur)	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*
Object.getOwnPropertyDescriptor(obj, prop)	Renvoie la configuration de la propriété *prop* de l'objet *obj*
Object.getOwnPropertyDescriptors(obj)	Renvoie toutes les configurations des propriétés de l'objet *obj*
Object.getOwnPropertySymbols(obj)	Renvoie un tableau des symboles présents sur l'objet *obj*
Object.getPrototypeOf(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)	Détermine si l'objet possède la propriété *prop* (héritage **exclu**)
isPrototypeOf(obj)	Détermine si l'objet existe dans la chaîne de prototypes de *obj*
propertyIsEnumerable(prop)	Détermine si la propriété *prop* est énumérable
• toLocaleString(locale)	Renvoie une chaîne de caractères représentant l'objet en tenant compte de la *locale*
• toString()	Renvoie une chaîne de caractères représentant l'objet
• valueOf()	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.from(iterable [, fnMap[, this]])	Crée un tableau à partir d'un *iterable* ou d'un objet ressemblant à un tableau (🇬🇧 *array-like*). La fonction *fnMap* sera appliquée à chaque élément lors de la création du tableau (cf. *map()*)
Array.isArray(obj)	Détermine si *obj* est un tableau
Array.of(elem1 [, elem2[, ..., elemN]])	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 *N* nombres de 0 à *N*-1
`Array.from(Array(N), (v,i) => i+1)`	Génère un tableau de *N* nombres de 1 à *N*
`[a, b] = [b, a]`	Échange les valeurs des variables *a* et *b*

`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 *tableau*

`chaine.split('')`
ou
`Array.from(chaine)`
ou
`[...chaine]`
	Crée un tableau depuis une chaîne de caractères

`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)

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

Array

MÉTHODES NON STATIQUES (31)

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