Vue.js : Le Javascript que vous devez connaître

Vue.js est une librairie Javascript qui permet de créer des interface web interactives. Il existe de très nombreuses librairie qui proposent la même chose dont la plus célèbre est AngularJS. Pour avoir une idée de la richesse de l'offre vous pouvez jeter un coup d'œil sur cette page. Alors étant donné cette prolifération pourquoi utiliser particulièrement vue.js ?

Si vous avez déjà utilisé AngularJS vous devez connaître la difficulté de prise en main qui est nécessaire, mais on a à disposition une grande richesse fonctionnelle. C'est le prix à payer. Sur l'autre versant on a des librairies beaucoup plus simples d'accès mais qui ont au final peu de choses à offrir. Vue.js se situe à la confluence de ces deux mondes en alliant avec efficacité simplicité et richesse. D'autre part il est très souple et n'impose pas un certain type d'organisation.

Je vous propose donc de découvrir cette librairie. Dans la première partie de cette série d'articles nous verrons l'essentiel de ce que la librairie nous offre de base, et vous verrez qu'il y a déjà profusion de possibilités. Dans la deuxième partie nous verrons comment enrichir la librairie selon nos besoins. Cette série sera loin d'épuiser le sujet mais il vous mettra le pied à l'étrier et, je l'espère, vous donnera envie d'utiliser cette librairie très prometteuse.

La première chose que nous allons faire est un peu de Javascript. Si vous maîtrisez déjà ce langage vous pouvez sauter cet article et passer directement au suivant et vous aurez toujours la possibilité d'y revenir en cas de besoin. Si ce n'est pas le cas prenez le temps de lire. Vous n'avez pas besoin d'être un gourou de Javascript pour utiliser vue.js mais vous devez en maîtriser les bases.

Javascript ?

Javascript est un langage dynamique orienté objet à prototype, ce qui signifie qu'il n'y a pas de classe. Il est essentiellement utilisé côté client pour offrir de l'interactivité mais il est aussi de plus en plus utilisé côté serveur, par exemple avec node.js.

Lorsqu'on compare Javascript à d'autres langages comme C# ou Java la première impression est celle de la simplicité. Il est vrai que ce langage se laisse facilement aborder et ne pose pas trop de problème à condition de ne pas trop le pousser dans ses retranchements, ce que nous ne ferons pas dans cet article. En effet Javascript est loin d'être aussi simple qu'on peut le penser.

Inclusion de Javascript

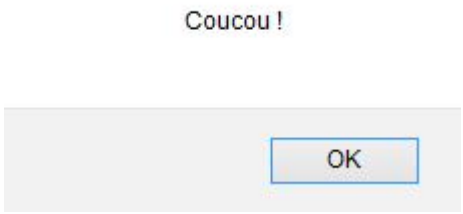
Comment est-ce qu'on inclut du code javascript dans une page HTML ? On peut le faire de deux façons.

Inclusion en ligne

La façon la plus simple pour inclure du Javascript est d'ajouter le code directement dans la page avec la balise <script> . Voici un exemple :

```
<!DOCTYPE html>
<html>
        <head>
                <title>JavaScript</title>
        </head>
        <body>
        <script>alert("Coucou !")</script>
        </body>
</html>
```

La méthode alert permet d'afficher une petite fenêtre avec un texte. Lorsque cette page se charge il doit apparaître ceci :



Mais ce n’est pas le meilleur endroit pour placer du Javascript. Il est plutôt recommandé d’adopter la seconde méthode que nous allons voir à présent.

Inclusion de la référence

Considérons le même cas que précédemment en changeant un peu le code de la page :

```
<!DOCTYPE html>
<html>
        <head>
                <title>JavaScript</title>
                <script src="coucou.js"></script>
        </head>
        <body></body>
</html>
```

Cette fois dans la balise script on référence le fichier coucou.js situé dans le même dossier. Voici le code de ce fichier :

```
alert("Coucou !");
```

Le fonctionnement sera exactement le même mais cette fois on a bien séparé le HTML et le Javascript.

Pour la clarté du code dans ces articles je serai amené à utiliser la première méthode.

Les instructions

Fondamentalement un script Javascript est constitué d’une suite d’instructions :

```
<!DOCTYPE html>
<html>
        <head>
                <title>JavaScript</title>
        </head>
        <body>
        <script>
                var a = 2;
                var b = 3;
                document.write(a + b);
        </script>
        </body>
</html>
```

La partie Javascript est constituée de 3 expressions :

1. on initialise la variable « a » et on lui affecte la valeur 2

2. on initialise la variable « a » et on lui affecte la valeur 3

3. on demande l'affichage de la somme « a + b »

Si vous ouvrez cette page vous verrez s'afficher 5.

Le point-virgule final des instructions n'est pas obligatoire mais vivement conseillé pour la clarté du code.

Évidemment Javascript dispose d'instructions conditionnelles qui permettent une exécution plus souple qu'une suite d'instructions figées comme nous le verrons bientôt !

Les commentaires

Il est très utile de commenter le code. Bien souvent on se comprend lorsqu'on l'écrit mais quelques mois plus tard lorsqu'on y revient on ne retrouve plus la logique qui le sous-tend. C'est encore pire lorsqu'une autre personne doit intervenir dessus. Donc commenter son code est une saine disposition qui devrait être systématique. Javascript propose les deux types classiques de commentaires comme utilisés pour C++, PHP et d'autres grands

classiques :

```
<!DOCTYPE html>
<html>
        <head>
                <title>JavaScript</title>
        </head>
        <body>
        <script>
                /*Le code suivant est destiné à illustrer
                l'utilisation des instructions en Javascript*/
                var a = 2;
                var b = 3;
                 // Ici on envoie à l'affichage le résultat de la
somme des deux variables
                document.write(a + b);
        </script>
        </body>
</html>
```

La première syntaxe permet d’englober plusieurs lignes alors que la seconde ne concerne que le texte disposé sur la même ligne.

Types et variables

Les variables

Les variables sont en quelque sorte des boîtes dans lesquelles on place des valeurs et dans lesquelles on peut ensuite les retrouver. On a vu l’utilisation de deux variables dans le code précédent avec ces deux instructions :

```
var a = 2;
var b = 3;
```

On utilise le mot clé var et un identifiant (le nom de la variable).

Dans le code précédent on a groupé dans la même instruction la déclaration et l’affectation des variables. On peut le faire en deux étapes :

```
// Déclaration
var a, b;
// Assignation
a = 2;
b = 3;
```

Une variable a une certaine portée. Lorsqu'on la déclare en dehors d'une fonction elle est globale, donc utilisable partout. Lorsqu'on la déclare dans une fonction elle devient locale à cette fonction, donc inconnue en dehors de la fonction.

Les types

Les variables peuvent contenir plusieurs types de valeurs : nombre entier, nombre à virgule, chaîne de caractères… Javascript est faiblement typé, autrement dit on peut mettre ce que l'on veut dans une variable, comme avec PHP. Javascript va donc utiliser un de ses types internes selon la valeur d'affectation. Cela offre de la souplesse mais il faut rester vigilant pour éviter les surprises !

Le type booléen (boolean)

C'est le type le plus simple puisqu'on a seulement deux valeurs possibles : true ou false. Voici un exemple :

```
<!DOCTYPE html>
<html>
	<head>
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		var a = true;
		var b = false;
		 document.write('La variable "a" est du type ' +
typeof(a) + ' et a pour valeur ' + a + '<br>');
		 document.write('La variable "b" est du type ' +
typeof(b) + ' et a pour valeur ' + b);
	</script>
	</body>
</html>
```

Le résultat est le suivant :

> *La variable « a » est du type boolean et a pour valeur true*
> *La variable « b » est du type boolean et a pour valeur false*

On a déclaré deux variables en leur affectant respectivement les valeurs true et false. La méthode typeof de javascript permet de connaître le type de la variable. On voit que Javascript a bien déterminé le type booléen.

Le type chaîne de caractères (string)

Une chaîne de caractères est tout simplement… une suite de caractères quelconques. Regardez le code suivant :

```
<!DOCTYPE html>
<html>
	<head>
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		var nom = "Dupont";
		var prenom = 'Pierre';
		 document.write('La variable "nom" est du type ' +
typeof(nom) + ' et a pour valeur ' + nom + '<br>');
		document.write('La variable "prenom" est du type '
+ typeof(prenom) + ' et a pour valeur ' + prenom);
	</script>
	</body>
</html>
```

Avec ce résultat :

> *La variable « nom » est du type string et a pour valeur Dupont*
> *La variable « prenom » est du type string et a pour valeur Pierre*

On voit qu’on peut utiliser indifféremment des guillemets ou des apostrophes. Notez aussi au passage qu’on ajoute (concaténation) deux chaînes avec l’opérateur « + ».

Le type nombre (number)

On a déjà vu ci-dessus l'affectation de nombres. Regardez ce code :

```
<!DOCTYPE html>
<html>
	<head>
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		var a = 28;
		var b = 12.99;
		 document.write('La variable "a" est du type ' +
typeof(a) + ' et a pour valeur ' + a + '<br>');
		 document.write('La variable "b" est du type ' +
typeof(b) + ' et a pour valeur ' + b + '<br>');
		document.write('La somme "a + b" est ' + (a + b));
	</script>
	</body>
</html>
```

Avec ce résultat :

La variable « a » est du type number et a pour valeur 28
La variable « b » est du type number et a pour valeur 12.99
La somme « a + b » est 40.99

Null et undefined

Il nous reste à voir deux cas importants. regardez ce code :

```
<!DOCTYPE html>
<html>
	<head>
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		var a;
		var b = null;
		 document.write('La variable "a" est du type ' +
```

```
typeof(a) + ' et a pour valeur ' + a + '<br>');
                document.write('La variable "b" est du type ' +
typeof(b) + ' et a pour valeur ' + b + '<br>');
        </script>
        </body>
</html>
```

Avec ce résultat :

> *La variable « a » est du type undefined et a pour valeur undefined*
> *La variable « b » est du type object et a pour valeur null*

Le type undefined est simple, la variable est déclarée mais ne contient aucune valeur.

Pour ce qui concerne la deuxième variable on voit que son type est object, nous verrons bientôt ce type un peu particulier. Sa valeur est null, ce qui signifie en fait que l'on ne veut pas qu'elle ait une valeur.

Les opérateurs

Javascript possède tous les opérateurs classiques.

Arithmétique

Voilà une peu d'arithmétique :

```
<!DOCTYPE html>
<html>
        <head>
                <title>JavaScript</title>
        </head>
        <body>
        <script>
                document.write('5 + 6 = ' + (5 + 6) + '<br>');
                document.write('5 - 6 = ' + (5 - 6) + '<br>');
                document.write('5 * 6 = ' + (5 * 6) + '<br>');
                document.write('5 / 6 = ' + (5 / 6) + '<br>');
                document.write('5 % 5 = ' + (5 % 5) + '<br>');
```

```
                document.write('6 % 5 = ' + (6 % 5) + '<br>');
        </script>
        </body>
</html>
```

Avec ce résultat :

```
5 + 6 = 11
5 – 6 = -1
5 * 6 = 30
5 / 6 = 0.8333333333333334
5 % 5 = 0
6 % 5 = 1
```

La seule difficulté réside dans l’opérateur « % » qui est tout simplement le reste de la division.

Comparaisons

Passons aux comparaisons :

```
<!DOCTYPE html>
<html>
        <head>
                <title>JavaScript</title>
        </head>
        <body>
        <script>
                document.write('5 > 6 = ' + (5 > 6) + '<br>');
                document.write('5 < 6 = ' + (5 < 6) + '<br>');
                document.write('5 >= 5 = ' + (5 >= 5) + '<br>');
                document.write('5 <= 5 = ' + (5 <= 5) + '<br>');
                document.write('5 == 5 = ' + (5 == 5) + '<br>');
                document.write('5 == 6 = ' + (5 == 6) + '<br>');
                document.write('6 != 5 = ' + (6 != 5) + '<br>');
                document.write('6 != 6 = ' + (6 != 6) + '<br>');
        </script>
        </body>
</html>
```

Avec ce résultat :

5 > 6 = false
5 < 6 = true
5 >= 5 = true
5 <= 5 = true
5 == 5 = true
5 == 6 = false
6 != 5 = true
6 != 6 = false

Rien de bien compliqué là dedans. Voyons un cas un peu plus délicat :

```
<!DOCTYPE html>
<html>
	<head>
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		var a = 5;
		var b = '5';
		document.write('a == b = ' + (a == b) + '<br>');
		document.write('a === b = ' + (a === b) + '<br>');
		document.write('a != b = ' + (a != b) + '<br>');
		document.write('a !== b = ' + (a !== b) + '<br>');
	</script>
	</body>
</html>
```

Avec ce résultat :

a == b = true
a === b = false
a != b = false
a !== b = true

L’opérateur triple (égalité stricte ou inégalité stricte)) vérifie le type en plus de la valeur, ce que ne fait pas le double opérateur. Il faut se méfier de ces comparaisons et utiliser les opérateurs « === » et « !== » toutes les fois où c’est possible.

Post et pré_incrémentation

Considérez ce code :

```
<!DOCTYPE html>
<html>
        <head>
                <title>JavaScript</title>
        </head>
        <body>
        <script>
                var a = 5;
                document.write('a = 5' + '<br>');
                document.write('++a = ' + ++a + '<br>');
                a = 5;
                document.write('--a = ' + --a + '<br>');
                a = 5;
                document.write('a++ = ' + a++ + '<br>');
                a = 5;
                document.write('a-- = ' + a-- + '<br>');
                a = 5;
                document.write('4 + ++a = ' + (4 + ++a) + '<br>');
                a = 5;
                document.write('4 + a++ = ' + (4 + a++) + '<br>');
                a = 5;
                document.write('4 + --a = ' + (4 + --a) + '<br>');
                a = 5;
                document.write('4 + a-- = ' + (4 + a--) + '<br>');
        </script>
        </body>
</html>
```

Avec ce résultat :

a = 5
++a = 6
–a = 6
a++ = 5
a– = 5
4 + ++a = 10
4 + a++ = 9
4 + –a = 8

4 + a– = 9

L’opérateur « ++ » ajoute 1 et l’opérateur « – » enlève 1 à la variable. On parle d’incrémentation et de décrémentation.

Lorsque l’incrémentation (ou la décrémentation) est avant la variable on l’applique avant l’opérateur et s’il est situé après on l’applique ensuite. Ce n’est pas clair ? Alors analysez le code !

Les boucles et instructions conditionnelles

On a vu qu’un script Javascript est une suite d’instructions. Il est souvent nécessaire de rompre la séquentialité pour effecteur des boucles, des sauts… voyons comment faire.

Les boucles

Regardez ce code :

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		for (var i = 0; i < 10; ++i) {
			document.write('i = ' + i + '<br>');
		}
	</script>
	</body>
</html>
```

Avec ce résultat :

i = 0

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
```

L’instruction for permet d’itérer en fixant une limite, ici i < 10.

On peut aboutir au même résultat en utilisant l’instruction while :

```
var i = 0;
while (i < 10) {
        document.write('i = ' + i + '<br>');
        ++i
}
```

Il existe aussi l’instruction do...while où le test est effectué à la fin :

```
var i = 0;
do {
        document.write('i = ' + i + '<br>');
} while(++i < 10)
```

Remarquez que cette fois j’ai utilisé l’opérateur d’incrémentation.

Les conditions

Regardez ce code :

```
var age = 20;
if(age > 18) {
        document.write('Vous êtes majeur');
}
```

L’instruction if permet de définir du code conditionnel. Si age > 18 est true alors le code inclus entre les accolades (on appelle ceci un bloc de code) est exécuté, sinon on continue avec la suite.

On peut prévoir une condition alternative :

```
var age = 14;
if(age < 18) {
        document.write('Vous êtes mineur');
} else {
        document.write('Vous êtes majeur');
}
```

Dans ce cas on a affaire à un mineur. On peut aussi enchaîner les conditions :

```
var age = 14;
if(age < 10) {
        document.write('Vous êtes de la première décade');
} else if(age < 20) {
        document.write('Vous êtes de la deuxième décade');
} else if(age < 30) {
        document.write('Vous êtes de la troisième décade');
} else {
          document.write('Vous êtes au-delà de la troisième
décade');
}
```

Il existe aussi l’instruction switch :

```
var count = 3;
switch(count) {
        case(1): document.write('Vous êtes au départ');
                break;
        case(2): document.write('Vous êtes bien parti');
                break;
        case(3): document.write('Vous êtes presque arrivé');
                break;
        default: document.write('Vous avez fini');
}
```

Avec ce résultat :

Vous êtes presque arrivé

Les fonctions

Une fonction est un bloc d'instructions qui est défini à un endroit mais peut être exécuté ailleurs autant de fois que l'on veut. Regardez cet exemple :

```
<!DOCTYPE html>
<html>
	<head>
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		function ma_fonction() {
			document.write("Coucou !" + '<br>');
		}
		ma_fonction();
		ma_fonction();
	</script>
	</body>
</html>
```

Avec ce résultat :

Coucou !
Coucou !

Une fonction se définit avec le mot clé function. On précise ensuite le nom de la fonction, on place des parenthèses dont nous allons voir bientôt l'intérêt et enfin on place le bloc de code entre des accolades. Pour l'appeler on utilise le nom de la fonction suivi des parenthèses.

Voici un autre exemple :

```
function ma_fonction(a, b) {
	return 2 * (a + b);
}
```

```
document.write("Le résultat est : " + ma_fonction(3, 4));
```

Avec le résultat :

```
Le résultat est : 14
```

On appelle a et b les arguments de la fonction. Lorsqu'on appelle la fonction il faut transmettre une valeur pour les arguments.

On peut aussi déclarer une variable pour contenir une fonction. Regardez ce code :

```
var action = function (a, b) {
        return 2 * (a + b);
}
document.write("Le résultat est : " + action(3, 4));
```

On obtient ainsi le même résultat, alors quel intérêt ? Dans ce cas aucun mais le fait d'avoir une fonction dans une variable permet de la passer comme argument d'une autre fonction :

```
var somme = function (a, b) {
        return a + b;
}
var produit = function (a, b) {
        return a * b;
}
function resultat(x, a, b) {
        document.write("Le résultat est : " + x(a, b) + '<br>');
}
resultat(somme, 3, 4);
resultat(produit, 3, 4);
```

Avec cet affichage :

```
Le résultat est : 7
Le résultat est : 12
```

On passe une fonction comme premier argument de resultat.

Les fermetures (closures)

Voyons maintenant quelque chose de plus délicat avec les fermetures (closures). Dans un premier temps regardez ce code :

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		function ajouteDouble(a, b) {
			function double(n) {
				return 2 * n;
			}
			return double(a) + double(b);
		}
		    document.write("Le résultat est : " +
ajouteDouble(3, 4) + '<br>');
	</script>
	</body>
</html>
```

Le rendu est :

Le résultat est : 14

On a une fonction dans une fonction. Quelle curieuse idée ! On aurait pu avoir une fonction double au même niveau que ajouteDouble et ça fonctionnerait de la même façon. La différence est que la fonction incluse ne peut pas être atteinte en dehors de la fonction englobante. Elle a accès aux variables de la fonction englobante mais l’inverse n’est pas vrai.

Voici un autre exemple pour aller plus loin :

```
function multiplier(a) {
	return function(b) {
		return a * b;
	}
```

```
}
var multipliePar5 = multiplier(5);
var multipliePar6 = multiplier(6);
document.write("Le résultat est : " + multipliePar5(4) + '<br>');
document.write("Le résultat est : " + multipliePar6(4) + '<br>');
```

Ce qui donne :

Le résultat est : 20
Le résultat est : 24

On crée des fonctions à partir de multiplier. Ces fonctions sont des fermetures (closures), elles conservent le contexte de leur création et elle deviennent accessible en dehors de la fonction englobante. Par exemple multipliePar5 se rappelle qu'on lui a transmis la valeur 5 et on peut l'utiliser n'importe où.

Les fermetures sont beaucoup utilisées dans les librairies Javascript.

Les tableaux

Les tableaux servent à mémoriser plusieurs valeurs dans une seule variable, peu importe leur type. Voici comment créer et parcourir un tableau :

```
<!DOCTYPE html>
<html>
	<head>
		<meta charset="utf-8">
		<title>JavaScript</title>
	</head>
	<body>
	<script>
		var tableau = [];
		tableau[0] = "Pierre";
		tableau[1] = "Paul";
		tableau[2] = "Arthur";
		for(var i = 0; i < tableau.length; ++i) {
			document.write("Le prénom à l'index " + i
+ ' est ' + tableau[i] + '<br>');
```

```
                }
        </script>
        </body>
</html>
```

Avec ce résultat :

Le prénom à l'index 0 est Pierre
Le prénom à l'index 1 est Paul
Le prénom à l'index 2 est Arthur

On peut initialiser directement les valeurs d'un tableau dans les accolades. Et on peut évidemment modifier les valeurs d'un tableau en utilisant l'index :

```
var tableau = ["Pierre", "Paul", "Arthur"];
tableau[0] = "Simon";
for(var i = 0; i < tableau.length; ++i) {
         document.write("Le prénom à l'index " + i + ' est ' +
tableau[i] + '<br>');
}
```

Ce qui donne :

Le prénom à l'index 0 est Simon
Le prénom à l'index 1 est Paul
Le prénom à l'index 2 est Arthur

La méthode length renvoie l'indice le plus grand + 1. On n'est donc pas vraiment sûrs d'obtenir ainsi le nombre total d'éléments. Voici une méthode plus sûre :

```
var tableau = ["Pierre", "Paul", "Arthur"];
tableau.forEach(function(valeur, index, tableau) {
        document.write("Le prénom à l'index " + index + ' est ' +
valeur + '<br>');
});
```

Vous obtenez :

Le type est object

Nous allons voir juste après les objets et ce que je vais en dire concerne donc aussi les tableaux. La différence c'est que les tableaux ont une indexation numérique alors que les objets ont une indexation nominale.

Les objets

J'ai dit plus haut que Javascript est orienté « objet ». Nous allons voir maintenant ce que ça signifie. C'est un point important de ce langage qui doit bien être compris.

Créer un objet

Voici un premier exemple où nous allons créer un objet :

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>JavaScript</title>
        </head>
        <body>
        <script>
                var mon_objet = {};
                mon_objet.nom = "Dupont";
                mon_objet.prenom = "Pierre";
                 document.write('Mon nom est ' + mon_objet.nom +
'<br>');
                      document.write('Mon prénom est ' +
mon_objet.prenom + '<br>');
        </script>
        </body>
</html>
```

Avec ce résultat :

Mon nom est Dupont
Mon prénom est Pierre

Regardez la syntaxe utilisée. On crée l'objet avec des accolades

vides. Ensuite on déclare des propriétés auxquelles on affecte des valeurs.

Cette façon de créer un objet est nommée syntaxe littérale. On peut aussi le faire de façon plus classique avec le mot clé new :

```
var mon_objet = new Object();
```

Voici une autre syntaxe très utilisée pour aboutir au même résultat :

```
var mon_objet = {
        nom: "Dupont",
        prenom: "Pierre"
};
document.write('Mon nom est ' + mon_objet.nom + '<br>');
document.write('Mon prénom est ' + mon_objet.prenom + '<br>');
```

Cette fois on déclare et affecte les propriétés directement entre les accolades. Remarquez l'utilisation du double point pour l'affectation dans ce cas.

Les propriétés

On peut évidemment changer la valeur d'une propriété :

```
var mon_objet = {
        nom: "Dupont",
        prenom: "Pierre"
};
mon_objet.nom = "Durand";
document.write('Mon nom est ' + mon_objet.nom + '<br>');
document.write('Mon prénom est ' + mon_objet.prenom + '<br>');
```

Avec ce résultat :

> *Mon nom est Durand*
> *Mon prénom est Pierre*

Voici une autre syntaxe pour un résultat identique :

```
var mon_objet = {
        nom: "Dupont",
```

```
        prenom: "Pierre"
};
mon_objet["nom"] = "Durand";
document.write('Mon nom est ' + mon_objet.nom + '<br>');
document.write('Mon prénom est ' + mon_objet.prenom + '<br>');
```

On peut donc au choix utiliser la syntaxe avec le point ou sous la forme de tableau, dans ce cas on place le nom de la propriété entre guillemets (ou apostrophes). Cette seconde syntaxe permet une approche dynamique. Regardez ce code :

```
var mon_objet = {nom: "Dupont"};
var propriete = 'prenom';
mon_objet[propriete] = "Pierre";
document.write('Mon nom est ' + mon_objet.nom + '<br>');
document.write('Mon prénom est ' + mon_objet.prenom + '<br>');
```

Le nom de la propriété est devenu dynamique, ce qui ouvre pas mal de perspectives de codage !

Les méthodes

Les objets ont des propriétés mais ils peuvent aussi avoir des comportements, il suffit de leur ajouter des méthodes. Regardez ce code :

```
<!DOCTYPE html>
<html>
        <head>
                <meta charset="utf-8">
                <title>JavaScript</title>
        </head>
        <body>
        <script>
                var mon_objet = {
                        nom: "Durand",
                        prenom: "Pierre",
                        identite: function () {
                                 document.write('Mon nom est ' +
this.nom + '<br>');
                                document.write('Mon prénom est ' +
this.prenom + '<br>');
                        }
```

```
            };
            mon_objet.identite();
        </script>
        </body>
</html>
```

Avec ce résultat :

> *Mon nom est Durand*
> *Mon prénom est Pierre*

On a défini une méthode identite qui est chargée d'afficher le nom et le prénom. Remarquez l'utilisation du mot clé this qui sert à référencer l'objet en cours. L'appel de la méthode se fait simplement avec la syntaxe avec le point.

En résumé

- Javascript est un langage dynamique, orienté objet.
- Javascript dispose de types et d'opérateurs.
- Javascript permet de créer des variables, des fonctions et des objets.