Le Language Javascript (ES5) : Premiers Pas

@Antoine Malpel - 2017

Insérer du code Javascript dans une page HTML

Il existe plusieurs façons d'insérer du code Javascript dans une page HTML.

En ligne (inline)

On place le code Javascript **directement dans le code d'une balise HTML**.

Exemple:

```
<!-- ici le navigateur affichera une alerte lors d'un click sur un bouton. -->
<button onclick="alert('vous avez cliqué')">Cliquez Ici</button>
```

Balise HTML <script>

Afin d'éviter de mélanger du code Javascript la partie HTML, il est également possible de placer le code dans une balise dédiée à cette usage.

Tout ce qui sera contenu entre la balise <script> et la balise de clôture correspondante </script> sera considéré comme du code Javascript.

Exemple:

```
    <script>
```

```
    var anneeDeNaissance;
    var anneeCourante;
    var age;

    age = anneeCourante - anneeDeNaissance;
    alert('Vous avez ' + age + ' ans');

    </script>
```

Tout code Javascript placé au dehors de ces balises sera considéré comme du code HTML.

Exemple:

```
    <div>
        <div>
            <p>Lorem ....</p>
        </div>
    </div>

    <div>
        var a;
        a = 10;
    </div>
```

Ici, au lieu d'excecuter du code Javascript nous verrons le code s'afficher dans la page en clair. L'utilisateur pourrait croire à un complot.

Depuis un fichier externe <script src="fichier.js">

La solution des pros.

Ici on peut isoler totalement le code *JavaScript* dans un fichier qui lui sera propre. On va donc créer un fichier contenant que du *JavaScript* (sans avoir à y mettre la balise *<script>*) puis référencer ce fichier depuis la page HTML avec l'attribut *src* de la balise *script*.

Exemple:

Soit un fichier nommé *demander-age.js* dans le même dossier que le fichier html.

Il faut donner l'extension js au fichier pour que cela fonctionne

```
// Dans le fichier demander-age.js
var age;
age = prompt('age ?');
alert('Donc l\'année prochaine vous aurez ' + ++age);
```

Et dans le code html de notre page

```
<script src="demander-age.js"></script>
```

Ecrire du Javascript

Ecrire du code JavaScript c'est éditer un fichier texte ... en respectant sa syntaxe bien entendu !

Editeur

Pour écrire du javascript il vous faut une éditeur de code de type bloc-note. Un traitement de texte, ou tout autre éditeur WYSIWYG ne convient pas. Vous pouvez également utiliser des outils plus performants tels que notepad++ ou Visual Studio Code qui proposent par exemple la coloration syntaxique. Ces outils permettent de rendre la vie du développeur plus agréable et permette de gagner en productivité.

Sensible la casse

Il est important Bien prendre en compte que le code Javascript est sensible à la casse. Le caractère 'e' minuscule ne vaut pas le caractère 'E' majuscule. De même pour les valeurs la chaine de caractère 'minuscules' est différentes de la valeur 'Minuscule'.

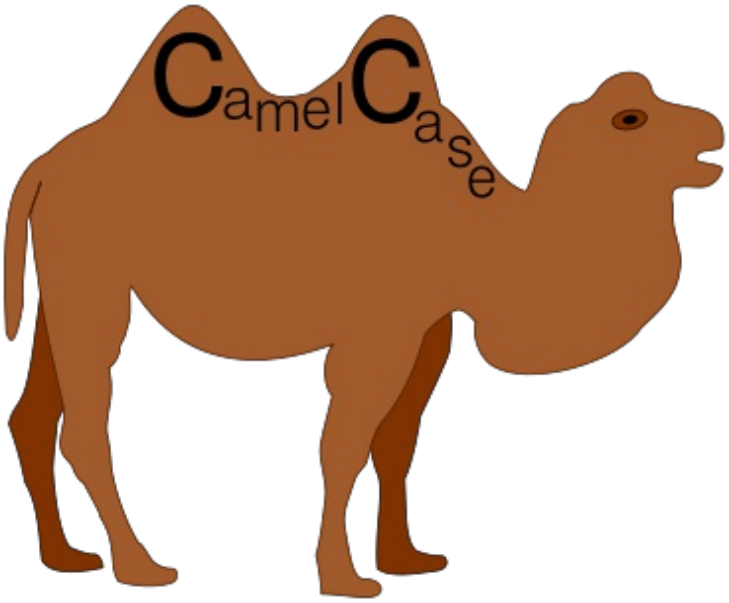
Donc si vous avez une variable nomUtilisateur elle ne sera pas reconnue avec NOMUTILISATEUR ni même nomutilisateur par exemple.

Convention CamelCase

Dans les langages informatique on essaye de prendre des conventions déjà établies, qui permette de répondre à des questions déjà posée. Par exemple, puisque ***javascript*** et sensible à la casse nous avons le choix d'écrire.

```
var estmajeur;
var est_majeur;
var EstMajeur;
var estMajeur;
```

Il est important de choisir une convention, ne pas déclarer un coup MaVariableA puis ma_variable_b car sinon nous devrons nous reposer la question à chaque fois que nous recherchons une variable.



Même si le développeur est un homme (ou une femme, cela arrive de plus en plus!) libre, la communauté javascript prend en général la notation dite CamelCase (lowerCamelCase) et il est plus simple de s'y plier. Ainsi vous verrez des bosses de dromadaire. Un met toujours une minuscule en premier (lowerCamelCase).

Les commentaires

Ecrire un commentaire rendra le code plus lisible et maintenable. Tout ce qui est écrit en commentaire est totalement ignoré par la machine.

Nous sommes donc libres d'écrire sans avoir à respecter une quelconque syntaxe.

Syntaxe:

```
// Commentaire sur une ligne se fait par '//'

/*
* Ici je peux faire une commentaire sur plusieurs ligne.
* Tout ce qui est compris entre '/*' et '*/' sera vu comme un commentaire.
*/
```

Variables

Déclarer une variables

La déclaration d'une variable se fait par le mot clé 'var', suivi du nom que l'on souhaite donner à cette variable.

On ne précise pas le type.

Exemple:

```
var a;
var nom;
var motBeauTableau;
```

le type sera connu à l'

Affectation d'une valeur

L'affectation d'une valeur se faire via l'opérateur '='. On place à gauche la variable et à droite une expression.

Exemple

```
var nom;

// en Euros.
var salaire;
var prime;
var revenus;

nom = 'Penelope';

salaire = 450000;
prime = 95000;
revenus = salaire + prime;
```

Note : on peut affecter une valeur à une variable non déclarée ! On dit alors que la variable a été créée implicitement.

```
    // DANGER !
    variableSortieDeNullePart = "Ce n'est pas conseillé';
```

ATTENTION ! cette variable sera Globale, ce qui peut entrainer des problématiques de maintenabilité et de compréhension du code.

Il est possible de déclarer une variable et de lui affecter une valeur (donc de lui donner un également un type) sur une seule ligne !

```
var age = 32;
```

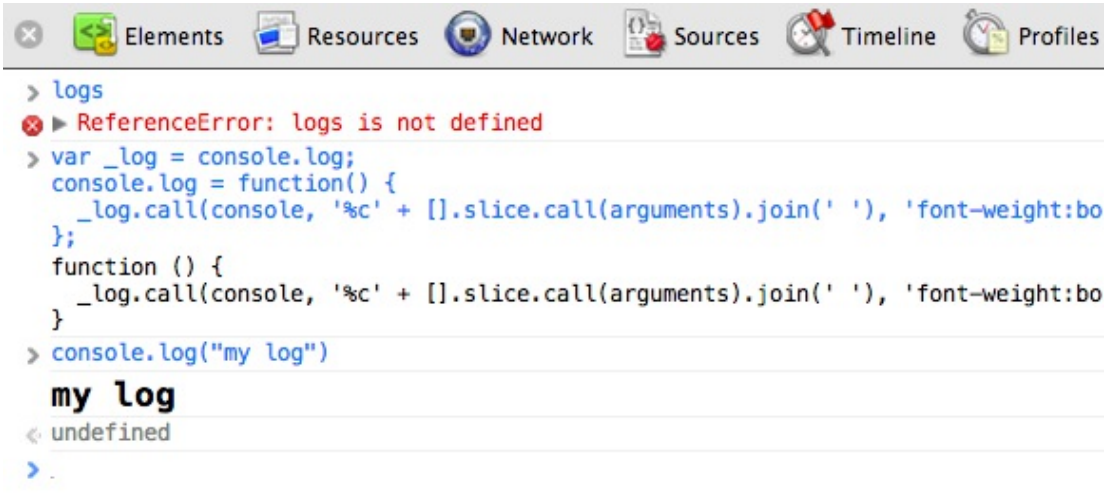
Afficher une variable

window.alert(message)

Affiche le contenu de la variable message dans une boite d'alerte.

```
var age;
age = window.prompt("Quel âge avez vous ?");
window.alert('Donc vous avez bien ' + age + ' ... ok ');
```

console.log(message)



```
var age;
age = window.prompt("Quel âge avez vous ?");
console.log('Il prétent avoir ' + age);
```

Affiche le contenu de variable message dans la fenêtre de débogage du navigateur avec la fonction (Aide pour ouvrir la fenêtre de débogage).

Dans le HTML directement, implique de manipuler la page HTML.

Nous allons apprendre en détail comment faire cela par la suite. N'ayez pas peur mais voici une façon de faire à simple titre d'information.

```
var divNode;
divNode = document.createElement('h2');
divNode.innerText = "Salut C'est JS ici";
document.lastChild.appendChild(divNode);
```

Les différents Type de Variables

Voici les différents types de variable que l'ont rencontre en Javascript.

number

Les valeurs de type nombres telles que 42, -3.24 ou 0

```
var nombreEleve = 17;
```

string

Les chaines de caractères telles que 'coucou', 'longue phrase....' ou même 'a'.
Il faut les délimiter par des simples ou doubles apostrophes.

```
var nom;
var charactere;
var chaineLongue;

nom = 'Arkaitz';
charactere = 'a';
chaineLongue = 'Nous allons apprendre en détail comment faire cela par la
suite.';

nom = Arkaitz       // ERREUR ! ici affecte la variable Arkaitz (qui n'existe
pas ) et non pas la valeur chaine de charactères 'Arkaitz'.
```

boolean

peut prendre la valeur ***true*** ou ***false*** (vrai et faux).

Exemple :

```
var majeur = false;
var riche = true;
```

Les variables sont typées dynamiquement, c'est à dire au moment d'une opération d'affectation de valeur.

```
var nom;            // nom est déclaré mais n'a pas de type.
var age;            // age n'as pas de type

nom = 'Arkaitz';    // nom est maintenant de type string et a pour valeur
'Arkaitz'
age = 32;           // age a maintenant pour type number et pour valeur 32
```

undefined

C'est le type par défaut d'une variable, avant qu'elle reçoive une valeur. Le type de valeur 'undefined' ne peux prendre pour valeur que undefined.

Si vous tentez d'accéder à une variable qui ne contient pas de valeur, elle retournera l'expression *undefined* qui est un est un type à par entière

```
var sansValeur;                  // variable déclarée mais sans affectation
de valeur

console.log(sansValeur);         // undefined
typeof (sansValeur)              // 'undefined' <- avec gillements ici c'est
le type
```

On peut également affecter *undefined* à une variable contenant déjà une valeur. Cela permet de purger une variable de sa valeur.

```
var nb = 4;
typeof(nb);                        // 'number'
console.log('nb vaut ' + nb);      // nb vaut 4
nb = undefined;
typeof(nb)                         // n'est plus défini : 'undefined'
console.log(nb);                   // undefined, comme si nous n'avions
```

```
jamais déclaré ou affecter de valeur sur nb.
```

Connaître le type d'une variable

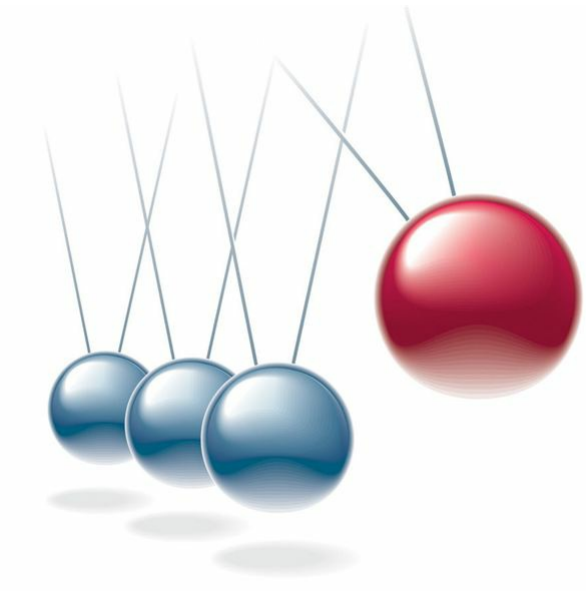
Comme nous l'avons vu le type d'une variable change au moment d'une opération d'affectation de valeur. On peut donc vouloir vérifier le type d'une variable à un moment donné de l'exécution du code. Javascript fourni une instruction dédiée typeof(variable) qui retourne le type sous forme de chaine de caractères.

Exemple :

```
var toto;
var type;
toto = 32 - 43;
console.log('valeur de toto ' + toto);      // -11;
type = typeof(toto);
console.log('type de toto ' + type);        // 'number'

// réaffectation
toto = 'c\'est l'histoire de toto...';
console.log('nouveau type de toto ', typeof(toto));     // toto est devenu de
type 'string'
```

Opérateurs



Les opérateurs tiennent compte du typage.

Opérateur arithmétiques (nombres)

- **+** : number + number donne la somme des deux nombres
- **-** : number - number donne la soustraction
- ***** : number * number multiplie les deux nombres entre eux

- **/** : number / number divise le premier nombre par le deuxième
- **%** : number - number donne le reste de la division entière

```
5 - 3       // donne 2
3 - 5       // donne -2
2.0 / -0.0  // renvoie -Infinity en JavaScript
```

Opérateur sur les chaines de caractères :

- **+** : string + string retourne la concaténation des deux chaines de charactères.

Opérteurs Booléens :

- ***&&*** : condition1 && condition2 ET logique, retourne true si et seulement si les deux conditions sont true, sinon retourne false
- **||** : condition1 || condition2 OU logique, retourne true si au moins une deux conditions est à true, sinon retoune false
- ***!*** : !condition NON logique, retourne true si condition est false et false si la condition est à true

Transtypage (cast)

Explicite

On peut convertir le type d'une valeur, sans garanti de succès, il faut que cela donne sens.

- Boolean(expression) retourne un booléen à partir de expression. Si expression est un nombre retourne vrai si différent de zéro.
- String(expression) retourne la chaine de caractères.
- Number(expression) retourne un nombre.

Implicite

Nous voyons que l'opérateur addition + peut être utiliser avec les types string et number. Si nous mélangeons les types string et number alors l'opérateur + cherchera à transformer la valeur de type string en type number, ***et ce de manière implicite!***. Ce qui peut avoir de fâcheuses conséquences comme nous allons le voir.

```
console.log(2 + '20');        // 202
```

Ici tout fait sens, la chaine de caractères '20' peut être convertie en nombre 20. Donc ~~en douce~~ la machine fait d'abord la conversion de '20' vers 20 puis applique la commande number + number et donnera donc la somme des deux nombres.

```
console.log(2 + Arkaitz)    // NaN
```

Par contre la chaine 'Arkaitz' ne pourra pas être converti en nombre, car cela ne fait pas sens. La conversion de 'Arkaitz' en nombre donnera une valeur particulière : NaN pour n'est pas un nombre (Not a Number).

console.log('Arkaitz' + 2) revient à faire (NaN + 2) qui donnera aub fiinal NaN.

Fonctions

Les fonctions se déclare avec le mot clé function suivi du nom de la fonction puis de parenthèses contenant la liste des nom des arguments séparés par une virgule. Par la suite on retrouve le corps de la fonction entouré par des ***{}***. On retourne la valeur par le mot clé return suivi de l'expression à retourner.

Pour appeler cette fonction on écrit le nom qu'on lui a attribué puis les valeurs que l'ont passe en argument, entre parenthèses.

Exemple :

```
// fonction nommée addition prenant deux paramètres et retournant
// le résultat de l'opérateur + sur ces deux arguments.
function addition(nombreA, nombreB) {
    return nombreA + nombreB;
}

// Appels à la fonction
addition(20, 30);       // somme prendra 50
addition(a, b);         // retournera a + b (espérons que a et b sont des
nombres ou qu'ils soient convertibles).
somme = addition('toto', 32.2)  // retournera 'toto32.2' dans variable Globale
(code poubelle)
```

Les Opérateurs de comparaison

Compare les deux valeurs et retourne un booléen.

- > : Opérateur de supériorité stricte
- < : Opérateur d'infériorité stricte
- >= : Opérateur de supériorité
- <= : Opérateur d'infériorité

Exemple :

```
var nb;
nb = 4;

nb <= 10;     // retourne true
nb < 2;       // retourne false
```

Tests d'égalité

== test d'égalité de valeur

Compare l'égalité de deux valeurs. Retourne un booléen.

Si on compare deux valeur de types distincts, un Transtypage implice aura lieu.

Exemples :

```
('Greta' == 'Greta')        // true
('Chipiron' == 'poisson')   // false
(10 * 10 == 100)            // true
("20" == 20)                // true.  (transtypage)
('Chipiron' == 'chipiron')  // true (différence dans la casse)
```

!== Inégalité

A l'inverse du test d'égalité, donnera vrai si les valeurs sont différentes.
C'est un raccourci vers le !(valA == valB) Retourne un booléen.

Avec deux valeur de types distincts, un Transtypage implice aura lieu.

Exemples :

```
('Greta' != 'Greta')        // false
('Chipiron' != 'poisson')   // true
(10 * 10 != 100)            // true
("20" != 20)                // false.  (transtypage)
```

=== test d'égalité de valeurs et de types

Comme l'opérateur == on teste l'égalité de valeur mais également le type des variables **sans transtypage implicite !!**.

Précédement on a vu que 20 == '20' retournait true car l'interpreteur de code (le nagivateur bien souvent) détecte que les deux varaibles ne sont pas du même type et procède implicitement à un transtypage du nombre 20 vers la valeur string '20'.

Ici le type est tout d'abord comparé. Si les deux types ne sont pas égaux alors il retourne false, sinon un nouveau test à lieu : le test d'égalité de valeur.

Donc contrairement à ce qui se passe si on utilisait == ici 20 === '20' retournera false puisque les types diffèrent (string pour '20' et number pour 20, souvenez vous de typeof).

Exemples :

```
('Greta' === 'Greta')        // true
('Chipiron' === 'poisson')   // false
(10 * 10 === 100)            // true
("20" === 20)                // false.  (différence de type)
```

!== test d'inégalité de valeurs et de types

Retourne true si le type et la valeur des deux varaibles sont identiques. Sinon retourne false.
Raccourci de !(valA === valB).

Exemples :

```
('a' !== 'a')                  // false
(10 !== 20)                    // true
('simone' !== 'SIMONE')        // true
(10 !== 'dix' && 10 !== '10')  // true
```

Les Tableaux array ou []

Les tableaux permettent de stocker un ensemble de valeurs dans une seul variable. On accède à une de ses valeur en précisant le rang de la valeur. Le rang est souvent spécifié par une valeur de type number appelé index.

Syntaxe:

```
// Deux façons déclarer
// Ici tableaux vides.
var tab = [];
var tabBis = new Array();

// avec des valeurs initiales
var tabTier = [1, 2, 3, 5];
// on peut mettre différents type de valeur
var tabMixto = [1, true, tab, {}, 'String'];

// Vide mais en spécifiant le nombre d'éléments.
var tabQuatro = new Array(4);
```

Pour donner le type array à une variable on peut procéder de plusieurs manières.

```
var tableau;

tableau = [];               // affectation d'un talbeau vide
tableau = new Array();      // affectation d'un talbeau vide
```

```
tableau = [1, 2, 3, 4];     // affectation d'un talbeau initialisé avec des
valeurs number/
```

Rappel : les variables n'ont de type qu'au moment de leur affectation. Il en est de même pour chaques éléments du tableau.

Des lors on peut mélanger les type de valeur dans un tableau. **ex:** tableau = [2, 'vélo', true, 4]

Accesseurs

tableau[index] : Pour accèder à un des valeurs on indique son rang. On fait de même pour assigner une valeur.

Attention, en informatique, comme chez les indiens d'Inde, on commence à compter à partir du zéro et non pas à un comme nous le faisons naturellement en Europe.

Pour obtenir le premier élément du talbeau on indique 0 et non pas 1. Pour le quatrième élément 3. Bref on soustrait 1 à la valeur du rang.

Exemple :

```
var temp;
var tableau;

tableau = [1, 2, 3, 4, 5];

// inversion des valeurs du premier et du dernier élément du tableau.
temp = tableau[0];              // copie de la valeur du premier element (1)
dans temp
tableau[0] = tableau[4];        // copie de la valeur (5) vers premier élement
tableau[4] = temp;              // copie de la valeur de temps (1) dans le
dernier element

console.log(tableau);           //  [5, 2, 3, 4, 1]
```

La longeur d'un tableau s'obtient en lisant la propriété (variable calculée incluse dans la variable tableau elle même) length

```
var t;
t = [1, 2, 3, 4, 5];
console.log(t.length);      // 5
```

Ajouter un élément

On passer par la méthode (fonction directment incluse dans la variable tableau) push() La valeur passée en paramètre sera ajouter à la fin du tableau, dont la propriété length sera modifié automatiquement.

Exemple :

```
var tableau;

tableau = [1, 2];
tableau.length;       // 2
tableau.push(3);      // tableau vaut maintenant [1, 2, 3]
tableau.length;       // 3
```

Supprimer un élément

- pop() Méthode de tableau qui supprime le dernier élément
- shift Méthode de tableau qui supprime le premier élément

Exemple : var tableau;

tableau = [1, 2, 3, 5, 6, 7, 8]; tableau.pop(); // tableau vaut maintenant [1, 2, 3, 4, 5, 6, 7] tableau.shift(); // tableau vaut maintenant [2, 3, 4, 5, 6, 7]

Parcours d'un tableau

Exemple :

```
var eleves;
var index;
eleves = ["JB", "Stéphanie", "Joe", "Agurtzane"];

for (index = 0; index < eleves.length ; index = index + 1) {
    alert("bonjour " + eleves[index]);
};
```

Les Objets object ou {}

Vaste sujet que les objets ! Issus du paradigme de programmation orienté objet on les retrouve largement dans le language Javascript.
Heureusement en Javascript le principe est très souple et conmprendre quelques notions de base peut se faire assez aisément.

/!\ Ceci est une légère introduction aux objet en JavaScript, le minimum vital pour survivre dans le mode du JavaScript.

Un objet Javascript est un type de variable conteneur universel dans laquel on pourrat placer librement d'autres variables.

Créer un objet

**Syntaxe : **

```
var objet;

objet = {};
objet = new Object();
```

Les propriétés d'un objet objet.property

Les variables contenues par un objet sont appelées propriétés. Une propriété d'objet n'a pas besoin d'être déclarée, on affecte directement une valeur à la propriété en utilisant son nom précéder d'un point . lui même précéder de la varialbe objet concernée.

**Syntaxe : **

```
variableObj.nomPropriété = expression;
```

Prenons par exemple une voiture.
Pour représenter cette voiture sous forme de variable unique pourrait créer une variable de type object et lui donner des valeur pour les dimensions, la couleur, etc ...

```
var voiture;

voiture = {};
voiture.modèle = 'Mazda 3';
voiture.couleur = 'rouge';
voiture.longueur = '4.5';   // en mètre
```

Libre à nous de modifier, lire, créer à tout moment de nouvelle propriété

```
// (suite du code précédent)
voiture.modèle = 'Peugeot 307';                                    // Ah oui
en fait c'était une peugeot.
voiture.kilométrage = 233045;
voiture.modèle = 'Mazda 3'                                         // Mais
non finalement c'était bien une Mazda 3, je croyais qu'il faisaient que des
piles pourtant.

var modelEtCouleur = voiture.modèle + ' - ' + voiture.couleur;     //
modelEtCouleur donne 'Mazda 3 - rouge'
```

On dit que c'est un objet dynamique, car on peut affecter une valeur librement et à tout moment.

Méthodes d'un objet

De la même manière que les variables contenues dans un objet devienent des proprités, contenir une fonction dans un variable de type objet devient une méthode de cet objet.

Function

Je ne vous ai pas tout dit sur les types de varaibles. Nous allons un autre type de variable: Les fonctions !!! Oui, en JavaScript, on peut placer une fonction dans une variable. Une telle variable prend alors le type Function.

```
var maFonction;

maFonction = function () {
    console.log('Hey je suis une fonction a exécuter');
}

maFonction();           // appel de la fonction affiche message 'Hey je
suis...' dans la console du navigateur
maFonction;             // sans les paranthèse la fonction ne sera pas
appelée.
typeof(maFonction)      // affiche ('Function') c'est bien un type de variable
!
```

Cela peut vous paraître étrange mais vous verrez que cela apporte beaucoup. Notamment le fait qu'on puisse placer dans notre objet un variable de ce type ! Une telle variable est appelée une **Méthode**.

Chaque méthode peut faire référence à l'objet lui même en utulisant le mot clé this. Pour accéder aux autres méthodes et variables dans une méthode on fait donc this.propriété ou this.methode()

Création d'une méthode

Syntaxe : objet.nomMéthode = function () { /* corps de la fonction */ };

Exemple :

```
var objet;

objet = {};                             // création et affectation d'un objet
objet.nom = 'Chimblick';                // ... de la proprité string nom.


objet.afficherNom = function () {
    console.log('je suis un objet nommé ' + this.name);
    // this.name ici correspond à 'Chimblick'.
};
// important il faut le ;
```

Attention, contrairement au fonction il est ici nécessaire de mettre un ; apres le délimiteur de fin de corps } de la fonction

Appel de la méthode

On place des parenthèses après l'accesseur, c'est à dire le nom de la méthode.

Syntaxe : objet.nomMéthode()

Le code de la ne sera pas executé si vous ommettez les paranthèses.

Exemple :

```
// suite du code précédent
objet.afficherNom();        // affiche 'Chimblick'
objet.nom = 'Bitonio';
objet.afficherNom();        // affiche 'bitonio'
```