JAVASCRIPT 1

[**Introduction**](#_ex9z49rygtrb) **4**

[**Type**](#_hfksgqliu70s) **4**

[**Manipulation numbers**](#_twob7rbffoty) **5**

[parseInt(a)](#_fyyfk5ykuxvd) 5

[parseFloat(a)](#_5nvbvf779k8w) 6

[isNaN(a)](#_ue99a5s2u640) 6

[toFixed(a)](#_ushf84yn0htx) 7

[**Manipulation fonctions**](#_xhdnth2l2saz) **8**

[Déclarations](#_eh3cw4mr2v3z) 8

[Utilisations fonctions usuelles](#_svm2k9sa0prb) 9

[Paramètres](#_ye9dp09h2i7o) 9

[**Manipulation strings**](#_wtym6xavnhnt) **11**

[toLowerCase()](#_883cj0hed2pl) 11

[toUpperCase()](#_tolfn6jsbukj) 11

[split(a)](#_badx1r5v65le) 12

[replace(a, b)](#_54z0ydihe64) 12

[indexOf(a)](#_jx9korltjhvg) 13

[lastIndexOf(a)](#_671twa6kdu1p) 13

[substr(start, fin)](#_6eigrnqee5vn) 13

[**Manipulation tableaux**](#_j57rbqpqxmf8) **14**

[isArray](#_aol2h3h07f7l) 14

[push(element)](#_fii6t0cghe5b) 15

[reverse()](#_m2mpl2p5etp7) 15

[sort(fn(a,b))](#_ltdd9zjgti14) 15

[reduce(fn(total, element), somme)](#_xrnsvyt4i7nu) 16

[filter(fn(element))](#_bu4ary5i60am) 16

[join(glue)](#_7n0ce4qi5x1d) 16

[concat(elements 1, élements2, ...)](#_j783q5a369k0) 17

[indexOf(element)](#_xb9igxqmzz5s) 17

[lastIndexOf(element)](#_2l5zkyeaif5c) 18

[map(fn(element))](#_he5dc3f6ykax) 18

[forEach(fn(e))](#_1z5ry2c69ri1) 18

[Suppression d’un élement](#_66h6a73ka7tl) 19

[**Manipulation objects**](#_ap25kobmzugz) **20**

[Accès aux clefs/valeurs](#_i5z86ypiq5jm) 20

[Définition d’une clefs/valeurs](#_bvj7h1sn6mht) 21

[Récupération d’une liste de clefs](#_ntwglqs7y5z6) 21

[Suppression d’une clef/valeurs](#_2wakqz464shq) 21

[Loop](#_f99h94se6wxs) 22

[Transformation objet en string](#_te7y3cs2x678) 23

[Transformation string en objet](#_928fzfksf51o) 25

[**Exemples complexes**](#_pct6zhdvaym7) **26**

# Introduction

Nul doute que tu connais javascript sur le bout des doigts, cependant il est bon de connaître par coeur la plupart des opérations élémentaires sur les entiers/flottants, les strings, les tableaux, et les objets. Nous allons revoir les opérations de bases sur les types primaires. Les opérations plus complexes sur les dates / regex, etc… seront vu dans un prochain chapitre.

# Type

Pour rappel en javascript les types de base sont les suivants :

* **number** : 3, 4.3, ...
* **string**: “bonjour”, “”, …
* **object**: null, {v1: 2}, [1, 2, 3], …
* **undefined**: non définis
* **function**: function(a){console.log(a);}

**Attention au type object qui contient null, objet javascript et tableau**

Une variable “var a;” n’a pas de type jusqu’a l’instanciation de la variable. Son initialisation définit son type et les opérations que l’on va pouvoir effectuer avec.

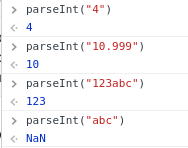
# 

# Manipulation numbers

* parseInt: transformation string en int
* parseFloat: transformation string en float
* isNaN: n’est pas un nombre ?
* toFixed: précision taille du float

## parseInt(a)

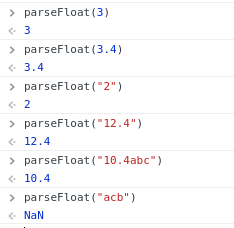
Retourne l’entier associé à la variable a, ou a peut être une string, un entier, un float, etc…



## 

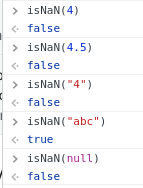
## parseFloat(a)

Retourne le float associé à la variable a, ou a peut être une string, un entier, un float, etc…



## isNaN(a)

Retourne true si a n’est pas un nombre, est capable de gérer des strings



De manière générale on vérifie après un parseInt ou un parseFloat que isNaN retourne false si l’on est pas sur du type de la variable

## 

## toFixed(a)

Transforme l’int ou le float en string avec a digits après la virgule. Attention, le nombre est transformé en string



# 

# Manipulation fonctions

## Déclarations

Les fonctions peuvent être déclarée de plusieurs façon:

*Déclaration standard:*

function fn(a)

{

console.log(a);

}

*Déclaration en variable:*

var fn = function(a)

{

console.log(a);

}

Déclaration anonyme en instantiation:

(function(e){

console.log(e);

})(1);

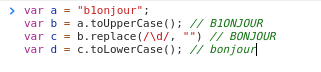
## 

## Utilisations fonctions usuelles

De manière générale les fonctions usuelles que nous allons voir ci dessous ne modifie pas les paramètres en entrées, elles renvoient simplement un nouveau résultat en ne modifiant pas la variable initiale.



Egalement, il est possible de les chaîner afin d’appliquer plusieurs fonctions séquentiellement sur un paramètre initial :

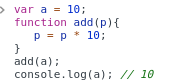




## Paramètres

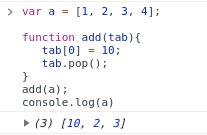
Les fonctions utilisent un passage de paramètre par copie pour les types simples (int, float, string) et par références pour les types complexes (tableau, objet). Attention donc car les objets et tableau peuvent se retrouver modifier dans les fonctions !

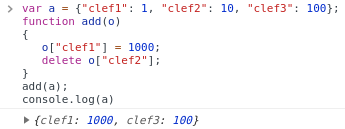
*Type simples, pas de modification dans la fonction*





*Types complexes, modifications dans la fonction*





# 

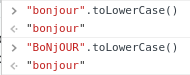
# Manipulation strings

* **toLowerCase**: passage en minuscule
* **toUpperCase**: passage en majuscule
* **split**: transformation en tableau avec un delimiter
* **replace**: remplacement de sous-string dans la string
* **indexOf**: recherche la première occurrence de sous-string dans la string
* **lastIndexOf**: recherche la dernière occurrence de sous-string dans la string
* **substr**: récupération d’une sous string a partir d’index

Nous ne détaillons pas comment effectuer une loop sur les caractères d’une string de caractère, car en pratique on ne fait jamais ça

## toLowerCase()

Transforme la string en miniscule



## toUpperCase()

Transforme la string en majuscule

# 

## 

## split(a)

Coupe une string selon le délimiter a, a peut être une string, un caractère ou vide. Attention la string est transformée en tableau



## replace(a, b)

Remplace a par b dans la string, a peut être une string ou une regex, b est une string. Attention, si a est une string le motif est remplacé une seule fois



## indexOf(a)

Recherche l’index de a dans la string (la première apparition), a peut être un caractère ou une string. Attention les index commence à 0, si a n’est pas présent dans la string, indexOf retourne -1



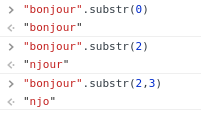
## lastIndexOf(a)

Recherche l’index de a dans la string (la dernière apparition), a peut être un caractère ou une string. Attention les index commence à 0, si a n’est pas présent dans la string, indexOf retourne -1



## substr(start, fin)

Retourne une sous-string de la string principale à partir de l’index start, fin n’est pas obligatoire, s’il est défini il correspond à la longueur de la string à extraire



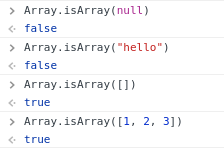
# Manipulation tableaux

Les tableaux sont sans doute l’un des objets javascript les plus utilisés. Leurs maitrises est essentielle.

* **isArray**: vérifie que l’objet est un tableau
* **push**: ajout d’un élément dans le tableau
* **reverse**: inverse l’ordre des éléments d’un tableau
* **sort**: tris un tableau
* **reduce**: effectue une somme sur un tableau
* **filter**: filtre un tableau
* **join**: transforme un tableau en string
* **concat**: merge plusieurs tableau
* **indexOf** : recherche d’un élément dans un tableau
* **lastIndexOf**: recherche d’un élément dans un tableau
* **map**: effectue des opérations sur chaque élément d’un tableau
* **forEach**: loop sur un tableau
* **suppression d’un élément**

## isArray

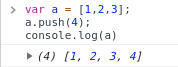
Vérifie si un tableau est bien un tableau



## 

## push(element)

Ajoute un élément à la fin d’un tableau. Attention la fonction renvoi l’élément ajouté et pas le tableau



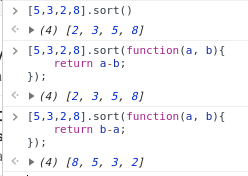
## reverse()

Retourne un tableau



## sort(fn(a,b))

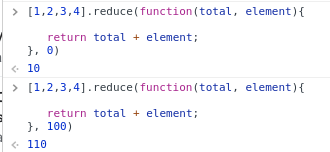
Tris un tableau, fn est optionnel et est une fonction de tris. Si fn n’est pas définis un tris croissant s’applique par défaut



## 

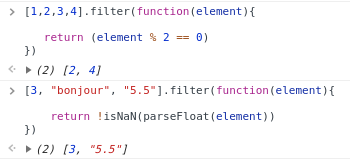
## reduce(fn(total, element), somme)

Permet de réaliser une somme d’un tableau. La fonction applique fn à chaque élément du tableau, total prend la valeur du total précédent et a été initialisé avec somme.



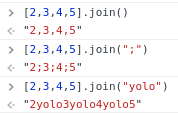
## filter(fn(element))

Filtre un tableau en appliquant fn sur chaque element. Fn doit retourner true si l'élément doit être dans le tableau final et false sinon



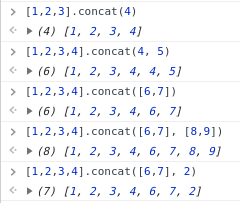
## join(glue)

Transforme un tableau en string en collant glue entre chaque element du tableau. Glue peut être vide ( “,” utilisé par défaut)



## concat(elements 1, élements2, ...)

Ajoute les éléments (autant que l’on veut) passés en paramètres au tableau. Les éléments peuvent être de type tableau, ou des objets, ou des float, des strings, etc...



## indexOf(element)

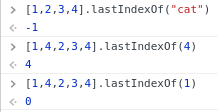
Recherche la première occurrence d’un élément dans un tableau. Attention fonctionne correctement avec les string, int, floats, peut s’avérer complexe avec les objets. Attention les index commence à 0, retourne -1 si l’élément n’est pas dans le tableau



## 

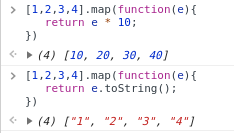
## lastIndexOf(element)

Recherche la dernière occurrence d’un élément dans un tableau. Attention fonctionne correctement avec les string, int, floats, peut s’avérer complexe avec les objets. Attention les index commence à 0, retourne -1 si l’élément n’est pas dans le tableau



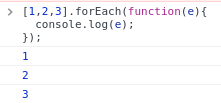
## map(fn(element))

Applique une fonction sur chaque élément d’un tableau, et retourne le tableau modifié. Fn doit renvoyer l'élément modifié



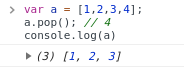
## forEach(fn(e))

Loop sur un tableau et appel fn pour chaque element

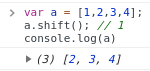


## Suppression d’un élement

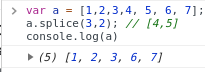
**pop()** : Suppression du dernier élément, attention retourne l’élément et non le tableau :



**shift():** Suppression du premier élément, attention retourne l’élément et non le tableau :



**splice(i, l):** Supprime l index en commencant à l’index i, , attention retourne l’élément et non le tableau :



# 

# Manipulation objects

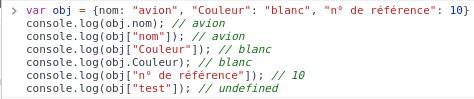
Les objets javascript sont des sortes de hash (dictionary pour ceux qui viennent du java), une clef qui est toujours une string est associée à une valeur qui peut être une string, null, un entier, un tableau, etc… ou encore un autre objet

* Accès aux clefs/valeurs
* Définition d’une clefs/valeurs
* Récupération liste des clefs
* Suppression d’une clefs/valeurs
* Loop sur objet
* Transformation string en objet
* Transformation objet en string

## Accès aux clefs/valeurs

On accède au valeurs d’un objet par les opérateurs . ou [ ], si l’on utilise le . la clef ne doit pas contenir d’espace, de tiret, etc…, si l’on utilise les crochets on doit obligatoirement utiliser une string entre les crochets.

L’accès à une clef non définis retourne undefined



## 

## Définition d’une clefs/valeurs

De la même manière que l’accès on associe une valeur à une clef via les opérateurs . et [ ] , si l’on utilise le . la clef ne doit pas contenir d’espace, de tiret, etc…, si l’on utilise les crochets on doit obligatoirement utiliser une string entre les crochets.

Si la clef existait déjà la valeur est remplacée, si la clef n’existait pas le couple clefs/valeurs est créée.



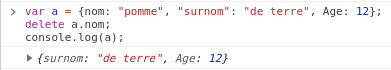
## Récupération d’une liste de clefs

Object.keys permet de récupérer la liste des clefs d’un objet dans un tableau.



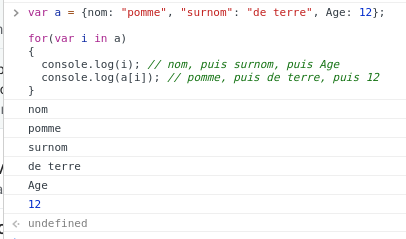
## Suppression d’une clef/valeurs

On peut supprimer une clef avec l'opérateur delete, c’est le seul moyen de faire.

******

## Loop

On utilise l’opérateur for pour looper sur les objets mais sans index direct avec un itérateur qui va prendre la valeur de chacune des clefs.

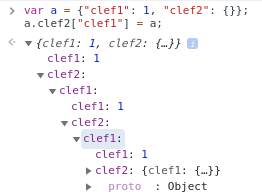


## 

## Transformation objet en string

On peut transformer un objet en string avec la fonction **JSON.stringify** , attention cette fonction peut envoyer une exception (qui potentiellement peut arrêter un serveur) si l’objet n’est pas valide ou circulaire (objet qui se contient lui même). Cette fonction doit systématiquement être entouré d’un try catch

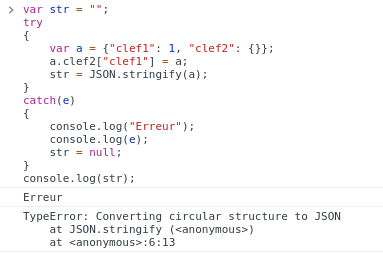
*Structure circulaire (clef2 contient l’objet a qui contient clef2, qui contient l’objet a, etc…)*



*Erreur circulaire*



*Comportement standard:*



## 

## Transformation string en objet

On peut transformer un objet en string avec la fonction **JSON.parse** , attention cette fonction peut envoyer une exception (qui potentiellement peut arrêter un serveur) si la string n’est pas un objet valide. Cette fonction doit systématiquement être entouré d’un try catch

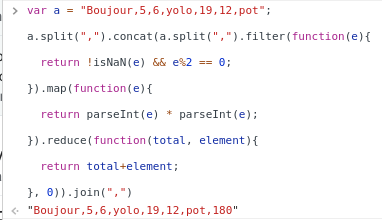
*String invalide:*   
  
  


*Comportement standard:*

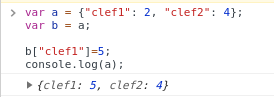


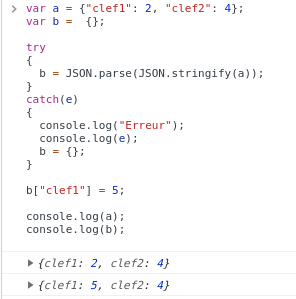
# Exemples complexes

*Faire la somme au carré des chiffres paires de la chaîne de caractère suivante (soit 180) et l’ajouter à la fin de la chaîne suivante:* “Boujour,5,6,yolo,19,12,pot”



*Cloner l’objet ci dessous afin que les opération suivantes ne modifies pas l’objet de base*

**

**