JavaScript, pour aller plus loin

David Roche lycée G Fichet (Bonneville Haute-Savoie)

v140413

Notes de l'auteur :

- Ce document est "la suite" du document "Les bases de JavaScript".
- Ce document n'est pas une "bible" du JavaScript; il a uniquement pour but de permettre aux élèves une première approche (en autonomie) de ce langage. Ce document ne prétend pas se substituer au travail en classe avec l'enseignant.
- Il est sans doute hors programme ISN, mais peut permettre à certains élèves "d'aller plus loin".
- Pour toutes remarques et commentaires, n'hésitez pas à me contacter : informatiqueaulycee@gmail.com

JavaScript : un langage orienté objet

JavaScript est un langage orienté objet, nous allons donc avoir à manipuler des "objets informatiques". Les objets informatiques ressemblent beaucoup aux objets de la vie de tous les jours, c'est d'ailleurs un peu l'intérêt de la chose.

Mais avant d'aller plus loin, faisons une parenthèse. Si parmi vous certains ont déjà eu l'occasion de rencontrer cette notion d'objet, notamment en travaillant sur des langages comme le Java ou le C++, il important que vous sachiez que JavaScript est un langage orienté objet par prototype. Ce mot prototype change beaucoup de choses, je vous invite donc à "oublier" (provisoirement) ce que vous avez déjà appris sur les objets (classe, instance,...), nous allons tout reprendre à zéro.

En Javascript, un objet est un conteneur qui contient des propriétés ! Qu'est-ce que des propriétés ?

Cela peut être beaucoup de choses. N'ayant pas l'intention de faire de vous "les rois de la programmation orientée objet en JavaScript" (j'en serai bien incapable), mais uniquement de vous donner les bases nécessaires pour poursuivre votre découverte de ce langage, nous allons nous limiter à 3 choses :

- une propriété peut-être un attribut, pour faire simple, un attribut est une variable propre à un objet.
- Une propriété peut-être une méthode, une méthode est une fonction qui agit sur notre objet.
- Une propriété peut-être un autre objet (nous ne verrons pas d'exemple ici, il faut juste que vous sachiez que cela existe).

Prenons un objet de la vie courante et "transformons-le" en objet JavaScript :

Une voiture a une marque et une couleur (entre autres) et bien "marque" et "couleur" seront des attributs de notre objet JavaScript voiture.

Une voiture peut accélérer, freiner, tourner..... et bien "freiner", "accélérer" et "tourner" seront des méthodes de notre objet JavaScript voiture.

Comment "fabrique"-t-on un objet ?

Il existe plusieurs façons de créer un objet, mais, toujours dans un souci d'aller au plus rapide, je ne vais vous en présenter qu'une (celle que nous utiliserons par la suite !). Avant de pouvoir créer l'objet en tant que tel, nous allons devoir construire un moule. À partir de ce moule, nous pourrons fabriquer autant d'objets que nécessaire. En POO (programmation orientée objet), ce moule s'appelle un constructeur. Un constructeur est une fonction qui porte le même nom que votre "classe" (dans notre cas Voiture). Le nom du constructeur doit commencer par une majuscule.

Un exemple avec, pour commencer, uniquement des attributs (nous verrons les méthodes plus loin)

exemple 1.1

```
function Voiture (marque, couleur) {
            this.marque = marque;
            this.couleur = couleur ;

}
```

Vous pouvez constater que notre fonction "Voiture" (notre constructeur d'objet de type voiture) peut posséder des arguments (marque et couleur). Ces arguments seront renseignés au moment de la création de nos objets voitures (souvenez-vous que pour l'instant nous avons uniquement fabriqué le moule).
Passons maintenant au contenu de ce constructeur. Vous avez sans doute remarqué le "this.couleur" et le "this.marque".

"this" est un mot clé très important en POO, il désigne l'objet courant, celui que l'on est en train d'utiliser. Oui, je sais, c'est un peu compliqué, nous reviendrons dessus un peu plus tard.
Maintenant que notre "moule" (constructeur) est prêt, passons à la création de notre premier objet.

exemple 1.2

```
function Voiture (marque, couleur) {
             this.marque = marque;
             this.couleur = couleur ;

}
var voit_1 = new Voiture ("Fiat", "rouge");
```

Nous avons créé une "variable" (voit_1) contenant un objet de type Voiture, nous avons donc créé un objet voit_1 qui est de type voiture. Pour créer cet objet, nous avons utilisé le mot clé "new" suivi du nom du constructeur (Voiture). Comme pour n'importe quelle fonction, les paramètres " Fiat" et "rouge" sont mis dans la parenthèse qui suit le nom du constructeur.

Nous pouvons créer un deuxième objet de type voiture :

exemple 1.3

```
function Voiture (marque, couleur) {
             this.marque = marque;
             this.couleur = couleur ;

}
var voit_1 = new Voiture ("Fiat", "rouge");
var voit_2 = new Voiture ("Renault", "verte");
```

Nous avons maintenant 2 "voitures" à notre disposition : une de marque Fiat et de couleur rouge et une de marque Renault et de couleur verte.

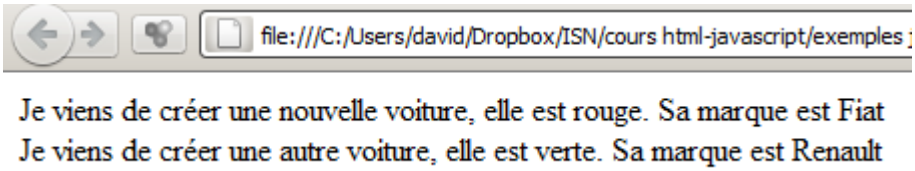
Avoir créé ces 2 objets c'est bien, mais comment les utiliser ?

Il est relativement simple d'accéder aux attributs d'un objet :
nom_de_l'objet.nom_de_l'attribut
La notation pointée (c'est son nom) n'est pas la seule possibilité, mais c'est la seule que nous verrons ici.
Dans notre exemple pour accéder à l'attribut couleur de l'objet voit_1, il faudra écrire :
voit_1.couleur
de même, pour accéder à l'attribut marque de l'objet voit_2 :
voit_2.marque
Voici un exemple :

exemple 1.4

```
function Voiture (marque, couleur) {
             this.marque = marque;
             this.couleur = couleur ;
}
var voit_1 = new Voiture ("Fiat", "rouge");
document.write("Je viens de créer une nouvelle voiture, elle est ",voit_1.couleur,". Sa
marque est ", voit_1.marque,"<br />");
var voit_2 = new Voiture ("Renault", "verte");
document.write("Je viens de créer une nouvelle voiture, elle est ",voit_2.couleur,". Sa
marque est ", voit_2.marque);
```

résultat 1.4



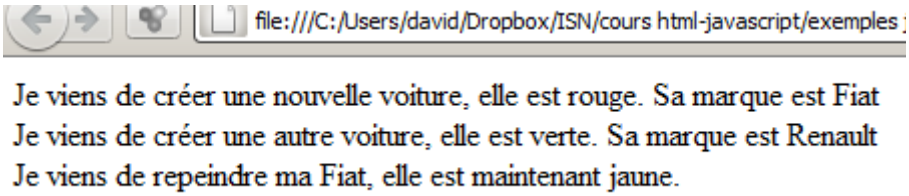
Il est bien évidemment possible de modifier l'attribut d'un objet (comme n'importe quelle variable) :

exemple 1.5

```
function Voiture (marque, couleur) {
             this.marque = marque;
             this.couleur = couleur ;

}
var voit_1 = new Voiture ("Fiat", "rouge");
document.write("Je viens de créer une nouvelle voiture, elle est ",voit_1.couleur,". Sa
marque est ", voit_1.marque,"<br />");
var voit_2 = new Voiture ("Renault", "verte");
document.write("Je viens de créer une autre voiture, elle est ",voit_2.couleur,". Sa
marque est ", voit_2.marque,"<br />");
voit_1.couleur="jaune";
document.write("Je viens de repeindre ma ",voit_1.marque, ", elle est maintenant
",voit_1.couleur,".");
```

résultat 1.5



Il est maintenant possible de mieux comprendre l'utilité du mot clé this. Au moment de la création de l'objet voit_1, le mot clé this a systématiquement été remplacé par "voit_1" (même chose pour "voit_2"). Cela correspond bien à la "définition" que je vous ai donnée un peu au-dessus, "this correspond à l'objet courant" ("this=voit_1" quand on crée ou qu'on utilise voit_1 et "this=voit_2" quand on crée ou qu'on utilise l'objet voit_2).

Avant de parler des méthodes, nous devons nous intéresser aux prototypes :

En JavaScript, tous les objets sont liés à un autre objet appelé prototype. Chaque objet possède son propre prototype. Si vous créez un objet de type Voiture, le prototype de Voiture sera accessible tout simplement en écrivant « Voiture.prototype ».

A quoi sert le prototype ?

Question complexe, j'aurais l'occasion d'y répondre un peu plus loin, quand nous aborderons la notion d'héritage. Pour l'instant, vous devez uniquement connaître la procédure à suivre pour créer une méthode :

La méthode « n'appartiendra » pas directement à l'objet, mais «appartiendra » à son prototype (ne surtout pas perde de vu qu'un prototype est un objet comme un autre, il peut donc posséder des attributs et des méthodes), pour définir une méthode « accelere » utilisable par les objets de type voiture, il faudra écrire :

Voiture.prototype.accelere=function() {….}

Voici un exemple

exemple 1.6

```
function Voiture (marque, couleur) {
	this.marque = marque;
	this.couleur = couleur ;
}
Voiture.prototype.accelere = function () {
	document.write ("Votre voiture de marque ",this.marque," accélère");
}
var voit_1 = new Voiture ("Fiat", "rouge");
voit_1.accelere();
```

résultat 1.6

```
Votre voiture de marque Fiat accélère
```

Un autre exemple

exemple 1.7

```
function Voiture (marque, couleur) {
	this.marque = marque;
	this.couleur = couleur ;
}
Voiture.prototype.accelere = function () {
	document.write ("Votre voiture de marque ",this.marque," accélère</br>") ;
}
var voit_1 = new Voiture ("Fiat", "rouge");
var voit_2 = new Voiture ("Renault", "verte");
voit_1.accelere();
voit_2.accelere();
```

résultat 1.7

```
Votre voiture de marque Fiat accélère
Votre voiture de marque Renault accélère
```

Remarquez bien l'utilisation du this dans la méthode accelere (). Comme pour les attributs, le this va successivement être « remplacé » par voit_1 puis par voit_2.

Revenons sur le processus mis en œuvre ici :
Au moment de l'utilisation de la méthode accelere (par exemple «`voit_1.accelere();`»), le moteur JavaScript va chercher une méthode accelere dans l'objet Voiture (il ne faut pas perdre de vue que voit_1 est un objet de type Voiture). Ne trouvant pas de méthode accelere dans l'objet Voiture, il va étendre sa recherche au prototype de l'objet Voiture. Il va, dans cet exemple, trouver la méthode recherchée dans ce prototype. S'il ne l'avait pas trouvé, il aurait poursuivi sa recherche dans le prototype du prototype de l'objet voiture..... Au bout d'un moment, il serait arrivé au prototype de l'objet Object (objet de base du JavaScript) et si une fois du plus sa recherche s'était révélée infructueuse, il aurait alors retourné une erreur (du type « méthode non définie »). Tout ce processus utilise ce que l'on appelle « la chaine des prototypes».

Je précise qu'il est possible de définir les méthodes directement dans le constructeur de l'objet Voiture, mais ce procédé est déconseillé (utilisation de beaucoup de mémoire) et je n'en parlerai pas ici.

Notion d'héritage en JavaScript :

Voici une partie difficile mais non indispensable dans un premier temps. Si vous avez un peu de mal, n'hésitez pas à passer « votre chemin ».

L'héritage est une notion fondamentale en POO. Pour une fois l'héritage porte bien son nom, imaginons un objet A qui possède un attribut attrA et une méthode methA. Soit maintenant un objet B qui lui possède un attribut attrB et une méthode methA. Comme vous allez le voir, l'idée d'héritage est relativement simple : Si B hérite de A, alors l'utilisateur d'un objet de type B aura à sa disposition atrB et methB (ça c'est normal !) mais il pourra aussi utiliser atrA et methA.
Si vous avez compris cela, vous avez compris le principe de l'héritage en POO.

Dans les langages orientés objet « classiques » (C++ ou Java), l'héritage est relativement simple à mettre en œuvre. En JavaScript, les choses sont plus complexes :

Il existe plusieurs façons de faire de l'héritage en JavaScript, mais nous allons ici en voir qu'une. Ce n'est pas la simplicité qui a guidé mon choix, j'ai choisi de vous exposer ici la méthode respectant (selon moi!) le plus « l'esprit JavaScript ».

Pour commencer, nous allons étudier un cas avec des constructeurs vides et des objets possédant uniquement des méthodes (pas d'attributs). Pour que l'objet de type B puisse utiliser la méthode issue de l'objet A, nous allons devoir utiliser la méthode create de l'objet Object (Object.create()).

Que fait cette méthode ?

Cette méthode permet de créer un objet, le prototype de cet objet correspondant à l'objet passé en paramètre de la méthode create. Rien compris ? Voici un exemple :

var monObj1=Object.create(monObj2)

Nous créons un objet monObj1, cet objet aura pour prototype monObj2.

Pour faire de l'héritage, nous allons faire comme suit :

var B.prototype=Object.create(A.prototype)

Nous créons (plutôt nous recréons) le prototype de l'objet B. Le prototype de l'objet B aura pour prototype le prototype de l'objet A (oui, je sais, c'est compliqué).

Voici un exemple :

exemple 1.8

```
//constructeur de A (vide mais nécessaire)
function A() {
}
//nous créons la méthode maMethodeA
A.prototype.maMethodeA = function() {
    document.write("je suis une méthode de l'objet A")
};
function B() {
}
//l'héritage est là !
B.prototype = Object.create(A.prototype);
var monObjetB = new B();
//nous utilisons la méthode maMethode avec un objet de type B
monObjetB.maMethodeA();
```

résultat 1.8

```
je suis une méthode de l'objet A
```

Parcourons la chaîne des prototypes pour l'objet de type B :

- l'objet de type B (monObjetB) ne possède pas de méthode maMethodeA, passons à la suite
- le prototype de l'objet de type B ne possède pas de méthode maMethodeA, suivant
- le prototype du prototype de l'objet de type B (c'est-à-dire le prototype de l'objet de type A) possède la méthode recherchée, fin de la recherche

Il est évidemment possible de définir des méthodes propres à l'objet B. Attention cependant a bien définir ces méthodes après la ligne permettant l'héritage («`B.prototype=Object.create(A.prototype);`»). Sinon votre méthode sera « écrasée ».

exemple 1.9

```
//constructeur de A (vide mais nécessaire)
function A() {
}
//nous créons la méthode maMethodeA
A.prototype.maMethodeA = function() {
    document.write("je suis une méthode de l'objet A")
};
function B() {
}
//l'héritage est là !
B.prototype = Object.create(A.prototype);
//création de la méthode spécifique à B
B.prototype.maMethodeB=function() {
    document.write("je suis une méthode de l'objet B")
};
var monObjetB = new B();
//nous utilisons la méthode maMethode avec un objet de type B
monObjetB.maMethodeA();
monObjetB.maMethodeB();
```

résultat 1.9

```
je suis une méthode de l'objet A
je suis une méthode de l'objet B
```

Parfois le constructeur de l'objet B doit appeler le constructeur de l'objet A (par exemple pour que l'objet B puisse profiter des attributs définis dans le constructeur de l'objet A). Pour effectuer cet appel, il est indispensable d'utiliser la méthode call.

Exemple 1.10

```
//constructeur de A
function A() {
    document.write("le constructeur de A a bien été appelé");
    this.monAttrA="je suis un attribut de A";
}
//nous créons la méthode maMethodeA
A.prototype.maMethodeA = function() {
    document.write("je suis une méthode de l'objet A")
};
function B() {
    //cette ligne permet d'appeler le constructeur de A
    A.call(this);
}
//l'héritage est là !
```

```
B.prototype = Object.create(A.prototype);
//création de la méthode spécifique à B
B.prototype.maMethodeB=function() {
    document.write("je suis une méthode de l'objet B")
};
var monObjetB = new B();
//nous utilisons la méthode maMethode avec un objet de type B
monObjetB.maMethodeA();
monObjetB.maMethodeB();
document.write(monObjetB.monAttrA);
```

résultat 1.10

```
le constructeur de A a bien été appelé
je suis une méthode de l'objet A
je suis une méthode de l'objet B
je suis un attribut de A
```

Le this dans «`A.call(this);`» est très important, il permet d'utiliser l'attribut de l'objet A avec un objet de type B (le this en paramètre de la méthode call permet de passer une référence à l'instance en cours).
Oui, je sais, cela devient très compliqué, mais rassurez-vous, vous n'avez pas vraiment besoin de comprendre le « pourquoi du comment », vous devez juste retenir que si vous avez besoin d'appeler le constructeur de l'objet parent (ici A) depuis le constructeur de l'objet enfant (B) il faudra utiliser call comme indiqué ci-dessus.

Et si mon constructeur parent prend des paramètres, comment faire ?

Il faut alors utiliser le deuxième paramètre de la méthode call

exemple 1.11

```
function A(color) {
    this.monAttrA=color;
}
A.prototype.maMethodeA = function() {
    document.write(this.monAttrA)
};
function B(couleur, nombre) {
    //cette ligne permet d'appeler le constructeur de A et de lui passer un paramètre
    A.call(this,couleur);
    this.nbr=nombre
}
B.prototype = Object.create(A.prototype);
var monObjetB = new B("vert",12);
document.write("couleur : ",monObjetB.monAttrA,", nombre : ",monObjetB.nbr);
```

Je vous laisse étudier cet exemple, normalement il est assez compréhensible.

Pour terminer ce chapitre, une petite mise en garde : la méthode create est issue d'ECMAScript5, elle fonctionnera donc uniquement avec des navigateurs récents. Il existe des solutions pour contourner le problème (polyfill)
voir ici :
https://developer.mozilla.org/enUS/docs/JavaScript/Reference/Global_Objects/Object/create

Utilisation du DOM

Qu'est-ce que le DOM ?

DOM signifie Document Object Model, oui, je sais, vous n'êtes pas plus avancés avec cela.

Le DOM est une API (Application Programming Interface), toujours pas plus avancés ?

Alors pour essayer de faire simple et sans trop entrer dans les détails, grâce au DOM vous allez pouvoir manipuler les balises de votre code HTML (et CSS) comme des objets JavaScript. Vous aurez donc à votre disposition des propriétés (attributs et méthodes) qui vous permettront de modifier vos balises comme bon vous semble (par exemple en réagissant à l'action d'un utilisateur).

Encore une fois je vais essayer ici de simplifier les choses au maximum, il faut savoir que c'est un sujet très complexe et que certains ouvrages de plus de 300 pages sont entièrement consacrés à la gestion du DOM (voir à la fin du document des liens "pour aller plus loin").

Créons une page HTML toute simple :

Exemple 2

code HTML

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<script src="exemple_2.js"></script>
</head>
<body>
	<p id="para_1">Ceci est un paragraphe avec un texte qui ne sert à rien</p>
</body>
</html>
```

code JavaScript

```
window.onload = function () {
	window.alert("Regarder votre page, elle va être modifié après avoir appuyé sur
OK");
	var monPara=window.document.getElementById("para_1");
	monPara.innerHTML = "Voilà le texte a bien été modifié";
}
```

NB : Regardez le code de votre page HTML à l'aide de firebug, vous pourrez constater que grâce à la manipulation du DOM, c'est bien la structure de la page qui a été modifiée (pas seulement le texte affiché à l'écran).

Que de nouveautés dans un si petit exemple !
Je ne reviens pas sur le code HTML, il ne devrait vous poser aucun problème.
En revanche, reprenons ligne par ligne le code JavaScript :

`window.onload = function () {`
Comme déjà vu précédemment, nous avons ici une fonction anonyme (une fonction sans nom). Tout notre code est contenu dans cette fonction anonyme. Pourquoi ?
Supprimez cette fonction anonyme et le `window.onload =` (en laissant bien sûr le reste du code !), que se passe-t-il ? Notre texte ne s'affiche plus avant l'apparition de la fenêtre pop-up (la fenêtre avec le texte `"Regarder votre page, elle va être modifié après avoir appuyé sur OK"`). Encore une fois pourquoi ?
Le code HTML est parcouru de "haut en bas", le fichier JavaScript est donc exécuté avant l'affichage de notre paragraphe (`<p id="para_1">ceci est un paragraphe avec un texte qui ne sert à rien</p>`). La fenêtre pop-up étant "bloquante", le reste du code de la page HTML ne sera

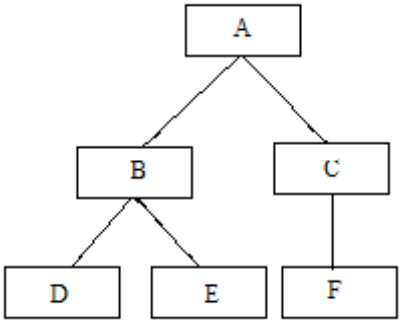
pas chargé tant que vous n'aurez pas appuyé sur OK (il en est de même pour le reste du code JavaScript).

`window.onload` est un gestionnaire d'événements (nous les verrons plus tard) qui permet d'attendre que la page HTML soit entièrement chargée avant de commencer à exécuter le code JavaScript. Plus exactement le contenu de la fonction anonyme ne sera pas exécuté tant que la page HTML ne sera pas entièrement chargée (nous avons ici associé l'événement "onload" à une fonction anonyme).

Passons à l'analyse de la deuxième ligne :

```
window.alert("Regarder votre page, elle va être modifié après avoir appuyé sur OK");
```

En JavaScript presque tout est objet, window est un objet. C'est l'objet global !
Comme déjà évoqué un peu plus haut, un objet B peut très bien être une des propriétés d'un objet A. Il existe un lien de "parenté" entre l'objet A et l'objet B : on dit que "B est l'enfant de A". Nous avons donc une arborescence que peut ressembler à ceci :



Quand vous travaillez sur une page web avec JavaScript, l'objet window correspond à l'objet A, c'est lui qui est "tout en haut" de l'arborescence. Tous les objets seront donc des descendants de window.
L'objet window possède un certains nombre de méthodes, parmi celles-ci vous allez trouver la méthode alert qui permet d'afficher une fenêtre "surgissante" (pop-up). Cette méthode alert prend pour paramètre une chaîne de caractères qui sera affichée dans la fenêtre. Attention, il faut éviter d'utiliser ces pop-up, à la longue, leurs apparitions continuelles peuvent être extrêmement désagréables pour l'utilisateur.
Dernière chose, l'objet window étant l'objet global, il peut être sous-entendu, et sa présence dans le code n'est pas obligatoire, vous pouvez donc écrire :

```
alert("Regarder votre page, elle va être modifié après avoir appuyé sur OK");
```

Cela fonctionnera tout aussi bien

3ème ligne : `var monPara=window.document.getElementById("para_1");`
La présence du mot var nous permet d'affirmer que monPara est une variable. Mais que va contenir cette variable ?
Nous pouvons déjà constater que nous avons ici un objet document (qui est une propriété de l'objet window). L'objet document représente l'ensemble des éléments de la page HTML (nous avons déjà eu affaire à lui quand nous avons écrit : document.write("mon texte"), write( ) est donc une méthode de document).
Cet objet document possède une méthode getElementById. Cette méthode attend comme argument la valeur d'un id de la page HTML (ici l'id de notre paragraphe a bien pour valeur para_1).
La méthode getElementById nous permet de créer un objet qui sera la version DOM (donc manipulable grâce au JavaScript) d'une balise HTML, en l'occurrence ici, la balise <p>.
Une fois cet objet créé, il est "rangé" dans la variable monPara. Nous aurons à de nombreuses reprises l'occasion d'utiliser cette méthode.

Dernière ligne de notre exemple : `monPara.innerHTML = "Voilà le texte a bien été modifié";`
Nous utilisons notre objet fraîchement créé : monPara. L'objet monPara (comme tous les objets créés grâce à la méthode getElementById) possède un attribut innerHTML (attention, j'ai bien dit attribut et pas méthode). La valeur de cet attribut correspond au contenu de la balise, qui est représentée par notre objet monPara. Pour bien que vous compreniez tout cela, nous allons très légèrement modifier l'exemple 2.1

exemple 2b

```
window.onload = function () {
	var monPara=window.document.getElementById("para_1");
	alert("avant : "+monPara.innerHTML);
	monPara.innerHTML = "Voilà le texte a bien été modifié";
	alert("Maintenant, la valeur du innerHTLM de l'objet monPara est :
"+monPara.innerHTML);
}
```

NB : remarquez la concaténation pour le paramètre de la méthode alert.

Nous verrons prochainement d'autres exemples d'utilisations d'innerHTML. Au risque de me répéter, il faut bien avoir conscience que j'ai essayé de simplifier les choses au maximum, l'utilisation du DOM pour rendre des pages dynamiques est très complexe, voici d'ailleurs un lien pour ceux qui voudraient en savoir un peu plus sur la question :
http://www.siteduzero.com/tutoriel-3-379719-manipuler-le-code-html-partie-1-2.html

Gestion des événements

Il existe plusieurs méthodes pour gérer les événements en JavaScript, mais nous allons n'en étudier qu'une (pour les autres méthodes, voir le lien donné plus bas).
Nous avons déjà vu un événement un peu plus haut : l'événement onload
Je vous donnerai une liste des événements, mais pour commencer notre étude, nous allons nous intéresser à l'événement onclick.

Exemple 3

fichier HTML

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<script src="exemple_3.js"></script>
</head>
<body>
	<div id="maBalise">
		<h1>Cliquez dans ce rectangle</h1>
	</div>
</body>
</html>
```

fichier CSS

```
#maBalise
{
	background-color : red;
	width : 250px;
	height : 100px;
}
```

fichier JavaScript

```
window.onload = function () {
	var balisediv=window.document.getElementById("maBalise");
	balisediv.onclick=function () {
		balisediv.innerHTML="<h1>Bravo vous avez cliqué</h1>";
	}
}
```

Je pense que les fichiers HTML et CSS ne devraient vous poser aucun problème.
Le fichier JavaScript est aussi relativement facile à comprendre. Seule nouveauté :

```
balisediv.onclick=function () {
		balisediv.innerHTML="<h1>Bravo vous avez cliqué</h1>";
	}
```

En cas de clic sur le rectangle rouge (la balise div représentée par l'objet JavaScript balisdiv), la fonction anonyme est exécutée : modification de la balise div à l'aide d'innerHTML.
Il faut savoir que l'utilisation d'une fonction anonyme n'est pas une obligation :

```
window.onload = function () {
	var balisediv=window.document.getElementById("maBalise");
	function monClic () {
		balisediv.innerHTML="<h1>Bravo vous avez cliqué</h1>";
	}
	balisediv.onclick= monClic;
}
```

Attention il ne faut surtout pas mettre de parenthèses après monClic (`balisediv.onclick= monClic;`). Dans le cas contraire, votre fonction sera immédiatement exécutée (même sans clic)

Voici une liste (non exhaustive) d'événements :

onmouseover	Faire entrer le curseur sur l'élément
onmouseout	Faire sortir le curseur de l'élément
onmousedown	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
onmouseup	Relâcher le bouton gauche de la souris sur l'élément
onmousemove	Faire déplacer le curseur sur l'élément
onkeydown	Appuyer (sans relâcher) sur une touche clavier sur l'élément
onkeyup	Relâcher une touche clavier sur l'élément
onkeypress	Frapper (appuyer puis relâcher) sur une touche clavier sur l'élément

Nous allons maintenant apprendre à gérer l'événement onkeydown.
Pour commencer, vous devez savoir qu' à chaque touche du clavier correspond un code (nombre).
Voici un lien qui vous donne le code des touches les plus courantes

http://www.commentcamarche.net/faq/18760-javascript-manipulation-des-controles-clavier#propriete-keycode-et-liste-des-valeurs

Il est possible d'envoyer un objet event à la fonction anonyme qui est exécutée en cas d'événement.
Cet objet event contient beaucoup d'informations. Il possède, entre autres, un attribut keyCode qui a pour valeur le code de la touche pressée.
Voici un exemple pour illustrer son utilisation :

Exemple 4

fichier HTML

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<script src="exemple_4.js"></script>
</head>
<body>
	<h1>Appuyer sur une flèche</h1>
	<div id="maBalise">
		Aucune touche
	</div>
</body>
</html>
```

fichier CSS

```
#maBalise
{
	background-color : red;
	width : 250px;
	height : 100px;
}
```

fichier JavaScript

```
window.onload = function () {
	var balisediv=window.document.getElementById("maBalise");
	window.onkeydown = function(event) {
		if (event.keyCode == 39) {
			balisediv.innerHTML = "flèche droite";
		}
		if (event.keyCode == 37) {
			balisediv.innerHTML = "flèche gauche";
		}
		if (event.keyCode == 38) {
			balisediv.innerHTML = "flèche haut";
		}
		if (event.keyCode == 40) {
			balisediv.innerHTML = "flèche bas";
		}
	}
}
```

Notre fonction anonyme reçoit l'objet event quand une touche est pressée. L'attribut keyCode de notre objet event a pour valeur le code de la touche. Nous testons la valeur de keyCode, si elle est égale à 39, la balise div de mon code HTML est alors modifiée (toujours par l'intermédiaire de l'objet balisediv et de l'attribut innerHTML). Le reste du code fonctionne exactement de la même façon.

Une fois de plus il y aurait beaucoup d'autre choses à dire sur la gestion des événements, je vous propose donc la lecture suivante pour pouvoir creuser la question : http://www.siteduzero.com/tutoriel-3-379722-les-evenements.html

jQuery
Write Less, Do More

"Écrire moins, en faire plus" est donc la devise de ce framework JavaScript.
Tout d'abord qu'est-ce qu'un framework ?
Un framework est une bibliothèque d'objets (objet au sens informatique du terme bien évidemment), censée faciliter la vie du développeur.
Jquery va donc nous apporter une collection de méthodes et d'attributs qui va nous rendre le travail plus facile.
Jquery va vous permettre de :

- modifier très facilement l'aspect de votre page (jouer sur les CSS)
- modifier le contenu de votre page (jouer sur le HTML)
- gérer les événements
- créer des animations sur votre page
- communiquer facilement avec un serveur (requête Ajax très simplifiée)

Oui, c'est vrai, modifier l'aspect de la page ou modifier le contenu de la page vous savez déjà faire (avec le DOM), mais jQuery va vous faciliter la vie en simplifiant énormément la procédure.
De toutes les façons, tout ce qui est faisable avec jQuery est aussi faisable sans jQuery, mais c'est plus compliqué.
Il existe des ouvrages de 500 pages exclusivement consacrées à jQuery. Vous comprendrez donc que je ne pourrais pas être exhaustif en quelques pages. La documentation est excellente (en anglais), je vous invite donc à la consulter régulièrement (notamment la partie "JQUERY API REFERENCE" qui recence toutes les propriétés de jQuery) :
http://docs.jquery.com/Main_Page
Pour utiliser jQuery, il existe plusieurs méthodes "d'installation". Je vais me contenter de vous expliquer la plus simple.
Pourquoi la plus simple ? Justement parce qu'il n'y a rien à installer, vous avez juste à ajouter une ligne dans votre fichier HTML :

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
```

Cette ligne devra être placée juste avant l'appel à votre propre fichier JavaScript :

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<link href="style_ex3.css" rel="stylesheet"/>
	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
	<script src="monfichier.js"></script>
</head>
```

Nous allons commencer par utiliser jQuery pour modifier le style d'une page.

La première chose à faire est de créer un objet jQuery à l'aide de la méthode jQuery(). Cette méthode a un paramètre : un sélecteur CSS.
Qu'est-ce qu'un sélecteur CSS ?

Considérons le fichier CSS suivant :

```
p
{
	margin : 10px ;
}

#maBalise
{
	background-color : green ;
}

.autresBalises
{
	font-size : 15px ;
}
```

p, #maBalise, et .autreBalise sont des sélecteurs CSS. Un sélecteur CSS vous permet d'identifier des balises HTML (p : tous les paragraphes de la page, #maBalise : la balise ayant pour id maBalise, .AutresBalises : les balises ayant pour class AutresBalises). Si nécessaire, n'hésitez pas à relire la partie CSS du document "HTML et CSS, les bases".

Commençons à travailler sur un nouvel exemple :

exemple 5

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<link href="style.css" rel="stylesheet"/>
	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
	<script src="monfichier.js"></script>
</head>
<body>
    <h1>Exemple jquery</h1>
    <div id="maBalise">
		<p>Appuyer sur R ou appuyer sur V</p>
    </div>
</body>
</html>
```

Rien de bien compliqué dans ce fichier HTML, nous avons créé une balise div ayant pour id maBalise.

```
#maBalise
{
	border : 2px solid black;
	padding : 10px;
	width : 250px;
	height : 100px;
}
```

RAS pour ce fichier CSS

Nous allons maintenant nous intéresser au fichier JavaScript :

```
window.onload = function () {
	maBalisejQ=jQuery("#maBalise");
}
```

La méthode onload ne doit plus avoir de secret pour vous. La deuxième ligne du script nous permet de créer un objet jQuery grâce à la méthode jQuery( ). Cette méthode a pour paramètre le sélecteur

#maBalise.
Cet objet va nous permettre de manipuler comme bon nous semble tout ce qui concerne la balise div ayant pour id maBalise (partie HTML et partie CSS)
Nous nous intéressons pour l'instant à la manipulation du style, nous allons donc modifier, dans le code JavaScript le style appliqué à la balise div :

```
window.onload = function () {
maBalisejQ=jQuery("#maBalise") ;
maBalisejQ.css("background-color","red") ;
}
```

La méthode css appliquée à l'objet jQuery que nous venons de créer va nous permettre de modifier les propriétés de style de notre balise div. La dernière ligne de code ajoute la propriété background-color (alors que cette dernière, à l'origine, n'était pas présente dans notre fichier css). La valeur de cette propriété sera red (faites attention à ne pas oublier les guillemets).
Grâce à la méthode css, il est possible de manipuler toutes les propriétés css.

La divise de jQuery commence par "Écrire moins", pour l'instant nous avons écrit 2 lignes de code, vous allez voir qu'il est possible de réduire nettement cela :
Tout d'abord, l'objet maBalisejQ n'a pas vraiment d'intérêt, c'est juste un intermédiaire, il est tout a fait possible de créer l'objet jQuery et de lui appliquer la méthode css dans la foulée :

```
jQuery("#maBalise").css("background-color","red") ;
```

Voilà, plus de maBalisejQ !

Les concepteurs de jQuery trouvaient que cela était encore trop long : il est en effet possible de remplacer la méthode jQuery par le caractère $. Ce qui nous donne :

```
$("#maBalise").css("background-color","red") ;
```

Vous pouvez constater que ces changements n'ont en rien modifiés le résultat.
Vous allez me dire : aucun intérêt, nous aurions pu directement ajouter le background-color au fichier css !
Oui, sauf qu'ici, nous travaillons sur du JavaScript, nous pouvons donc programmer la modification du style. Voici un exemple pour vous en convaincre (les fichiers HTML et CSS restent identiques) :

```
window.onload = function () {
	window.onkeydown = function(event) {
		if (event.keyCode == 82) {
			$("#maBalise").css("background-color","red");
		}
		if (event.keyCode == 86) {
			$("#maBalise").css("background-color","green");
		}
	}
}
```

Je vous laisse tester et réfléchir à cet exemple (le code 82 correspond à la touche r, le code 86 correspond à la touche v). Cela ne devrait vous poser aucune difficulté.

Un grand nombre de sélecteurs est disponible, n'hésitez pas à consulter la documentation officielle (dans la partie Selectors).

Il existe aussi d'autres méthodes jQuery qui permettent la manipulation du style CSS, voici une liste de méthodes que je vous demande de bien vouloir étudier à l'aide de la documentation officielle (partie CSS) :

- addClass( )
- removeClass ( )
- toggleClass ( )

Manipulation du DOM

Nous allons maintenant modifier le contenu d'une page.

La méthode html ( ) vous permet de modifier votre code HTML, le principe est très proche d'innerHTML.

Exemple 6

Fichier HTML

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<link href="style2.css" rel="stylesheet"/>
	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
	<script src="monfichier2.js"></script>
</head>
<body>
	<h1>Exemple jquery</h1>
    <div id="maBalise">
		<p>Appuyer sur R</p>
	</div>
</body>
</html>
```

fichier CSS : identique à l'exemple précédent

fichier javaScript

```
window.onload = function () {
	window.onkeydown = function(event) {
		if (event.keyCode == 82) {
			$("#maBalise").html("<h1>Bravo, très bien joué</h1>");
		}
	}
}
```

Comme vous pouvez le constater, le contenu de la balise div (`<p>Appuyer sur R</p>`) est **remplacé** par l'argument de la méthode html (`"<h1>Bravo, très bien joué</h1>"`).

Il existe une méthode un peu équivalente, la méthode texte. Différence entre les deux ?
La méthode texte n'interprétera pas les balises HTML comme des balises HTML, mais tout simplement comme du texte. Pour vous en convaincre, dans l'exemple précédent, remplacez la méthode html ( ) par la méthode text ( ).

Autres méthodes très utiles, append ( ) et prepend ().
Voici un exemple avec append (les fichiers html et css restent identiques) :

exemple 7

```
window.onload = function () {
	window.onkeydown = function(event) {
		if (event.keyCode == 82) {
			$("#maBalise").append("<p>Après</p>");
		}
	}
}
```

append vous permet d'insérer du contenu à la fin de la balise div ayant l'id "maBalise". Nous avons bien ici une insertion, pas un remplacement comme pour la méthode html ( ).
Je pense que vous aurez compris que la méthode prepend a le même rôle que la méthode append, sauf que le contenu est ajouté au début (faites vous-même le test).

La méthode appendTo ( ) ressemble beaucoup à la méthode append ( ). La cible et l'objectif sont inversés par rapport à la méthode append ( ). Voici un exemple un peu plus complexe :

fichier HTML

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<link href="style4.css" rel="stylesheet"/>
	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
	<script src="monfichier4.js"></script>
</head>
<body>
	<h1>Exemple jquery</h1>
    <div id="maBalise">
		<p>Appuyer sur R</p>
	</div>
</body>
</html>
```

fichier CSS

```
#maBalise
{
	border : 2px solid black;
	padding : 10px;
	width : 250px;
	height : 100px;
}

.maClasse
{
	border : 2px solid red;
	padding : 15px;
	width : 300px;
	height : 150px;
	background-color: green;
}
```

fichier javaScript

```
window.onload = function () {
	window.onkeydown = function(event) {
		if (event.keyCode == 82) {
		$("<div></div>").html("<h1>Hello World</h1>").addClass("maClasse").appendTo("body");
		}
	}
}
```

Exemple un peu plus complexe donc, mais qui illustre bien la puissance de jQuery :

En cas d'appui sur R :

- vous créez un objet jQuery qui va correspondre dans le DOM a une balise div ouvrante et fermante (`$("<div></div>")`)
- vous ajoutez à cette balise div un titre h1 (`html("<h1>Hello World</h1>")`)
- vous attribuez à cette balise div une classe de style (`addClass("maClasse")`)
- Vous insérez cette balise div à la balise body (`appendTo("body")`)

Essayez de faire la même chose en JavaScript "classique", c'est faisable, mais, beaucoup plus long !

Voici quelques autres méthodes qui pourraient vous être utiles (voir la doc officielle) :

- after ( ) ou insertAfter ( )
- before ( ) ou insertBefore ( )
- attr ( )

Les événements

Avec Jquery la gestion des événements est grandement simplifiée (il n'y a plus de problème de compatibilité de certains navigateurs).
Pour commencer nous allons remplacer notre "window.onload" par un événement jQuery : $(document).ready ( ).

Pourquoi ?

Notamment parce que window.onload attend que toute la page soit chargée (images comprises, ce qui peut être très long dans certains cas) alors que $(document).ready ( ) attend que le DOM soit chargé (ce qui est beaucoup moins long). Nous aurons l'occasion de l'utiliser dans les exemples suivants.
Premier événement, le clic de souris :

exemple 8

Le fichier CSS reste identique

fichier HTML

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<link href="style5.css" rel="stylesheet"/>
	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
	<script src="monfichier5.js"></script>
</head>
<body>
	<h1>Exemple jquery</h1>
    <div id="maBalise">
		<p>Cliquez dans cette fenêtre</p>
	</div>
</body>
</html>
```

fichier javaScript

```
$(document).ready(function () {
```

```
	$("#maBalise").click(function(event) {
		$("<div></div>").html("<h1>Hello World</h1>").addClass("maClasse").appendTo("body");}
	);
});
```

Nous avons ici affaire à des fonctions anonymes, faites très attention aux points virgules. Le reste ne devrait pas vous poser de problème.
Le principe est simple : objet_concerné.événement (fonction_anonyme ( ) {.....}) ;
Dans notre cas "objet_concerné" correspond à `$("#maBalise")` et "événement" correspond à click.

Il est aussi possible d'utiliser un objet de type event :

fichier html

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<link href="style6.css" rel="stylesheet"/>
	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
	<script src="monfichier6.js"></script>
</head>
<body>
	<h1>Exemple jquery</h1>
    <div id="maBalise1">
		<p>Cliquez dans cette fenêtre</p>
	</div>
	<div id="maBalise2">
	</div>

</body>
</html>
```

fichier css

```
#maBalise1
{
	border : 2px solid black;
	padding : 10px;
	width : 250px;
	height : 100px;
}
```

fichier javaScript

```
$(document).ready(function () {
	$("#maBalise1").click(function(event) {
		$("#maBalise2").html("<p>Coordonnée X : " + event.pageX + "</p><p>Coordonnée Y : " +
		event.pageY + "</p>");}
	);
});
```

Nous utilisons ici event.pageX et event.pageY qui nous permettent de connaître les coordonnées de la souris (dans la fenêtre "maBalise1") au moment du clic (origine : en haut et à gauche)

La consultation de la doc officielle (partie "Events") vous permettra d'avoir une vue plus exhaustive des événements.

Il nous reste à voir les effets et les animations jQuery.
Avant d'aborder cette dernière partie, je me dois de vous rappeler que nous avons vu que CSS3 nous permet de réaliser des animations et des effets sophistiqués (voir HTML et CSS, pour aller plus loin). CSS3 étant moins gourmand en ressource que javaScript, il est donc préférable de favoriser la solution CSS3 par rapport à la solution jQuery. Pour être tout à fait honnête, je vous

rappelle aussi que les navigateurs quelque peu anciens ne supportent pas le CSS3.
Comme d'habitude, tout est une histoire de compromis.

Voici quelques méthodes qui pourraient vous êtes utiles :

La méthode hide ( ) vous permet de cacher un élément, la méthode show ( ) vous permet de le faire apparaître. Ces 2 méthodes acceptent un paramètre : la durée de la disparition (ou de l'apparition) en millisecondes.

Exemple 9

Fichier HTML

```
<!doctype html>
<html lang="fr">
<head>
	<meta charset="utf-8">
	<title>Programmation JavaScript</title>
	<link href="style7.css" rel="stylesheet"/>
	<script src="http://ajax.googleapis.com/ajax/libs/jquery/1.9.1/jquery.min.js"></script>
	<script src="monfichier7.js"></script>
</head>
</head>
<body>
	<h1>Exemple jquery</h1>
	<h2 id="app">Cliquez ici SVP pour faire apparaître</h2>
	<h2 id="disp">Cliquez ici SVP pour faire disparaître</h2>
    <div id="maBalise1">
		<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. In porta
venenatis odio at mollis. Cras ligula sem, convallis non mollis et, tincidunt at erat.
Donec vitae volutpat erat.Pellentesque a risus justo. Cras dictum varius purus, eget
mollis felis varius a. Quisque feugiat ligula mattis nunc ultrices pretium. Sed at </p>
	</div>
	<div id="maBalise2">
	</div>

</body>
</html>
```

fichier CSS

```
#maBalise1
{
	border : 2px solid black;
	padding : 10px;
	width : 750px;
	height : 250px;
}
```

fichier javaScript

```
$(document).ready(function () {
	$("#disp").click(function(event) {
			$("#maBalise1").hide("500");}
	);
	$("#app").click(function(event) {
			$("#maBalise1").show("400");}
	);
});
```

La méthode toggle ( ) a un certain intérêt, puisqu’elle cache un élément s'il est visible et le rend visible s'il est caché.

Autres méthodes à tester, les méthodes fadeIn ( ) et fadeOut ( ), elles permettent aussi de faire

apparaître ou disparaître du contenu, mais de façon,.... "plus agréable" que hide et show (ces méthodes acceptent aussi une durée comme argument). A noter qu'il existe aussi une méthode fadeToggle ().

slideDown ( ) et slideUp ( ) vous permettent de "dérouler" (ou "d'enrouler") un élément verticalement (à tester aussi en remplacement des méthodes hide et show du dernier exemple).

Enfin pour terminer, il faut que vous sachiez qu'il existe une méthode animate ( ) qui ressemble beaucoup à "animation" du CSS3. Consulter la doc officielle de jQuery pour plus d'informations sur le sujet (partie Effects).

Il reste beaucoup de choses à dire sur jQuery mais je pense qu'avec les éléments donnés dans ce document vous pouvez d'ores et déjà vous "amuser" un peu. Pour aller encore plus loin, une fois de plus, consultez la documentation officielle (vous y trouverez aussi des tutoriels, dont certains en français !).