JavaScript

Méthodes, techniques et outils de développement

Olivier Dommange
2016

Sommaire

Javascript est un langage qui s'associe au HTML pour dynamiser les pages Web. La balise <script> lui est exclusivement dédiée depuis la version du HTML 5. Plus besoin de définir le MIME type lors de la liaison d'un fichier Javascript externe comme cela devait se faire auparavant `<script type="text/javascript">`. Sa place est dorénavant au même titre que le CSS, un outil au service du contenu, facilitant sa consultation et dynamisant les interfaces.

La position qu'occupe Javascript s'est davantage renforcée avec les librairies de codes (**framework**) qui simplifient son utilisation. Les navigateurs comme ceux des supports mobiles interprètent de mieux en mieux les fonctions du langage. Il apparaît difficile d'imaginer ne plus utiliser d'animations pour augmenter la navigabilité ou de disposer d'opérations asynchrones pour dynamiquement mettre à jour des contenus d'une page.

L'utilisation de Javascript va dorénavant au-delà de l'interprétation qu'en font les navigateurs Web. Il peut-être utilisé côté serveur avec des solutions telles que **node.js** qui s'avère être une alternative performante aux langages serveur existants. De plus, Javascript s'intègre dans plusieurs logiciels et systèmes d'exploitation. De même que de plus en plus de sites Web prennent la forme d'application (édition d'images, traitement de textes, tableur...) grâce aux possibilités qu'offre le langage.

Javascript est un langage dont les spécifications ont été standardisées par l'association ECMA International (European Computer Manufacturers Association Script). **ECMAScript** est le produit de cette standardisation (ECMA-262 specification et ISO/IEC 16262). Il n'est pas surprenant de rencontrer des similitudes entre Javascript et Actionscript qui ont tous deux comme référence commune ECMAScript. Une précision encore, Javascript n'est pas Java. Aucune similitude n'existe entre ces langages, autre que le nom qu'ils portent et une influence somme toute relative.

Ce document propose des méthodes et techniques de développement avec ce langage. Il a pour but de présenter des méthodes d'utilisation du langage dans le cadre de développements. Bien qu'un bref retour sur les principes de bases soit proposé, il est encouragé, voire très utile, de se familiariser avec les notions de base de la programmation : les conditions, boucles et fonctions.

Développer avec Javascript

Javascript interagit avec HTML et CSS, son champ d'action est le document, la fenêtre du navigateur en plus des **composants du code HTML**. Javascript est à même de composer, modifier ou supprimer du code HTML ou CSS. Il peut interagir avec un serveur à distance ou transmettre dynamiquement des informations d'un site à un autre. Son potentiel permet d'envisager des possibilités intéressantes certes mais implique parfois un développement tout de même soutenu.

A cela s'ajoute une problématique d'interprétation du code par les différents **navigateurs** et des versions qui impliquent des aménagements qui relèvent parfois du grand écart. Bien que cette problématique soit de moins en moins importante, elle demeure bien réelle.

Enfin s'ajoute les supports mobiles et la déclinaison d'événements tels que le « *touch* » qui impliquent une considération supplémentaire à prévoir. Sans parler de l'art d'adapter les interfaces aux multiples tailles d'écrans (Responsive WebDesign) qui implique d'ajuster animations, mises en page et outils de navigation pour toutes tailles d'écran. Plusieurs de ces ajustements se font avec le CSS mais Javascript n'est pas en reste.

L'utilisation de librairies de codes (framework) telles que **jQuery** va s'avérer être une solution privilégiée pour ces « problématiques ». Elles permettent de gagner du temps tout en harmonisant le fonctionnement du script pour les différents navigateurs et leurs multiples versions. jQuery offre également une solution mobile avec **jQuery mobile**. Il est dorénavant difficile d'envisager un développement sans l'utilisation de librairies. Afin de profiter pleinement du potentiel des librairies, il s'avère cependant utile de bien connaître l'utilisation de Javascript, son potentiel, ses règles et les outils qui le composent.

Introduire du code Javascript dans une page HTML

Toutes les méthodes ne sont pas équivalentes quant à la pratique à adopter. Javascript peut s'introduire de 3 manières dans le code HTML. Il est possible de recréer les mêmes animations et interactions avec les 3 méthodes présentées. Cependant, la méthode à privilégier est celle qui facilitera la maintenance du code et qui est dissociée du HTML.

Modes	
Intégré dans une balise HTML	Le code Javascript est directement posé dans une balise HTML par l'entremise d'attributs qui font référence à un événement. Cette méthode alourdit fortement le code et rend la maintenance du code complexe. Elle n'est pas recommandée. `1 <img src="image.jpg" onclick="this.src='image-active.jpg';" />`
Inséré dans une balise `<script>`	Insertion du code Javascript dans la balise HTML <script> réservée à l'exécution du Javascript ou à la liaison d'un fichier externe. La balise <script> peut être inscrite à n'importe quel endroit dans le code HTML. Il est recommandé de l'inscrire dans l'entête ou le pied de page. Cette méthode peut être tolérée dans le cas où un code de faible taille (moins d'une dizaine de lignes) provient d'un outil tiers. Google Analytics notamment propose un code à introduire dans les pages d'un site.

```
<script>
      // Javascript
</script>
```

Fichier externe

Insertion du code Javascript dans un fichier externe au format (.js).

Cette méthode est celle à adopter. Des fichiers peuvent être créés pour chaque animation d'un projet Web.

```
<script src="js/myScript.js"></script>
```

Comment Javascipt agit-il sur le HTML ?

Lorsque Javascript modifie l'apparence d'un élément HTML, il modifie les valeurs des attributs de cette balise. Il en va de même pour les modifications apportées par Javascript sur les styles (CSS). Dans ce cas précis, Javascript utilise l'attribut style d'une balise pour indiquer les propriétés modifiées.

On constate que ce qui est convenu comme une mauvaise pratique lorsqu'on crée du code CSS s'avère tout de même être le mode d'application qu'emploie Javascript. En fait, Javascript génère et intervient dynamiquement sur le code HTML. Ceci se constate lorsqu'on observe de plus près le code créé avec Javascript : a un objet est associé le nom d'un attribut à qui la valeur est modifiée.

Les outils de développement intégrés dans les navigateurs permettent d'observer les modifications qui se font dynamiquement. Firebug est une extension équivalente qui peut être installée dans le navigateur Firefox.

Exemple 1 :

```
document.getElementById('img').src='image.jpg';
```

Résultat : `<img id="img" src="image.jpg" />`

Exemple 2 :

```
document.getElementById('img').style.opacity='0.5';
```

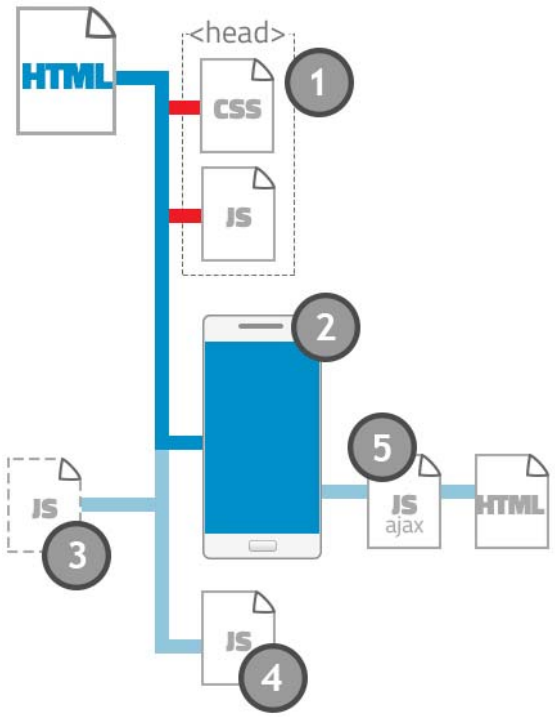
Résultat : `<img id="img" style="opacity:0.5" />`

Les méthodes d'assignation du Javascript

Comme cela vient d'être présenté, Javascript modifie les éléments du code HTML en donnant de nouvelles valeurs aux attributs des balises. Ce qui signifie que le code HTML et CSS doivent être chargés dans le navigateur avant que le Javascript ne s'exécute. Autrement, le code Javascript tentera de s'adresser à une balise HTML inexistante puisqu'elle n'est pas encore chargée dans la page. Conséquence, aucune modification n'est apportée à la page.

Une seconde considération importante. Chaque élément (CSS, Javascript, image, vidéo,...) qui se charge dans la page crée un effet de blocage et prolonge le moment où la page s'affichera dans le navigateur. Il est donc important que la priorité soit surtout laissée au HTML. Il est possible d'améliorer le chargement de la page grâce à Javascript en indiquant le moment où certains médias peuvent être chargés dans la page mais il est également déterminant de préciser le moment où le Javascript lui-même se charge.

Voici la séquence de chargement et d'affichage de la page et les méthodes utiles pour optimiser le temps de chargement.



1. Les fichiers liés se chargent avec le code HTML et le contenu de la page créant un ralentissement à l'affichage du contenu.
2. Affichage des contenus de la page Web sur le support.
3. L'attribut defer synchronise le chargement du code Javascript après que le contenu HTML est affiché.
4. Code Javascript présent en fin de la page HTML. Le chargement implique un délai supplémentaire.
5. La méthode AJAX permet de charger du contenu suite à un événement ou en fonction d'indications établies en Javascript.

Lier les fichiers Javascript à la fin du document HTML

Considérant que le code de la page HTML est pris en charge au fur et à mesure que la page est interprétée par le navigateur, lier les fichiers Javascript en fin de page assure que le code sera interprété en fin de chargement de la page. Bien que ce soit une façon de définir correctement la séquence de chargement cette méthode comporte un aspect peu enviable.

Premièrement, le code en bas de page n'est pas très élégant. Il devrait être posé en haut de la page dans la balise `<head>`. La balise `<script>` permet d'introduire du code Javascript à n'importe quel endroit dans le code HTML, comme la balise <style> pour le CSS. Dans ces deux cas, il n'est pas très « propre » d'utiliser ces balises dans le HTML, il est d'ailleurs recommandé d'externaliser les fichiers de ces deux langages pour le rendre accessible pour l'ensemble des pages d'un site. Dans la section des bonnes pratiques qui suit, sont présentés les cas où les fichiers liés en bas de page peuvent s'avérer utiles et la méthode `defer` qui permet d'indiquer à un code Javascript de ne se charger dans le navigateur que lorsque le HTML et CSS sont présents.

Exécuter Javascript dans un deuxième temps

Le chargement du fichier Javascript dans la mémoire du navigateur est une chose, son exécution implique l'exploitation de ressources supplémentaires. Il est possible que le code s'assure que la page et ses ressources (images et vidéos) soient chargées avant d'exécuter quelque Javascript que ce soit. Ceci se fait en imbriquant le code dans l'événement `onload` de l'objet `window`.

```
window.onload = function(){
      // le code à exécuter une fois la page et ses ressources chargées…
}
```

L'attente que tous les éléments de la page soient chargés peut s'avérer plutôt longue considérant que les ressources peuvent disposer d'un certain poids. Il peut par conséquent suffire de n'attendre que la mise en mémoire du document.

```
document.addEventListener( "DOMContentLoaded", function(){
      // le code à exécuter une fois le document chargé…
}
```

Dans le chapitre traitant des étapes du processus d'assignation d'événements sera présenté l'utilisation de ces ressources dans le développement.

AJAX

L'une des solutions également envisageable est de faire appel à AJAX. Cette méthode permet de charger un fichier ou du contenu à n'importe quel moment ou suite à l'exécution d'un événement dans la page, tel un clic ou le chargement complet de la page.

La section à la page 43 de ce document présente AJAX et son mode de chargement du contenu.

Les bonnes pratiques d'utilisation de Javascript

Les pratiques de développement avec Javascript se définissent en fonction de deux facteurs particuliers, le chargement du script et la maintenance du code. Il s'agit d'enjeux liés à la performance et la durabilité d'un projet.

Les versions de codes comprimés des librairies

Le chargement du script engendre un temps de transfert du fichier vers le navigateur puis le temps d'exécution du script. Une librairie comme jQuery implique le chargement d'un fichier de plus de 240 Ko dans sa récente version si on utilise la version avec le code aménagé et commenté. Une version de production, comprimée, sans espaces ni commentaires existe du même code. Le fichier pèse moins de 90 Ko. Toujours privilégier les versions comprimées des librairies.

« Minifier » son propre code Javascript

En juin 2015, une page Web dispose en moyenne d'un poids de 2,3 Mo. Ce poids moyen par page ne cesse d'augmenter et crée de sérieux problèmes de performance. 16% de ce poids provient des fichiers Javascript (344 Ko). Bien que la puissance des supports mobiles ait augmenté, la connexion à Internet demeure instable et le temps de chargement des pages demeure long. Toute solution pour faciliter le temps de chargement d'une page devrait être envisagée. La « minification » du code (suppression des espaces et commentaires) permet de diviser le poids des fichiers Javascript. Un effort déjà louable dans l'économie de poids.

Les API et outils tiers en bas de page

Toujours dans le volet qui concerne le temps de chargement sont à considérer les API (Application programming interface) des outils tiers utilisés dans une page provenant de ressources externes. Les outils des réseaux et médias sociaux ou des cartes sophistiquées et adaptées sur mesure de Google Maps impliquent d'échanger des informations avec une interface distante (sur un autre serveur). Le temps de chargement de l'outil, l'échange de données et l'exécution du script ont un effet inquiétant sur le temps de chargement de la page. Les codes de ces outils tiers auraient tout intérêt à être logés en pied de page (avant la fermeture de la balise <body>) pour permettre au code HTML et CSS de s'exécuter et se charger sans délais supplémentaires.

Utiliser l'attribut `defer`

Il est possible d'insérer les balises `<script>` en haut de page (dans la balise `<head>`) et d'indiquer au navigateur de prévoir le chargement uniquement lorsque les codes HTML et CSS seront chargés. Ceci grâce à l'utilisation de attribut defer qui est interprété par IE10 et les navigateurs récents.

```
<script defer="defer" src="script.js"></script>
```

Attention cependant à ne pas différer le chargement des librairies telles que jQuery. Malheureusement il n'est pas (encore) possible d'indiquer l'ordre du chargement des fichiers différés avec cet attribut. Résultat, il est probable que la librairie se charge après un script qui en fait référence.

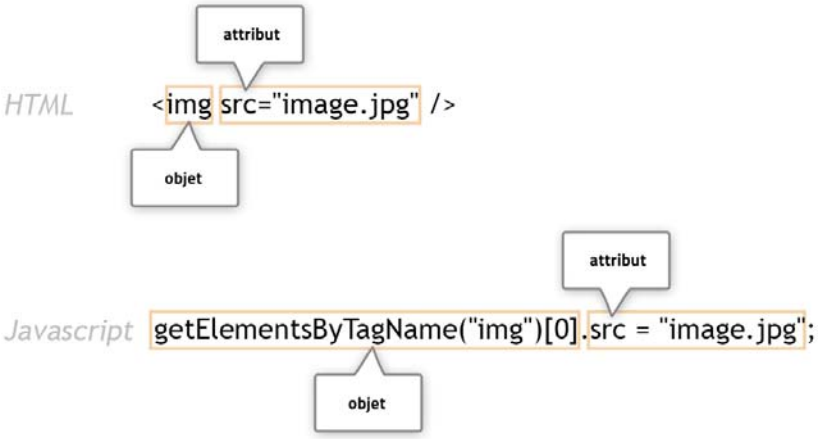
Créer des fichiers Javascript externes

Pour faciliter la maintenance et la distribution du code entre les fichiers HTML d'un projet (et cela se fait aussi pour le CSS), le code Javascript devrait se trouver dans un fichier à part. Il peut s'avérer utile de séparer le code d'animations spécifiques dans des fichiers distincts. On retrouverait ainsi un fichier *slideshow.js*, *parallax.js*, *menu.js*,… La maintenance du code sera plus aisée et les fichiers Javascript ne seront liés que dans les pages disposant de ces animations.

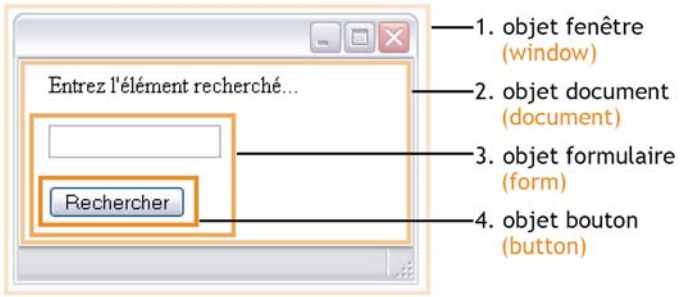
Le DOM

Javascript peut intervenir sur de nombreux composants : la fenêtre du navigateur, le document HTML, les balises HTML ainsi que leur contenu. Tout cela, grâce au DOM. Le DOM signifie **Document Object Model**. En d'autres termes, les composants sont disposés dans une arborescence et prennent la forme d'objets…

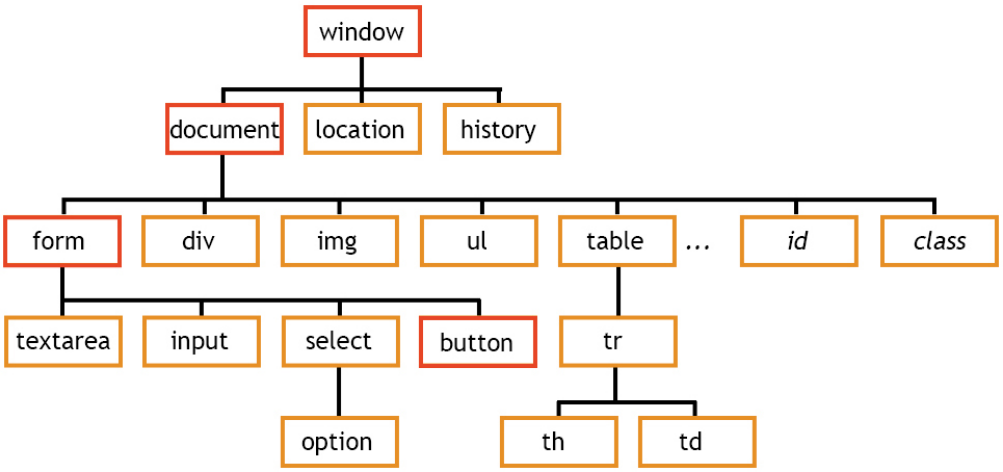
L'indication **« objet »** fait allusion à la programmation orientée objet. Une des références déterminantes avec cette méthode est qu'un objet dispose d'attributs et de méthodes. En s'attardant à la dénomination utilisée avec HTML, on constate que l'on parle également d'objets (les balises) et d'attributs. Une image `<img>` notamment dispose de l'attribut `src`.



Le HTML est une arborescence d'objets. Une balise `<html>` contient une balise `<body>` qui contient une balise `<form>` qui contient à son tour une balise `<bouton>`. Ce raisonnement s'applique à tous les éléments du DOM. Cette arborescence permet à Javascript d'accéder à tous les objets du document HTML.



Le repérage d'un objet se fait en fonction de sa position dans la structure du DOM. Il est possible d'accéder à un objet en partant du haut de la structure mais également depuis un autre objet en définissant sa position depuis celui-ci.



La forme Javascript pour accéder à l'objet HTML **« *button* »** peut s'envisager comme suit :

```
1  var button = window.document.getElementsByTagName("form")[0].¬
               getElementsByTagName("button")[0];
```

Une forme simplifiée de l'exemple précédent peut être utilisée en accédant directement à l'objet en ne faisant référence qu'au document :

```
1  var button = document.getElementsByTagName("button")[0];
```

Il est dorénavant possible d'utiliser une nouvelle forme qui fait référence à la syntaxe CSS. Cette méthode fonctionne sur navigateurs récents et IE9 et + :

```
1  var button = document.querySelectorAll("form button")[0];
```

Accéder aux balises HTML du DOM

L'accès aux objets du DOM permet de transformer l'apparence ou le comportement de cet objet. L'accès à ces objets peut se faire de plusieurs manières. Toutefois la méthode employée ne retourne pas le même type de contenu. Le type est soit un **objet** ou un **tableau d'objet**. Cette subtilité aura un incident sur la manière de traiter le contenu récupéré.

Les méthodes `querySelector()` et `querySelectorAll()` permettent de récupérer des éléments du DOM en utilisant la même syntaxe que celle du CSS pour définir un sélecteur. Ainsi le symbole [#] fait référence à un objet (balise) portant un attribut `id` et le symbole [.] fait référence à un attribut `class`. Les sélecteurs peuvent également être accompagnés des pseudo-classes et pseudo-éléments connus du CSS pour effectuer des sélections ciblées. Cette méthode est pratique et facile à utiliser pour ceux qui connaissent le CSS.

IE 9 et les versions plus récentes de ce navigateur reconnaissent et interprètent cette forme simplifiée tout comme les autres navigateurs récents. Dans les exemples qui suivent sont présentés les méthodes pour récupérer un objet ou un tableau d'objets.

Sélecteur	

Nom de balise	Récupère un **tableau d'objets** en fonction du nom que portent les balises.
	`var lis = document.getElementsByTagName('li');`
	`var lis = document.querySelectorAll('li');`

Valeur de l'attribut id	Récupère **un objet** portant l'attribut id recherché.
	Le nom donné à l'attribut id d'un objet HTML doit être unique dans une page.
	`var menu = document.getElementById('menu');`
	`var menu = document.querySelector('#menu');`

Nom d'une classe	Récupère un **tableau d'objets** HTML disposant du nom de la classe recherchée.
	`var lis = document.getElementsByClassName('list');`
	`var lis = document.querySelectorAll('.list');`

Le symbole point

Le symbole point est utilisé pour circuler d'un objet à un autre dans le HTML. Il complète souvent un énoncé en identifiant un attribut à qui une valeur est extraite ou modifiée ou en assignant une méthode afin de récupérer des informations d'un objet ou lui implémenter de nouvelles fonctionnalités.

Ainsi en voulant modifier la source d'une image sera identifié l'image par l'une des méthodes présentées précédemment en indiquant une nouvelle valeur à l'attribut `src`.

```
1  document.querySelectorAll( 'img' )[0].src = 'image-hover.jpg';
```

Dans ce cas-ci, le symbole point se traduit par **« contenu dans... »**. Le document contient une balise image qui contient un attribut src dont la valeur est dorénavant `'image-hover.jpg'`.

La direction ne va toutefois pas que dans un sens. Il est possible de remonter dans la structure du DOM ou consulter un **objet « frère »**. Le point sépare toujours les objets entre eux, leurs attributs et leurs méthodes.

```
1  document.querySelectorAll( 'li' )[0].parentNode.style.marginLeft = '10px';
```

Dans l'exemple qui précède, le premier symbole point sépare les objets entre eux et se traduit également par **« contenu dans... »**. Cependant l'attribut « `parentNode` » permet d'accéder à un objet parent. Ce qui se traduit par **« qui est contenu par... »**. En l'occurrence, puisqu'il s'agit du parent d'une balise `<li>` cela devrait correspondre à une balise `<ul>`. Le second point indique que l'on souhaite accéder à l'attribut style (de la balise `<ul>`. Le dernier point indique que c'est le style CSS se référant à la marge gauche qui est à modifier.

Combiner les solutions pour mieux cibler

Circuler dans la structure du DOM implique de connaître les balises HTML et leur disposition. La position d'un objet suffit à l'identifier et le distinguer des autres balises.

En souhaitant par exemple cibler les balises `<li>` disposées dans une balise `<ul id="menu">` il serait tout indiqué d'utiliser l'attribut `id` de la balise `<ul>` comme référence pour sélectionner les objets qu'elle contient. Cela évitera que toutes les balises `<li>` de l'entier de la page qui soient récupérées, comme dans l'exemple suivant.

```
1  var lis = document.querySelectorAll( 'li' );
```

```
1  var lis = document.querySelectorAll( 'ul#menu' )[0].querySelectorAll( 'li' );
```

Dans cet autre exemple ce ne sont que les balises `<li>` contenues dans la balise `<ul>` disposant d'un attribut `id` dont la valeur est « menu » qui sont sélectionnées. On peut faire encore plus simple avec l'exemple suivant.

```
1  var lis = document.querySelectorAll( 'ul#menu li' );
```

On obtiendrait toujours un tableau d'objets (`<li>`) comme dans l'exemple précédent. Et si je souhaitais n'avoir accès qu'au premier objet (`<li>`).

```
1  var li = document.querySelector( 'ul#menu li:first-child' );
```

On exploite ainsi les mêmes sélecteurs que le fait CSS. Dans l'exemple précédent, seul un objet est récupéré. Ceci grâce à la méthode `querySeletector()` et à la récupération du premier élément avec `first-child` dans l'énoncé du sélecteur.

Comment exploiter un objet vs un tableau contenant des objets.
Sélectionner une ou des balises HTML en Javascript se fait selon les différentes méthodes qui viennent d'être présentées. Comme indiqué, le résultat de la sélection prend l'apparence d'un objet ou d'un tableau de d'objets. Lorsqu'on fait référence à un tableau d'objets, il s'agit d'un **tableau de données (array)** qui dispose des différentes balises sélectionnées.

En prenant l'exemple de la sélection de balises `<li>` dans une page.

```
1  var lis = document.querySelectorAll( 'li' );
```

Le résultat aura l'apparence d'un tableau de données automatiquement composé de clés numérotées de 0 au nombre de balises trouvées.

```
1  var lis = |---cle---|-valeur-|
             |    0    |  <li>  |
             |    1    |  <li>  |
             |    2    |  <li>  |
             |    3    |  <li>  |
                ...
```

Étant logées dans ce tableau, les balises `<li>` peuvent être exploitées au travers d'une boucle ou en faisant appel à la clé du tableau correspondant à la balise à sélectionner, en l'indiquant entre les symboles crochets `[]`.

```
document.querySelectorAll( 'li' )[0].style.marginLeft = '15px';
```

Dans l'exemple précédent c'est la première balise (disposant de la clé « 0 ») qui verra la valeur de sa marge gauche modifiée à 15 pixels.

```
var lis   = document.querySelectorAll( 'li' );
var nbLis = lis.length;

for( i = 0; i < nbLis; i++ )
{
    document.querySelectorAll( 'li' )[i].style.marginLeft = '15px';
}
```

Dans cet autre exemple, toutes les balises `<li>` ont été modifiées. Une boucle `for()` passe en revue les objets du tableau en utilisant la position qu'ils y occupent par l'indication de la clé entre crochet et la variable `[i]` qui est automatiquement incrémentée en fonction du nombre de balises récupérées.

Une technique plus récente permet également l'exploitation d'un tableau d'objets avec `[].forEach.call()` qui permet l'exploitation d'un tableau par l'entremise d'une fonction anonyme.

```
[].forEach.call( document.querySelectorAll( 'li' ), function( li ){
    li.style.marginLeft = '15px';
});
```

Circuler dans le DOM

En repérant un objet dans le DOM, il est possible de circuler dans l'arborescence à partir de celui-ci en accédant à d'autres objets qu'il soient **« parent »**, **« frère »** ou **« enfant »**. En fonction de la position d'un objet dans le code ce qui est récupéré peut être une balise HTML, un attribut ou du texte (il existe d'autres possibilités telles que des commentaires).

Sélecteur	

Parent	Récupère l'élément parent (balise HTML, attribut, texte, ..). .
	`1 var parent = document.getElementById('logo').parentNode;`
	L'élément sélectionné est un objet
	
	Récupère la balise HTML parent.
	`1 var parent = document.querySelector('#logo').parentNode;`
	L'élément sélectionné est un objet

Frère précédent	Récupère l'élément (balise HTML, attribut, texte, ...) qui précède un objet.
	`1 var previous = document.getElementById('logo').previousSibling;`
	L'élément sélectionné est un objet
	
	Récupère la balise HTML qui précède un objet.
	`1 var previous = document.querySelector('#logo').previousElementSibling;`
	L'élément sélectionné est un objet

Frère suivant	Récupère l'élément (balise HTML, attribut, texte, ..) qui suit un objet.
	`1 var next = document.getElementById('logo').nextSibling;`
	L'élément sélectionné est un objet
	
	Récupère la balise HTML qui suit un objet.
	`1 var next = document.querySelector('#logo').nextElementSibling;`
	L'élément sélectionné est un objet

Enfant	Récupère les balises HTML contenues dans un objet. `1 var menusContent = document.getElementById('menu').children;` *L'élément sélectionné est un tableau (array) d'objets*

```
1 var menusContent = document.querySelector('ul#menu').children;
```

L'élément sélectionné est un tableau (array) d'objets

Premier enfant

Récupère le premier élément enfant d'un objet.

```
1 var menusFirst = document.getElementById('menu').firstChild;
```

L'élément sélectionné est un objet

```
1 var menusContent = document.querySelector('ul#menu li:first-child');
```

L'élément sélectionné est un objet

Dernier enfant

Récupère le dernier élément enfant d'un objet.

```
1 var menusContent = document.getElementById('menu').lastChild;
```

L'élément sélectionné est un objet

```
1 var menusContent = document.querySelector('ul#menu li:last-child');
```

L'élément sélectionné est un objet

Des sélections raffinées

Les méthodes `querySelector()` et `querySelectorAll()` permettent d'effectuer des sélections de balises HTML en utilisant la syntaxe CSS des pseudo-classes. Ceci offre la possibilité de faire des recherches pointues d'objets.

```
1 var element = document.querySelector('ul#menu:li:nth-child(odd)');
```

L'élément sélectionné est un objet

```
1 var element = document.querySelector('p:not(.intro)');
```

L'élément sélectionné est un objet

Créer dynamiquement du contenu dans le DOM

En plus de modifier l'apparence et le comportement d'objets dans le DOM, il est possible d'y introduire, remplacer ou retirer du contenu.

Écrire dans une balise HTML peut se faire par l'utilisation de la méthode `innerHTML()`. Elle remplacera le contenu existant par celui indiqué dans la parenthèse de la méthode `innerHTML()` (comme paramètre). Des balises peuvent être introduites également et prendront la mise en forme définie par le CSS.

```
document.querySelector( 'div#box' ).innerHTML( '<p>Contenu ajouté</p>' );
// Resultat : <div id="box"><p>Contenu ajouté</p></div>
```

Attention cependant, les éléments ajoutés ne peuvent avoir aucun événement assigné par cette méthode. Et ce même si un événement a été déclaré comme attribut.

```
document.querySelector( "div#box" ).innerHTML( '<button onclick="maFonction()">Clic
</button>' );
// Resultat : <button>Clic</button>
// Attention!!! Aucun appel à maFonction() au clic sur l'objet
```

Il est également possible d'ajouter un élément HTML avec la méthode `appendChild()`.

```
var element = document.createElement( 'a' );
element.setAttribute( 'href', 'http://www.site.com' );
element.textContent = 'Voir le site';
document.querySelector( "div#box" ).appendChild( element );
// Resultat : <div id="box"><a href="http://www.site.com">Voir le site</a></div>
```

Supprimer un élément HTML se fait avec la méthode `removeChild()`.

```
// <div id="content">
//   <div id="elementtodelete"></div>
// </div>
var elmToDelete = document.querySelector( 'div#elementtodelete' );
var content     = document.querySelector( 'div#content' );
content.removeChild( elmToDelete );
```

Les librairies de code (« framework »)

Il existe des dizaines et des dizaines de librairies de code Javascript. Certaines d'entre eux ont influencé la syntaxe de Javascript cherchant notamment à simplifier le mode de sélection des éléments dans le DOM donnant naissance aux méthodes `querySelector()` et `querySelectorAll()`.

Le but des librairies de codes est de simplifier l'accès aux éléments HTML, de proposer des animations sans avoir à tout coder et d'effectuer des opérations complexes qui doivent s'exécuter de la même façon sur les différents navigateurs et leurs nombreuses versions. De ces librairies existent **Prototype**, **MooTools**, **script.aculo.us**, **YUI** ou **jQuery**.

Des librairies facilitent l'exploitation des contenus, c'est le cas de **AngularJs** ou harmonisent l'affichage du HTML 5 dans les navigateurs avec **Modernizr**. Ou permettent encore d'envisager l'utilisation de Javascript sur un serveur Web avec **node.js** (note : ceci implique la mise en place d'une librairie sur le serveur).

Des plugins sont également disponibles pour des utilisations spécifiques, telles que l'implémentation d'un éditeur de contenus (WYSIWYG) avec **TinyMCE**, d'opérations particulières avec **Sortable**, **jQuery UI** ou encore la création de galeries d'images avec **LightBox** ou ses multiples déclinaisons.

Des **API** (Application programming interface) fournissent les ressources de sites tiers pour faciliter leur exploitation. C'est le cas de **Google Maps** pour l'utilisation et la modification de cartes géographiques ou encore **Google Charts** pour la création de graphiques statistiques. Les réseaux sociaux disposent de leurs API ce qui permet de créer des outils utilisant les ressources de ces sites. Que ce soit pour se connecter à une session ou afficher les contacts du réseau d'un visiteur, par exemple.

Quelques rappels

Petit retour sur quelques principes de base d'utilisation du langage Javascript. Ces principes et outils sont essentiels à une bonne utilisation du langage. Leur compréhension permet d'implémenter le code de manière efficace et plus aisé à maintenir.

Les variables

Une variable est une sorte de contenant permettant le transfert de données et l'altération de sa valeur au fil du déroulement des opérations. Une variable peut contenir différents types de contenus. La distinction des types est déterminante pour savoir employer une variable correctement.

La déclaration d'une variable

Les variables ont une portée plus ou moins limitée en fonction de leur emplacement, de leur déclaration et de la manière dont elles sont initiées. Le mot-clé `var` permet d'indiquer une variable dont la portée sera limitée à l'espace dans lequel elle a été déclarée. Elle est toutefois disponible dans les fonctions si elle a été déclarée à l'extérieur de celles-ci. Cette variable est considérée comme un attribut. Il s'agit d'un principe de l'orienté objet qui s'applique aux variables et aux fonctions.

Sans le mot-clé `var` la variable est globale. Sa valeur peut être modifiée à tout moment ce qui peut rendre la cohérence du code difficile et la maintenance périlleuse. A éviter donc.

Les variables se composent de caractères alphanumériques. Seuls les symboles tiret bas simple (_) et dollar ($) peuvent être utilisés dans le nom qui peut leur être donné.

Les variables locales

Elles se déclarent avec le mot-clé `var` devant. Leur portée se limite à l'espace dans lequel elles sont déclarées.

```
1  console.log( myVar ); // Error : La variable n'est pas définie (undefined)
2
3  function myFunction()
4  {
5      var MyVar = 7;
6      console.log( myVar ); // Affiche 7
7  }
```

Il est possible d'utiliser une variable dans une fonction bien qu'elle ait été déclarée à l'extérieur de cette fonction. Cette variable est considérée comme un attribut d'objet.

```
var myVar = 7;

console.log( myVar ); // Affiche 7

function myFunction()
{
    console.log( myVar ); // Affiche 7
}
```

Les variables globales

La variable globale se déclare sans le mot-clé `var`. Où qu'elle soit, elle est accessible à tout moment.

```
console.log( myVar ); // Affiche 7

function myFunction()
{
    myVar = 7;
    console.log( myVar ); // Affiche 7
}
```

Les types de contenus des variables

Javascript n'est pas strict dans sa façon de déclarer une variable ou modifier sa valeur. Aucun typage n'est à préciser. Cependant une utilisation adéquate des variables passe tout de même par la compréhension des types de valeur qu'elles peuvent contenir.

Types	

Chaîne de caractères	La chaîne de caractères (« string » en anglais) dispose d'un contenu composé de caractères, chiffres et symboles. Les guillemets simples ou doubles peuvent être employés.
	`var myVar1 = 'Ceci est du texte.';`
	`var myVar2 = "Ceci est du texte.";`
	`var myVar3 = 'L\'autre texte.';`

Valeur numérique	Une valeur numérique peut être intégrée à une opération mathématique. Dans le cas où une variable ne contient pas une valeur numérique et qu'elle est utilisée dans une opération mathématique, Javascript indique que la valeur de la variable est `NaN` (pour « not a number »).

```
1 var myN1 = 5;
2 var myN2 = 10;
3 var result = myN1 + myN2;
```

Booléen	Une valeur booléenne est une valeur qui est soit *true* ou *false*.
Null	Bien que vide, une valeur est définie à la variable. Il s'agit d'un objet.
Undefined	Indique qu'aucune valeur n'a été attribuée à la variable.
Tableau de données	Multiples valeurs inscrites dans un tableau. Chaque valeur dispose d'une clé permettant d'y faire référence. Les tableaux de données peuvent contenir des valeurs combinées de n'importe quel type (chaîne de caractères, valeur numérique, tableau de données, objets,...).

```
1  var jours = [ 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi' ];
2  // Affiche mardi
3  console.log( jours[1] );
4
5  var pers = [];
6  pers['prenom'] = 'John';
7  pers['nom'] = 'Doe';
8  // Affiche John
9  console.log( pers['prenom'] );
10
11 // Alternative
12 var pers = new Array();
13 pers['prenom'] = 'John';
14 pers['nom'] = 'Doe';
15 // Affiche John
16 console.log( pers['prenom'] );
```

Objets	Javascript a la particularité de tout considérer comme un objet. Le langage dispose des objets du DOM et de plusieurs composants (Array, Date, Math,...). Des objets peuvent être codés et ajoutés au langage. L'appel de la variable déclarée permet d'accéder aux attributs et méthodes de l'objet en ajoutant les symboles point (.).

```
1 var obj = new myObject();
2
3 // Accès a un attribut
4 obj.property;
5
6 // Accès a une méthode
7 obj.method();
```

Quelques astuces

Pour vérifier qu'une variable existe, il suffit d'utiliser l'opérateur `typeof` dans une condition. Elle retourne le type de valeur d'une variable et l'indicatif « `undefined` » est retourné dans le cas d'une variable non déclarée.

```
if( typeof myVar === 'undefined' )
{
    myVar = 7;
}
// typeof 'text'       === 'string';
// typeof 20           === 'number';
// typeof true         === 'boolean';
// typeof [1, 2, 3]    === 'object';
// typeof function(){} === 'function';
```

Transformer une chaîne de caractères en nombre à l'aide des méthodes `parseInt()` ou `parseFloat()`.

```
var integer = '20';
var float = '0.5';
console.log( integer * 2 ); // Error NaN
console.log( float * 2  );   // Error NaN

integer = parseInt( integer );
float    = parseInt( float );
console.log( integer * 2 ); // 40
console.log( float * 2  );   // 1
```

La concaténation

La concaténation est un processus par lequel les valeurs d'une ou plusieurs variables s'associent entre elles ou à une chaîne de caractères. En Javascript, le symbole plus (+) permet cette association.

```
var jours = 7;
console.log( 'Il y a ' + jours + 'dans une semaine' );

var pers = { prenom:'John', nom:'Doe' };
console.log( 'Votre nom est ' + pers.prenom + ' ' + pers.nom + ' ?' );
```

Les tableaux de données (array)

Les tableaux de données permettent de cumuler plusieurs valeurs dans un seul ensemble. Chaque valeur dispose d'une clé de référence ce qui facilite leur accès. Lorsqu'elles ne sont pas déclarées, les clés de références s'initient par le chiffre 0 et s'incrémentent automatiquement.

```
var jours = [ 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi' ];
/*
Le tableau $jours est conçu comme ceci
|-cle-|--valeur--|
|  0  | lundi    |
|  1  | mardi    |
|  2  | mercredi |
|  3  | jeudi    |
|  4  | vendredi |
*/

// Affiche mardi
console.log( jours[1] );
```

La clé de référence peut être déclarée et nommée.

```
var pers = [];
pers['prenom'] = 'John';
pers['nom']    = 'Doe';
/*
Le tableau $jours est conçu comme ceci
|---cle---|-valeur-|
| prenom  | John   |
| nom     | Doe    |
*/

// Affiche John
console.log( pers['prenom'] );
```

Un tableau peut disposer de plusieurs niveaux. En d'autres termes, un tableau peut en contenir d'autres.

```
var pers = [
      [ 'John', 'Doe' ],
      [ 'Jane', 'Dew' ]
];
/*
```

```
Le tableau pers est conçu comme ceci
|-cle-|----------valeur--------|
|  0  |  |--cle--|-valeur-|  |
|     |  |   0   | John   |  |
|     |  |   1   | Doe    |  |
|  1  |  |--cle--|-valeur-|  |
|     |  |   0   | Jane   |  |
|     |  |   1   | Dew    |  |
*/

// Affiche Jane
console.log( pers[1][0] );
```

Pour obtenir au tableau de plusieurs niveaux avec des clés définies, il faudra déclarer la valeur pour chaque clé.

```
var pers = [];
pers[0][ 'prenom' ] = 'John';
pers[0][ 'nom' ]    = 'Doe';
pers[1][ 'prenom' ] = 'Jane';
pers[1][ 'nom' ]    = 'Dew';
/*
Le tableau pers est conçu comme ceci
|-cle-|----------valeur--------|
|  0  |  |---cle---|-valeur-|  |
|     |  | prenom  | John   |  |
|     |  |  nom    | Doe    |  |
|  1  |  |---cle---|-valeur-|  |
|     |  | prenom  | Jane   |  |
|     |  |  nom    | Dew    |  |
*/

// Affiche Jane
console.log( pers[1]['prenom'] );
```

Les tableaux de données sont des objets…

Cela peut paraître surprenant. Par contre cela à l'avantage que les tableaux de données disposent d'attributs et de méthodes qui permettent d'exploiter leur contenu. D'ailleurs, Javascript considère que **tout ou presque est objet** : les tableaux de données (Array), les dates (Date), les opérations mathématiques (Math),...

Ainsi la déclaration d'un tableau peut également se faire selon cette méthode (il s'agit d'une alternative aux méthodes présentées précédemment).

```
1  var pers = new Array();
2
3  jours[ 'nom' ] = 'Doe';
4
5  console.log( jours );
```

L'attribut `length` est utilisé pour connaître le nombre de valeurs dans le tableau.

```
1  var jours = [ 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi' ];
2  var nbJours = jours.length ;
3  console.log( nbJours ); // 5
```

De même que la méthode `sort()` peut être utilisée pour organiser un tableau par ordre alphanumérique.

```
1  var jours = [ 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi' ];
2  var joursSort = jours.sort();
3  console.log( joursSort ); //[ 'jeudi', 'lundi', 'mercredi', 'mardi', 'vendredi' ];
```

Les tableaux de données et les boucles

L'exploitation d'un tableau de données peut se faire avec les boucles `for` et `while`. Dans la pratique la méthode la mieux adaptée est la boucle `for`. Elle implique cependant de compter le nombre d'éléments que compose le tableau avant d'effectuer l'opération de la boucle.

```
1  var jours = [ 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi' ];
2  var nbJours = jours.length;
3  for( i = 0 ; i < nbJours; i++ )
4  {
5        console.log( i ); // Affiche la clé (0, 1, 2, 3,...)
6        console.log( jours[ i ] ); // Affiche lundi, mardi, mercredi,...
7  }
```

Astuce : Afficher tout ce que contient un tableau de données

Il n'est pas rare que l'on souhaite consulter le contenu d'un tableau pour connaître sa structure et les valeurs qu'il dispose. En appelant la variable contenant le tableau dans la méthode console.log() le tableau complet sera affiché dans la console de développement du navigateur.

```
1  var jours = [ 'lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi' ];
2
3  console.log( jours );
```

Les fonctions

Une fonction permet de grouper une série d'opérations qui ne seront déclenchées que lorsque la fonction sera appelée. Elle se conçoit comme suit :

```
1  function changeImage()
2  {
3      document.querySelector('#image').src = 'images/image-selected.jpg';
4  }
5
6  // Exécute la fonction (et les opérations qu'elle contient)
7  changeImage();
```

Des valeurs peuvent être transmises lors de l'appel de la fonction. Ceci permet d'altérer le fonctionnement d'une fonction et de la rendre plus modulable. Ces valeurs passent par des paramètres définis lors de la création de la fonction. Les paramètres sont des variables uniquement accessibles dans la fonction.

```
1  function changeImage ( element, image )
2  {
3      element.src = image;
4  }
5
6  // Exécute une première fois la fonction
7  changeImage( document.querySelector('#image1'), 'images/image-selected.jpg' );
8
9  // Exécute une seconde fois la fonction
10 changeImage( document.querySelector('#image2'), 'images/image-selected.jpg' );
```

Les fonctions peuvent également retourner un résultat. Le type de format des valeurs retournées sont les mêmes que pour les variables (chaîne de caractères, valeur numérique, tableau de données ou objet).

```
function somme( param1, param2 )
{
    return param1 + param2;
}

console.log( somme( 4, 6 ) ); // Affichera 14

console.log( somme( 6, 6 ) ); // Affichera 12
```

La fonction anonyme

La fonction anonyme peut paraître particulière. Elle est appelée sans qu'un nom ne lui soit attribué et peut être associée à une variable. Elle est cependant pratique, particulièrement dans l'application dynamique d'événements à des objets HTML. Son utilité demeure de grouper des opérations comme le fait une fonction nominative. La différence s'observe dans l'utilisation qu'on en fait. Elle s'avère notamment très utile lors de l'assignation d'un événement à un objet (balise HTML).

Voici un premier exemple qui introduit une fonction anonyme dans une variable.

```
var somme = function( param1, param2 )
{
    return param1 + param2;
}

console.log( somme( 4, 10 ) ); // Affiche 14
```

Cet exemple pourrait se décliner de cette façon. La fonction est directement introduite dans une autre fonction sans avoir à la nommer. A noter que les valeurs des paramètres (variables mises entre parenthèses) doivent être déclarées avant leur introduction dans la fonction.

```
var param1 = 4;
var param2 = 10;

console.log( function(){ return param1 + param2 } ); // Affiche 14
```

Les fonctions anonymes sont très souvent associées à un événement (clic de bouton de souris, chargement de la page, appui d'une touche de clavier,...). Étant donné qu'une fonction permet de grouper des opérations, celles-ci se déclenchent lorsque l'événement est exécuté.

```
document.querySelector( '#btn' ).click = function()
{
    // Opérations menées lors du clic sur l'élément portant l'id btn
}
```

La fonction anonyme et ce qu'elle contient se déclenche lorsque l'utilisateur clique sur l'élément HTML qui dispose de l'attribut `id` dont la valeur est « btn ». La fonction anonyme permet de réserver l'exécution des opérations qu'elle contient qu'au moment de l'exécution de l'événement.

La fonction anonyme et le mot-clé « this »

Puisqu'une fonction anonyme est associé à un objet, il est possible de faire référence à cet objet dans la fonction grâce au mot-clé `this`.

```
document.querySelector( '#btn' ).click = function()
{
    this.style.opacity = 0.5;
}
```

Dans cet exemple, l'élément HTML qui dispose de l'attribut `id` dont la valeur est « btn » verra son opacité réduite de 50% lorsqu'un utilisateur cliquera dessus.

`this` est l'objet cliqué et est un point de repère dans la structure du DOM qui permet de récupérer les éléments parents, frères ou enfants qui l'entourent. Grâce à `this` certaines identifications sont grandement simplifiées.

Ainsi, dans une liste de rubriques conçues avec les balises `<ul>` et `<li>`, il est utile d'identifier l'élément à partir duquel un événement est déclenché pour mener les opérations souhaitées sur les objets qu'il contient.

```
document.querySelector( 'ul#menu li' ).click = function()
{
    this.style.opacity = 0.5;
}
```

Dans l'exemple qui précède, seul la balise `<li>` qui a été cliquée verra son opacité modifiée.

Les fonctions sont-elles des objets ?

Cela dépend... Il y a le principe et la pratique, soit **la manière d'utiliser la fonction**. Pour le principe, Javascript considère une fonction comme un objet. Ce qui rend ambigu la nature même d'une fonction (par le nom même qui lui est donné) est ce qui la distingue vraiment d'un objet.

En ce qui concerne la pratique, la manière dont une fonction est déclarée ou utilisée fera qu'elle ce qui distingue d'une fonction ou d'un objet. Une fonction s'utilise comme cela a été indiqué précédemment.

```
1  function somme( param1, param2 )
2  {
3      return param1 + param2;
4  }
```

Cependant, du moment où le mot-clé `new` est utilisé, la fonction se comporte plus clairement comme un objet.

```
1  function calcul( param1, param2 )
2  {
3      this.num1 = param1;
4      this.num2 = param2;
5  }
6
7  var operation = new calcul( 4, 6 );
8  operation.num1;  // Affiche 4
```

Cette fonction contient également des attributs qui agissent comme constructeur. Cette même fonction peut contenir d'autres fonctions qui seront considérées cette fois comme des méthodes comme dans un objet. Ceci sera détaillé dans la section qui concerne les objets.

Les objets

Les objets sont les éléments qui composent la programmation orientée objet (POO). Le concept et la manière de travailler avec les objets peut demander un effort particulier pour celui qui ne s'en est jamais servi. Pourtant, les objets se composent d'outils connus : de variables et de fonctions. Elles font respectivement référence aux attributs (variables) et aux méthodes (fonctions) d'un objet. De plus, la syntaxe utilisée en Javascript est déjà orientée objet... En fait, le DOM (Document Object Model) est par analogie le composant du HTML qui est composé d'objets (les balises) et de leurs attributs.

Cette relation entre l'orienté objet et la manière dont Javascript dispose de la structure du HTML est particulière mais a l'avantage de rendre accessible et compréhensible l'utilisation de cette méthode de développement.

Tout est objet

Javascript a la particularité de considérer tout comme objet. Des objets peuvent être conçus mais des objets sont proposés par le langage pour des utilisations communes, telles que les chaînes de caractères, les tableaux de données, les dates ou encore les opérations mathématiques. Même les fonctions ! Ce qui signifie qu'une fonction qui aurait la forme d'un objet peut contenir une autre fonction (qui serait elle une méthode !!!)...

Une particularité avec laquelle un développeur doit savoir composer. Mis à part le cas confus qui vient d'être cité, la compréhension et l'utilisation demeurent relativement simples. D'ailleurs, le format JSON qui sera traité plus loin dans ce document dispose également d'une forme objet. Ce qui illustre une certaine unité.

La composition d'un objet

Un objet dispose d'attributs (variables) et de méthodes (fonctions) qui se rapportent aux informations et manipulations dont dispose l'objet. A titre d'exemple, supposons qu'un objet serve à l'animation d'un diaporama (slideshow), les **attributs** disposeraient d'informations telles que le numéro de l'image affichée ou encore le nombre total d'images du diaporama. A cet objet s'ajouterait des **méthodes** qui effectueraient les animations de changement d'images ou encore les actions lorsque l'utilisateur clique sur la flèche de droite, de gauche encore une des vignettes.

Javascript propose deux formes de syntaxe pour les objets... La première forme est celle employée par plusieurs langages de programmation qui permet de créer une classe qui sera instanciée lors de la création d'objets. La particularité de Javascript est que le terme « function » est employé pour la création de la classe... De cette façon un constructeur (fonction d'initiation de la classe) est disponible. Voici un exemple de la composition et l'utilisation d'une classe en Javascript.

```
1  function SlideShow( speed )
2  {
3          // Attributs
```

```
4       this.speed =  speed;
5
6       // Méthodes
7       this.slideRight = function()
8       {
9             // Opérations
10      }
11  }
12  // slide1 est un objet de MyFileClass
13  var slide1 = new SlideShow( 1000 );
14
15  // Accède à une méthode de la classe slideShow
16  slide1.slideRight;
```

La seconde forme de syntaxe crée directement l'objet sans qu'elle n'ait à être instanciée. Elle se dit « littérale ». Elle utilise une toute autre syntaxe qui est commune à JSON telle que définie par ECMAScript.

```
1   var Personnage =
2   {
3       // Attributs
4       force: 9,
5
6       // Méthodes
7       attack: function()
8       {
9             // Opérations
10      }
11  }
12  // Accède à la méthode de l'objet Personnage
13  Personnage.attack();
```

Il est tout de même possible de créer un objet de cet objet avec la méthode `Object.create()`. A partir du précédent exemple, voici comment cela se définit.

```
1   var personnage2 = Object.create( personnage );
2
3   personnage2.force = 10;
4   personnage2.attack();
```

Voici un petit lexique et quelques exemples des termes employés dans l'orienté objet.

Termes		

Classe	Elle est le système complet disposant des informations et manipulations prévues.	`function MySlideClass( parametre )`
		`{`
		`  // Attributs`
		
		`  // Méthodes`
		`}`

Objet	A ne pas confondre avec la classe. Pourtant ils sont intimement liés. Un objet est en fait défini pour chaque élément utilisant une classe. Lorsque qu'un objet est créé, il s'agit d'une instance de la classe.	`// mySlide1 est un objet de MySlideClass`
		`var mySlide1 = new MySlideClass();`
		`// mySlide2 est autre un objet de MySlideClass`
		`var mySlide2 = new MySlideClass();`
		
		`// Accède à l'attribut concernant la vitesse`
		`console.log( mySlide1.speed );`

Instance	Instance et objet font référence à la même chose. Il s'agit de synonymes. On parle d'instance d'une classe lorsqu'un objet est créé.	

Attribut (ou propriété)	Il s'agit de variables disposées dans la classe qui recensent des informations.	`function MySlideClass()`
		`{`
		`  this.speed = 1000;`
		
		`  // Méthodes`
		`}`

Méthode	Il s'agit de fonctions qui permettent d'effectuer des opérations spécifiques.	`function MySlideClass()`
		`{`
		`  // Attributs`
		
		`  var slideRight = function()`
		`  {`
		`  // Opérations`
		`  }`
		`}`

Les objets intégrés

Javascript définit des objets pour de nombreux éléments, ce qui permet d'accéder à des attributs et méthodes propres à ces objets.

La syntaxe de Javascript indique que des fonctions (méthodes) et attributs utilisés proviennent de ces classes intégrées. On peut le constater avec l'exemple suivant où l'attribut `length` est associé à un tableau de données (array).

```
var days = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi'];
days.length; // 5
```

La classe employée donne accès aux méthodes qui lui sont propres. Considérant que les classes ne sont pas explicitement déclarées, comme c'est le cas avec l'exemple précédent où la méthode `length` de la classe `Array()` est employée sans avoir déclaré le nom de cette classe, il est important de comprendre les types de classes intégrées afin d'accéder aux attributs et méthodes dont elles disposent. Voici la liste des classes intégrées de Javascript.

Objets		

Chaîne de caractères (string)	Transformer une chaîne de caractères à l'aide des méthodes de la classe `String()`.	`var s = new String( 'images/1.jpg' );`
		`s.split( '/' ); // ['images', '1.jpg']`
		
		`// Syntaxe simplifiée`
		`var s = 'images/1.jpg';`
		`s.split( '/' ); // ['images', '1.jpg']`

Nombre (number)	Transformer un nombre (entier ou décimal) à l'aide des méthodes de la classe `Number()`.	`var n = new Number( 3.1614 );`
		`n.toFixed(); // 3.16`
		
		`// Syntaxe simplifiée`
		`var n = 3.1614;`
		`n.toFixed(); // 3.16`

Array	Intervention sur les valeurs et les composantes de tableaux de données à l'aide des méthodes de la classe `Array()`.	`var s = new Array( '1.jpg', '2.jpg' );`
		`s.push( '3.jpg' );`
		
		`// Syntaxe simplifiée`
		`var s = [ '1.jpg', '2.jpg' ];`
		`s.push( '3.jpg' );`

Math	Opérations mathématiques sur un nombre avec l'une des méthodes concernées. La particularité de cette classe est quelle est statique. Il n'y a pas à la déclarer, ce qui autrement entraînerait une erreur.	`Math.floor( 3.1416 ); // 3`

Date	Formate et calcule les dates.	`var d = new Date( 2015, 11, 25 );` `d.getDay(); // 25`

Expressions régulières	Les expressions régulières disposes de méthodes permettant de vérifier le format et contenu d'une chaîne de caractères.	`var str = 1032;`
		`var regex = new RegExp( ^\d{4,5}$ );`
		`regex.test( str ); // True`
		
		`// Syntaxe simplifiée`
		`var str = 1032;`
		`var regex = ^\d{4,5}$ ;`
		`regex.test( str ); // True`

Booléen (boolean)	Classe statique faisant référence aux conditions vraies (`true`) ou fausses (`false`). Comme pour la classe Math, il n'y a pas à la déclarer.	`var b = Boolean( 2 );  // True`
		`var b = Boolean( 0 );  // False`
		`var b = Boolean( 3 > 2 );  // True`
		`var b = Boolean( 'str' );  // True`
		`var b = Boolean( '' );  // False`

Les objets du BOM

Javascript dispose d'objets faisant référence aux composants du navigateur (BOM – Browser Object Model) et son environnement. Ceux-ci disposent de méthodes et attributs qui permettent de récupérer des informations ou d'en transformer le comportement.

L'objet History

L'objet `history` appartient à l'objet `window`. Comme son nom l'indique, il est possible d'intervenir et d'exploiter l'historique de navigation. Cet objet ne dispose que de la classe `length` et des méthodes `back()`, `forward()` et `go()`.

```
history.back(); // Enverra l'utilisateur à la page précédemment consultée.
```

L'objet Location

L'objet `location` appartient à l'objet `window`. Il dispose des informations concernant l'URL. Il permet ainsi de recueillir des informations provenant de l'URL ou de rediriger l'utilisateur vers une nouvelle page.

```
location.assign( 'http://www.site.com' ); // Ouvre la page www.site.com
```

L'objet Navigator

L'objet `navigator` est un objet appartenant à l'objet `window`. Permet d'obtenir des informations que le navigateur peut transmettre. Il est à noter que les navigateurs ne mettent pas tous de la même façon ces informations.

```
navigator.geolocation.getCurrentPosition( function( position ){
      console.log( position.coords.latitude );  // La latitude du navigateur
});
```

L'objet Screen

L'objet `screen` est un objet appartenant à l'objet `window`. L'objet `screen` dispose d'informations concernant l'écran de l'utilisateur.

```
screen.height; // Indique la hauteur de la fenêtre de l'utilisateur
```

L'objet Window

L'objet `window` contient tous les autres composants HTML de la page. Il fait référence à la fenêtre du navigateur et dispose des informations du document. Les attributs de `window` permettent de recueillir des informations sur la fenêtre.

```
window.pageYOffset; // Nb de px du déroulement de la page dans le navigateur.
```

Les méthodes de l'objet `window` donnent accès à certaines fonctionnalités du navigateur.

```
window.print(); // Ouvre la fenêtre d'impression de la page.
```

Les objets du DOM

Tous les objets du DOM (Document Object Model) sont intégrés dans l'objet `window`. Ces objets ont comme référence commune l'objet document qui se trouve au sommet de cette structure. Il dispose lui-même d'attributs et méthodes qui permettent de conditionner le comportement des composants de la page HTML.

L'objet Document

L'objet document intervient sur le contenu de la page HTML. Les possibilités qu'offre cet objet sont très nombreuses. Il peut notamment composer du code HTML et du contenu, ajouter un événement, vérifier l'état de

chargement de la page, indiquer la date de modification de la page, récupérer tout les composants des balises <head>, <body> ou d'un élément spécifique.

Crée un élément HTML

```
1 var newDiv = document.createElement( 'div' ); // Créer un objet <div>
2 newDiv.setAttribute( 'id', 'newarea' );
```

Récupère des éléments de la page

```
1 var imgs = document.querySelectorAll( 'img' ); // Récupère les <img> de la page
```

Attributs et méthodes des objets du DOM

Les balises, attributs et contenus de la page HTML sont également considérés comme des objets et disposent donc d'attributs et de méthodes. Ils servent à récupérer des informations concernant l'élément ou à le transformer.

```
1 var element = document.querySelector( 'div#page' );
2 element.innerHTML( '<p>Page</p>' ); // Inscrit du texte dans <div id="page">
```

Les événements

Les événements sont des méthodes attribuées aux objets disposés dans le document.

Le déclenchement d'opérations succède à un événement qui est effectué soit par l'utilisateur qui manipule les éléments de la page et la fenêtre à l'aide de la souris, de l'écran tactile ou du clavier, soit par le processus de chargement de la page qui signale notamment les erreurs rencontrées ainsi que le terme du processus.

Dans les tableaux qui suivent sont décrits les événements courants accompagnés d'une liste de balises à qui ces événements peuvent être attribués.

La fenêtre

Evénement	Description	Balise(s) concernée(s)
onload	La page Web est chargée dans le navigateur.	<body>, <img>, <script>, <link>, <iframe>
onerror	Le code de la page génère une erreur.	<body>, <img>
onresize	Redimensionner la fenêtre.	<body>
onscroll	Dérouler la fenêtre du navigateur.	<body>
onunload	Quitter la page Web.	<body>

Evénement	Description	Balise(s) concernée(s)
onfocus	Dans un formulaire, un élément est sélectionné.	<body>, <select>, <input>, <textarea>
onblur	Un élément précédemment activé est désactivé par un clic.	<body>, <select>, <input>, <textarea>
onchange	Dans un formulaire, changement de valeur dans un champ.	<select>, <input>, <textarea>
onselect	Dans un formulaire, un champ est sélectionné.	<select>, <input>, <textarea>
onsubmit	Dans un formulaire, cliquer sur le bouton d'envoi du formulaire.	<form>
onreset	Dans un formulaire, les données sont effacées en appuyant sur le bouton « reset ».	<form>

Le clavier

Evénement	Description	Balise(s) concernée(s)
onkeydown	Enfoncer une touche du clavier.	<body>, <a>, <img>, <input>, <textarea>
onkeypress	Appuyer une touche du clavier.	<body>, <a>, <img>, <input>, <textarea>
onkeyup	Relâcher une touche appuyée du clavier.	<body>, <a>, <img>, <input>, <textarea>

La souris

Evénement	Description	Balise(s) concernée(s)
onclick	Cliquer sur un élément.	<a>, <select>, <input>
ondblclick	Double-cliquer sur un élément.	<body>, <a>
onmouseover	Déplacer le curseur de la souris sur un élément.	*(toutes)*
onmouseout	Déplacer le curseur de la souris hors d'un élément.	*(toutes)*
onmousedown	Enfoncer le bouton de la souris sur un élément.	*(toutes)*
onmouseup	Relâcher le bouton de la souris sur un élément.	*(toutes)*
onmousemove	Déplacer la souris.	*(toutes)*
onwheel	Dérouler la molette de la souris.	*(toutes)*
oncontextmenu	Faire apparaître le menu contextuel à l'aide du bouton droit de la souris.	*(toutes)*

L'écran tactile

Evénement	Description	Balise(s) concernée(s)
ontouchstart	Enfoncer le bouton de la souris sur un élément.	*(toutes)*
ontouchleave	Relâcher le bouton de la souris sur un élément.	*(toutes)*
ontouchmove	Déplacer la souris.	*(toutes)*

`ontouchcancel`	Interruption spontanée du « touch » suite à une interférence générée par un autre événement ou un élément externe (popup, sortie de fenêtre).	*(toutes)*

Le presse-papier

Evénement	Description	Balise(s) concernée(s)
`oncopy`	Copier une image ou du texte.	*(toutes)*
`oncut`	Couper une image ou du texte.	*(toutes)*
`onpaste`	Coller une image ou du texte.	*(toutes)*

Les médias

Evénement	Description	Balise(s) concernée(s)
`onprogress`	Lorsque le média se charge dans la mémoire.	`<video>`, `<audio>`
`oncanplay`	Lorsque le média est en mesure d'être lu (une fois chargé dans la mémoire).	`<video>`, `<audio>`
`onplay`	Lorsque le média est en cours de lecture.	`<video>`, `<audio>`
`onpause`	Lorsque le média est en pause.	`<video>`, `<audio>`
`onended`	Lorsque le média atteint la fin de la lecture.	*`<video>`, `<audio>`*
`onwaiting`	Lorsque le média est en attente (mise en mémoire).	*`<video>`, `<audio>`*
`onvolumechange`	Lorsque le volume change ou est éteint.	*`<video>`, `<audio>`*

Assigner un événement à un objet

L'assignation d'un événement permet d'indiquer ce qui se fera lorsque l'événement aura lieu. Cette assignation implique de repérer l'objet puis lui indiquer l'événement qui lui sera assigné.

```
1 document.querySelector( 'div#box' ).onclick = function(){
2     // Opérations prévues suite au déclenchement de l'événement.
3 };
```

```
1 document.querySelector( 'div#box' ).addEventListener( 'click', function(){
2     // Opérations prévues suite au déclenchement de l'événement.
3 }, false;
```

Retirer un événement préalablement assigné

Bien qu'assigné à un objet un événement peut être retiré. Ceci implique cependant de définir une fonction qui indiquera l'action à opérer lorsque l'événement sera déclenché. Car il s'agira d'une référence à qui il faudra à nouveau appeler lors de la suppression de l'événement.

```
var clickOperations = function()[
    // Opérations prévues suite au déclenchement de l'événement.
}

document.querySelector( 'div#box' ).addEventListener( 'click', clickOperations, false
);

document.querySelector( 'div#box' ).removeEventListener( 'click', clickOperations,
false );
```

Intervention sur l'objet événement

Lorsqu'un événement appelle une fonction, il est possible d'utiliser les informations de l'événement ou de conditionner son comportement à l'aide d'objets intrinsèques à l'événement. Cela peut s'avérer utile pour connaître la touche appuyée par l'utilisateur sur son clavier ou la position du curseur de la souris, par exemple. Pour cela, il suffit d'introduire un paramètre à la fonction. Elle disposera des méthodes et attributs de l'objet événement utilisé.

```
document.querySelector( '#box' ).onkeypress = function( event ){
    // La variable event dispose des attributs et méthodes de l'événement déclenché
    // event.clientX : Position horiz. en px de la souris par rapport à la fenêtre
    // event.clientY : Position vert. en px de la souris par rapport à la fenêtre
    // event.which : Indique par un numéro la touche enfoncée par l'utilisateur
    // ...
};
```

Retirer un événement existant

La balise `<a>` dispose également d'un événement `onclick` qui lui est attribué du moment où l'attribut `href` a été indiqué. Pour retirer cet événement, il suffit d'appeler la méthode `preventDefault()`.

```
document.querySelector( 'a#link' ).onclick = function( e ){
    e.preventDefault();
};
```

Supprimer la propagation d'un événement

En assignant un événement à un élément HTML, les éléments « enfants » disposeront également de l'événement. En d'autres termes s'il est question de rendre cliquable une balise `<ul>` les balises `<li>` qu'elle contient déclencheront également l'opération sous-entendue par le clic de la souris. Il est sans doute dans ce cas souhaité de supprimer la propagation de l'événement afin qu'il ne soit réservé qu'à la balise pour laquelle il a été défini.

```
1 document.querySelector( 'div#box' ).onclick = function( e ){
2     e.stopPropagation();
3 };
```

La position du curseur de la souris

Le déplacement d'un objet dans l'interface Web est possible en relevant la position du curseur de la souris en utilisant l'événement `onmousemove()`. En le combinant avec les événements `onmousedown()` et `onmouseup()`, il est possible de créer une opération de « drag n'drop ».

```
1  document.querySelector( '#box' ).onmousedown = function(){
2      var element = this;
3
4      var dragndrop = function( e ){
5          var positionX = e.clientX;
6          var positionY = e.clientY;
7          element.style.position = 'absolute';
8          element.style.left = positionX + 'px';
9          element.style.top = positionY + 'px';
10     }
11
12     document.addEventListener( 'mousemove', dragndrop, false );
13
14     document.onmouseup = function( e ){
15          document.removeEventListener( 'mousemove', dragndrop, false );
16     }
17 };
```

La même opération est possible sur mobile avec l'événement `touchmove()`, `touchstart()` et `touchend()`. L'extraction de la position du curseur (représenté par le doigt) est possible avec l'attribut `changedTouches`. Sera également ajouté la méthode `preventDefault()` qui supprime l'effet par défaut du défilement lors du déplacement du doigt sur un support mobile qui est de faire dérouler la page.

```
1  document.querySelector( '#box' ).ontouchstart = function(){
```

```
    var element = this;

    var dragndrop = function( e ){
        var touchobj = e.changedTouches[0];
        var positionX = touchobj.clientX;
        var positionY = touchobj.clientY;
        element.style.position = 'absolute';
        element.style.left = positionX + 'px';
        element.style.top = positionY + 'px';
        e.preventDefault();
    }

    document.addEventListener( 'touchmove', dragndrop, false );
    document.ontouchend = function( e ){
         document.removeEventListener( 'touchmove', dragndrop, false );
    }
};

```

Les touches du clavier

A l'utilisation du clavier il est possible de détecter la touche enfoncée par l'utilisateur en relevant le numéro du chiffre avec l'attribut `which` de l'objet événement (l'attribut `keyCode` est utilisé pour IE).

```
document.onkeydown = function( e ){
    var keyDowned = e.which || e.keyCode;
    if( keyDowned === 37 ){
        // Operation lorsque la flèche gauche du clavier est activée
    }else if( keyDowned === 38 ){
        // Operation lorsque la flèche vers le haut du clavier est activée
    }else if( keyDowned === 39 ){
        // Operation lorsque la flèche droite du clavier est activée
    }else if( keyDowned === 40 ){
        // Operation lorsque la flèche vers le bas du clavier est activée
    }else{
        // Operation lorsque toute autre touche du clavier est activée
    }
};
```

Méthode d’assignation d'un événement à un objet

Afin de préserver le HTML de tout code Javascript et ainsi les dissocier, il est indiqué d'assigner les événements aux objets depuis le code Javascript. Ceci afin d'éviter de faire appel aux attributs qui font référence aux événements dans le HTML. Comme `<a href="javascript:exeMyJsFunction()">` ou encore `<img onclick="exeMyJsFunction()">` que l'on peut considérer comme une mauvaise pratique. Le code HTML s'en trouve alourdi d'un code étranger et la maintenance en sera plus difficile à assurer.

Assigner un événement en 4 étapes

Ces étapes sont un moyen de mettre en œuvre un développement permettant de franchir étape par étape un processus logique ce qui, de ce fait, est plus simple à comprendre. Forcément ce processus sera étoffé d'opérations complémentaires qui peuvent s'avérer plus complexes. Il demeure que la démarche proposée permet d'assigner des événements aux objets HTML depuis le Javascript ce qui assure une distinction des rôles.

1. **Chargement des éléments de la page**

   Objectif : S'assurer que la page et son contenu soient chargées.

   Cette étape préliminaire sert à s'assurer que l’assignation des événements aux éléments du HTML ne se fera qu’une fois que la page sera chargée dans le navigateur et surtout que ces éléments HTML seront disponibles.

   A cette fin est associé l'événement `onload()` à l'objet `window`. C'est à l'intérieur de la fonction anonyme déclarée que seront associés les événements.

```
1  window.onload = function(){
2
3  }
```

2. **Assigner les événements**

   Objectif : Désigner la liste des événements prévus dans l'interface.

   Associer les événements aux objets de l'interface et leur déclarer la fonction anonyme qui contiendra les opérations à mener lorsque les événements seront déclenchés.

```
1  window.onload = function(){
2
3        document.querySelector( 'li' ).onclick = function(){
4
```

```
5         }
6
7         document.querySelector( 'ul.menu li' ).onmouseover = function(){
8
9         }
10
11        document.querySelector( 'ul.menu li' ).onmouseout = function(){
12
13        }
14  }
```

3. **Identifier le ou les objets à modifier**

   Objectif : Cibler les objets et contenus à modifier dans l'interface.

   Cette étape implique d'utiliser les outils de sélection d'objets. Ceux-ci permettent de sélectionner un seul objet, avec la méthode `querySelector()`, ou un groupe d'objets avec `querySelectorAll()`. Il peut s'avérer utile de relire le chapitre sur le DOM de ce document qui traite de la manière d'exploiter correctement ces outils.

   Il est possible d'identifier les objets à partir de l'élément à qui l'événement a été assigné. Dans le cas, par exemple, où celui-ci contient ces objets. Le mot-clé `this` fait justement référence à l'élément à qui la fonction anonyme a été associée.

   L'exemple qui suit utilise les différents cas de figure exposés.

```
1   window.onload = function(){
2
3         document.querySelector( 'ul.menu li' ). onmouseover = function(){
4
5               // Toutes les balises <img> contenu dans 'div#page'
6               document.querySelectorAll( 'div#page img' );
7
8               // Balise <a> contenu dans le 'ul.menu li' en cours de survol
9               this.querySelector( 'a' );
10
11        }
12        //...
13  }
```

4. **Modifier un ou des attributs de l'objet**

Objectif : Opérer les transformations sur les objets et/ou contenus.

Cette dernière étape implique d'effectuer les transformations aux objets précédemment sélectionnés. Plusieurs possibilités de transformations sont possibles :

Modifier les valeurs d'un attribut HTML

En ayant identifié un élément qui fait référence à une balise HTML, il est possible de modifier la valeur d'un de ces attributs. Il suffit de déclarer le nom de cet attribut pour en modifier la valeur.

```
1  this.querySelector( 'a' ).href = 'http://www.site.com';
```

Assigner une classe CSS

En référence à l'exemple précédent, il est possible d'assigner une nouvelle classe qui est prédéfinie dans le CSS et qui permettra de donner une nouvelle apparence à l'élément identifié. Attention toutefois, étant donné que le mot-clé `class` est réservé, le nom de l'attribut a été transformé en Javascript par `className`.

```
1  this.querySelector( 'a' ).className = 'selected';
```

Assigner du code CSS depuis Javascript

En référence à ce qui a été présenté comme options, il est possible d'utiliser l'attribut `style` pour faire appel aux propriétés CSS qui peuvent être appliquées directement dans le Javascript.

```
1  this.querySelector( 'a' ).style.color = '#00f';
2  this.querySelector( 'a' ).style.textDecoration = 'underline';
```

Il est à noter que les propriétés CSS disposant d'un trait d'union s'emploient en retirant le trait d'union et en assignant une majuscule au second mot. Par exemple, `text-decoration` devient `textDecoration` ou `background-image` devient `backgroundImage`.

Modifier le contenu de l'élément

Une autre possibilité est de modifier le contenu disposé dans l'élément. Remplacer du contenu peut se faire à l'aide de l'attribut innerHTML comme ceci :

```
1  this.querySelector( 'a' ).innerHTML = 'Hyperlien'; // <a>Hyperlien</a>
```

Il est également possible d'un intégrer de nouveaux objets, comme ceci :

```
1  var span = document.createElement('span');
2  this.querySelector( 'a' ).appendChild = span; // <a><span></span></a>
```

Rappel :

S'il est question de modifier un groupe d'objets sélectionnés à l'aide de la méthode `querySelectorAll()`, il faudra employer une boucle pour opérer les transformations de chaque élément.

```
1  var imgs   = document.querySelectorAll( 'div#page img' );
2  var nbImgs = imgs.length;
3
4  for( i = 0; i < nbImgs; i++ )
5  {
6      document.querySelectorAll( 'div#page img' )[i].style.marginLeft = '15px';
7  }
```

La dynamique de transmission des contenus

Javascript peut exploiter dynamiquement des contenus, ce qui lui donne une vocation supplémentaire qui va au-delà de l'animation des interfaces. Des échanges de données dynamiques peuvent se faire entre Javascript et un serveur distant grâce à la méthode AJAX.

AJAX

Grâce à AJAX (Asynchronous Javascript And XML) il est possible de faire des transferts de données et de communiquer avec un serveur distant sans que la page n'ait besoin d'être rechargée. Comme son nom l'indique, l'opération est ce qu'on appelle asynchrone.

Les échanges avec le serveur se font à l'aide du protocole HTTP. Ce protocole permet d'échanger avec un serveur et d'obtenir une réponse de celui-ci indiquant l'état du traitement de la demande accompagné des contenus en cas de succès. HTTP transmet donc des informations du client (navigateur) au serveur dans un sens comme dans l'autre.

Voici les composants des informations transmises entre le client et le serveur via HTTP.

Envoi HTTP du client au serveur

Exemple de transmission d'informations du client au serveur.

```
GET /page.html HTTP/1.0
Host: example.com
Referer: http://example.com/
User-Agent: Opera/9.25 (X11; Linux i686; U; fr-ca)
```

Ligne de commande

Indique la méthode de transfert des données et l'action attendue du serveur. Les méthodes fréquentes provenant d'un site Web sont GET et POST. Il existe également HEAD, OPTIONS, CONNECT, TRACE, PUT, PATCH ou DELETE.

Sont également spécifiés l'URL et la version du protocole HTTP.

Entête de requête

Précise le nom de domaine (Host), l'adresse de provenance du document qui est à l'origine de la requête (Referer) et le nom du logiciel de transfert de la demande (User-Agent).

Réponse HTTP du serveur au client

Exemple d'une réponse transmise du serveur au client.

```
HTTP/1.0 200 OK
Date: Fri, 15 Jan 2016 23:59:59 GMT
Server: Apache/0.8.4
Content-Type: text/html
Content-Length: 33
Expires: Wed, 01 Jan 2020 00:59:59 GMT
Last-modified: Fri, 25 Dec 2015 14:21:40 GMT

<div>Contenu de la reponse.</div>
```

Ligne de statut

Indique le statut de la réponse. Voici les principaux statuts des serveurs :

200 indique que la requête a été traitée avec succès.

403 Indique que l'accès à cette ressource est interdite.

404 indique que la ressource n'est pas trouvée.

500 indique que le serveur a une erreur.

Entête de réponse

Indique la date de la réponse (Date), par quel logiciel serveur (Server), avec quel type MIME de ressources transmises (Content-Type), disposant de quel poids en octets (Content-Length), expirant (Expires) et ayant été modifié (Last-modified) à quelle date

Corps de réponse

Le contenu de la ressource transmise.

L'objet XMLHttpRequest

Javascript effectue une requête à un serveur distant grâce à l'objet XMLHttpRequest. Cet objet est en mesure de transmettre les données au serveur et traiter la réponse afin de permettre l'utilisation du contenu.

XMLHttpRequest

Attributs

readyState	Etat de transfert des données. Passe successivement de 0 à 4. 0 : non initialisée — La méthode open n'a pas encore été appelée 1 : en chargement — Le premier appel HTTP :: open() 2 : chargée — Les entêtes HTTP ont été reçues :: send() 3 : en traitement — Le fichier est en cours de chargement 4 : terminée — Le chargement de la ressource est complétée
status	Code de réponse du serveur 200 est ok, 404 si la page n'est pas trouvée...
statusText	Message du serveur en relation au status.
responseText	Contient les données chargées dans une chaîne de caractères.
responseXml	Contient les données chargées sous forme xml, les méthodes du DOM servent à les extraire.
onreadystatechange	Propriété activée par un évènement de changement d'état. On lui assigne une fonction.

Méthodes

open(mode, url, boolean)	Connexion avec le serveur mode — Type de requête, GET ou POST url — Chemin du fichier sur le serveur. boolean — Asynchrone (true) / Synchrone (false). en option, il est possible d'ajouter un login et un mot de passe.
setRequestHeader(type, genre)	Définit l'entête HTTP envoyée. Nécessaire pour l'envoi par la méthode POST (par exemple) : type — "Content-Type" genre — "application/x-www-form-urlencoded"
send(params)	Envoi de la requête au serveur. Méthode GET — null Méthode POST — Tansfert des données (ex. : "param1=valeur&param2=valeur")
abort()	Arrête le processus de l'opération.

L'objet `XMHttpRequest` peut s'utiliser sur tous les navigateurs depuis la version 10 de IE. L'adaptation qui suit est toutefois nécessaire pour les versions antérieures. Elle consiste à détecter l'objet équivalent à `XMHttpRequest` existant dans le navigateur pour l'exécution des échanges avec le serveur. En l'occurrence, il s'agit de l'objet `ActiveXObject` disponible dans le navigateur Internet Explorer. Une fois l'objet détecté il sera transmis de manière à exploiter ses ressources (attributs et méthodes).

```
var initXhr = function(){
	var xhr;
	try{ xhr = new XMLHttpRequest(); }
	catch( try_microsoft1 ){
		try{ xhr = new ActiveXObject( "MSXML2.XMLHTTP" ); }
		catch( try_microsoft2 ){
			try{ xhr = new ActiveXObject( "Microsoft.XMLHTTP" ); }
			catch( erreur ){
				xhr = null;
			}
		}
	}
	return xhr;
}
```

Procédé d'opération d'une interaction en AJAX

L'opération de transmission et récupération des informations entre Javascript et un serveur distant s'effectue en 5 étapes.

1. **Chargement des éléments de la page**

   Cette étape préliminaire sert à définir l'événement qui déclenchera le processus en AJAX. Il peut s'enclencher dès la page chargée avec l'association de l'événement `onload()` à l'objet `window`. Il peut également faire l'objet d'un appel à intervalle régulier grâce à la fonction Javascript intégrée `setInterval()`.

   Il peut s'avérer être un choix stratégique de localiser les opérations en AJAX dans une fonction qui sera appelée à tout moment par un événement ou de manière périodique.

```
var ajaxOperation = function()
{
	// Processus AJAX...
}

```

```
6  window.onload = function()
7  {
8        // Appel de la fonction ajaxOperation() dès que la page est chargée.
9        ajaxOperation();
10
11       // Appel toutes les 10 secondes de la fonction ajaxOperation()
12       setInterval( function(){ ajaxOperation(); }, 10000 );
13 }
```

2. **Transmission de la requête au serveur**

   Cette opération consiste à transmettre les données à un fichier sur le serveur distant par la méthode GET ou POST. Ces données pourraient, par exemple, provenir d'un formulaire ou être prédéfinies lors de l'appel du document au serveur distant.

   Cette opération se fait par l'exécution de la méthode `open()` de l'objet `XMHttpRequest`. Ainsi, au préalable sera récupéré l'objet `XMLHttpRequest` initié dans une fonction prévue à cet effet (ici appelé `initXhr()`).

```
1  var ajaxOperation = function()
2  {
3        var doAjax = initXhr();
4        var urlVar = 'action=do&app=ajax';
5        doAjax.open( 'GET', 'ajax.php?' + urlVar, true );
6        doAjax.send();
7  }
```

*Exemple d'envoi par la **méthode** GET*

La méthode POST implique de préciser dans l'entête le type de contenu transmis par l'entremise de la méthode `setRequestHeader()`.

```
1  var ajaxOperation = function()
2  {
3        var doAjax = initXhr();
4        var urlVar = 'action=do&app=ajax';
5        doAjax.open( 'POST', 'ajax.php', true );
6        doAjax.setRequestHeader( 'Content-type', 'application/x-www-form-
   urlencoded' );
7        doAjax.send( urlVar );
8  }
```

*Exemple d'envoi par la **méthode** POST*

3. **Traitement côté serveur**

Le fichier localisé sur le serveur distant peut être de n'importe quel format. Ce peut être une image, comme un fichier HTML ou encore un langage serveur (tel que PHP). Ce qui doit être prévu est qu'un contenu doit être transmis.

Dans le cas de l'utilisation d'un fichier PHP, il sera question de prévoir un affichage du contenu transmis par l'utilisation de la fonction PHP intégrée `echo()` ou `print()`.

```
1  if( $_GET[ 'action' ] === 'do' && $_GET[ 'app' ] === 'ajax' )
2  {
3         echo 'réponse du <strong>serveur</strong>';
4         exit;
5  }
```

Le format JSON est aisé à employer et bien interprété par Javascript. Il peut s'avérer utile de transmettre les contenus sous cette forme à l'aide de la fonction intégrée `json_encode()` qui transforme un tableau de données au format JSON. Cette méthode à l'avantage de permettre le transfert de plusieurs données distincts. Dans l'exemple qui suit sont séparés dans les attributs `'process'` et `'data'` les contenus.

```
1  if( $_GET[ 'action' ] === 'do' && $_GET[ 'app' ] === 'ajax' )
2  {
3         echo json_encode(['process'=>'success', 'data'=>'La <em>réponse</em>']);
4         exit;
5  }
```

4. **Réception de la réponse en provenance du serveur**

Cette étape consiste à vérifier que l'échange a pu se faire en évaluant le processus avec la méthode `onreadystatechange()` dans laquelle seront indiqués l'état du transfert par l'attribut `readyState` et le statut de la réponse (200 : succès, 404 :page introuvable) par l'attribut `status`.

```
1  var ajaxOperation = function()
2  {
3         var doAjax = initXhr();
..         // Procédure de transmission de la requête (étape 2)
8
9         doAjax.onreadystatechange = function(){
10           if( doAjax.readyState == 4 && doAjax.status == 200 )
11           {
12              // Processus de traitement une fois la réponse réussie (étape 5)
```

```
13          }
14       }
15 }
```

5. **Transmission des contenus à l'interface**

   La dernière étape consiste à interpréter les contenus reçus et les transmettre à l'interface. Ces contenus seront disposés soit dans l'attribut `responseXML`, pour les données au format XML (peu utilisé et pas présenté dans les exemples qui précèdent) ou `responseText`.

   L'affichage dans l'interface peut se faire en définissant l'attribut innerHTML qui « écrira » le contenu dans un sélecteur.

```
 1 var ajaxOperation = function()
 2 {
 3       var doAjax = initXhr();
..       // Procédure de transmission de la requête (étape 2)
 8
 9       doAjax.onreadystatechange = function(){
10          if( doAjax.readyState == 4 && doAjax.status == 200 )
11          {
12             document.querySelector( 'span' ).innerHTML = doAjax.responseText;
13          }
14       }
15 }
```

   La récupération de contenus provenant d'un format JSON se fera également par l'intermédiaire de l'attribut `responseText`. La chaîne de caractères pourra être repassée à son format JSON d'origine à l'aide de la méthode `parse()` de l'objet `JSON`.

```
 1 var ajaxOperation = function()
 2 {
 3       var doAjax = initXhr();
..       // Procédure de transmission de la requête (étape 2)
 8
 9       doAjax.onreadystatechange = function(){
10          if( doAjax.readyState == 4 && doAjax.status == 200 )
11          {
12             var respJson = JSON.parse( doAjax.responseText );
13             document.querySelector( 'span' ).innerHTML = respJson.data;
14          }
15       }
16 }
```

Format JSON

Apparu en 2005, JSON (JavaScript Object Notation) est un format d'échange de données considéré très performant. Basé sur un syntaxe similaire à Javascript (ECMAScript), il est plutôt simple à aborder et permet de rendre portable des contenus qui pourront notamment être exploités par d'autres langages et API d'outils tiers.

Composer un JSON

La composition d'un JSON s'apparente à celle d'un objet littéral en Javascript.

```
var jsonFormat = {"attribute":'contenu'}; // Guillemets doubles
```

L'exploitation de son contenu se fera comme ceci :

```
console.log( jsonFormat.attribute ); // Affiche 'contenu'
```

Le format JSON dispose d'une structure qui permet d'y insérer des nombres, chaînes de caractères (entre guillemets), des valeurs booléennes, des tableaux de données (entre crochets) ainsi que des objets (entre accolades). A noter que JSON ne supporte pas les fonctions ni le format des dates (qui seront convertis en caractères).

```
var jsonFormat = {
    "number":100,
    "string":'contenu',
    "boolean":true,
    "array":['contenu', 'contenu'],
    "objet":{'attribut':'contenu', 'methode':function(){
      console.log('contenu');}},
};
```

La composition d'un JSON peut également se faire sous la forme de tableaux de données. Ce qui permet de cumuler plusieurs éléments sous la même forme syntaxique et facilite son exploitation ensuite grâce à une boucle notamment.

```
var jsonFormat = [
    {"user":'John', 'age':32},
    {"user":'Jane', 'age':30},
    {"user":'Joe', 'age':23},
];
```

L'exploitation de ce JSON se fera comme ceci :

```
console.log( jsonFormat[0].user ); // Affiche 'John'
```

Ou à l'aide d'une boucle.

```
[].forEach.call( jsonFormat, function( element ){
    console.log( element.user ); // Affiche le nom de chaque user
});

// ou encore...

var nbElements =  jsonFormat.length;
for( i = 0; i < nbElements; i++ )
{
      console.log( jsonFormat[i].user ); // Affiche le nom de chaque user
}
```

L'objet JSON

L'objet JSON permet de convertir des contenus au format JSON, ce qui facilite leur exploitation dans d'autres langages de programmation notamment.

Pour convertir une chaîne de caractère au format JSON, il est convenu d'utiliser la méthode `parse()`. Ceci sera possible pourvu que la syntaxe de la chaîne de caractères corresponde à celle prévue en JSON.

```
var str =  '{"user":'John'}';
var jsonFormat = JSON.parse( str )
console.log( jsonFormat.user ); // Affiche le nom de chaque user
```

Bien qu'un objet ait l'apparence d'un JSON, il n'en est pas un. Pour convertir un objet Javascript au format JSON, il convient d'utiliser la méthode `stringify()`.

```
var obj = {'user':'John'};
var jsonFormat = JSON.stringify( obj )
console.log( jsonFormat.user ); // Affiche le nom de chaque user
```

jQuery

jQuery est un framework qui a pour but de simplifier le développement d'interfaces Web avec Javascript. Certaines des fonctions de cette librairie remplacent toute une série d'opérations Javascript pour le même résultat. L'autre grande force de jQuery est sa compatibilité entre les navigateurs. L'utilisation de la librairie permet d'introduire sereinement des opérations qui seront supportées de la même manière sur les principaux navigateurs. Alors que l'équivalent en Javascript peut impliquer des ajustements parfois chirurgicaux et que les animations et transitions CSS s'introduisent dans les navigateurs mais de manière inéquitable.

La librairie se télécharge depuis le site jquery.com. La version 2.x de jQuery ne supporte plus les versions 6, 7 et 8 de Internet Explorer. La syntaxe étant la même que la version 1.x, il est inutile d'envisager de revenir en arrière.

Pour l'utiliser, le fichier téléchargé doit être lié au document HTML et doit précéder tout code Javascript faisant référence aux fonctions de la librairie jQuery. Afin de ne pas retarder le chargement de la page HTML, il peut s'avérer utile de poser les liens vers les fichiers externes au bas du document HTML. Comme expliqué à la page 4 du document.

```
..              <!-- Code HTML ... -->
50              <script src="jquery.js"></script>
51              <script src="myscript.js"></script>
52          </body>
53  </html>
```

L'approche de la librairie

jQuery a adopté une logique intuitive et s'adresse aux Webdesigners connaissant bien CSS et cherchant à introduire des animations dans leurs interfaces. jQuery a utilisé le principe d'identification des éléments en reproduisant la syntaxe de CSS. Un principe adopté et de mieux en mieux supporté en Javascript avec les méthodes `querySelector()` et `querySelectorAll()`. Les développeurs y trouvent également leur compte. Car rien n'empêche de combiner Javascript et jQuery tout en profitant d'une syntaxe simplifiée et d'un nombre considérablement réduit de code. En fait, une bonne connaissance de Javascript permettra de profiter pleinement des avantages de jQuery.

Enfin, jQuery propose une série de fonctions d'animations prêtes à l'emploi. Il suffit de préciser la vitesse, le mode d'animation ou encore ce qui se passera une fois l'animation terminée.

Les sélecteurs

Les sélecteurs se définissent selon la syntaxe suivante `$(selecteur)`. Ils servent à identifier un composant ou un/des élément(s) dans la page. Des sélecteurs prédéfinis sont proposés. C'est le cas du sélecteur permettant d'identifier la fenêtre `$(window)` ou le document `$(document)`. Un autre sélecteur défini qui a son équivalent Javascript `this` est `$(this)` qui fait référence à l'élément dans lequel la fonction anonyme est employé. Pour en savoir plus à propos de `this` consulter la page 26 de ce document.

Puis il y a les sélecteurs définis cette fois par la terminologie des sélecteurs en CSS 3. C'est au développeur de définir le sélecteur correspondant aux éléments à identifier dans la page. Ces sélecteurs sont placés entre guillemets.

```
1  $( 'div' )                    // Toutes les balises <div>
2  $( 'span.blue' )              // Toutes les balises <span class="blue">
3  $( 'ul.menu li' )             // Toutes les balises <li> dans <ul class="menu">
4  $( 'dl.def dt:first-child' ) // La première balise <dt> dans <dl class="def">
```

Les événements

Les événements jQuery reproduisent ceux prévus en Javascript et emploient la même procédure d'attribution et de définition des opérations, à savoir qu'un événement se rattache à un sélecteur et les opérations sont introduites dans une fonction anonyme. Cette procédure est décrite en détail à la page 40 de ce document.

```
1  $( selecteur ).event(function(){
2        // operations
3  });
```

La principale différence avec Javascript réside dans la syntaxe. Particulièrement la déclaration de la fonction anonyme contenant les opérations.

```
1  // Javascript
2  window.onload = function(){
3        // operations
4  };
5
6  // jQuery
7  $( window ).load(function(){
8        // operations
9  });
```

Comme le présente l'exemple précédent, le nom des événements n'est pas le même en jQuery. Souvent le « on » placé devant le nom des événements est décliné et ne s'emploie pas de la même façon. L'événement Javascript `onclick()`, par exemple, devient `click()` en jQuery. jQuery dispose toutefois d'une méthode `on()`. Celle-ci permet également d'attribuer un événement à un ou des éléments de la page.

```
$( 'img' ).on( 'click', function(){
      // operations
});
```

Même résultat que :

```
$( 'img' ).click( function(){
      // operations
});
```

Ou encore la méthode `bind()` :

```
$( 'img' ).bind( 'click', function(){
      // operations
});
```

Suppression d'événements

Que ce soit `on()` ou `bind()`, ces méthodes ont leur opposé qui permettent de retirer un événement attribué respectivement avec `off()` et `unbind()`.

```
$( 'img' ).on( 'click', function(){ /* operations */ } ); // Ajout evenement
$( 'img' ).off( 'click' ); // Retrait evenement
```

Événements jQuery spécifiques

jQuery met à disposition des événements à usage spécifiques que Javascript ne dispose pas autrement. C'est le cas par exemple de la méthode one() qui prévoit un seul clic possible sur le ou les élément(s) qui en dispose(nt).

```
$( 'img' ).one( function(){
      // operations ne s'executeront qu'un seule fois.
});
```

jQuery propose également des événements qui disposent de deux états. C'est le cas par exemple de hover() qui regroupe les événements onmouseover() et onmouseout() sous deux fonctions anonymes séparées par une virgule.

```
$('img').hover(function(){
      // operations lors du survol des balises <img>
},function(){
      // operations lors de la fin du survol des balises <img>
});
```

jQuery et le CSS

jQuery propose une façon de faire simplifiée pour disposer dynamiquement des propriétés CSS d'un élément. La méthode CSS() dispose de la capacité de modifier les propriétés ou de récupérer les valeurs des propriétés déjà assignées. L'assignation se fait soit en définissant une seule propriété et sa valeur comme paramètres, soit un groupe de propriétés dans un objet (entre accolades).

```
// Une seule propriété CSS
$('img').css('border-left','2px solid #000');

// Plusieurs propriétés CSS
$('img').css( {'border-left':'2px solid #000', 'margin':'2px'} );

// Possibilité d'utiliser la syntaxe de Javascript pour les propriétés
// mais sans les guillemets
$('img').css( { borderLeft:'2px solid #000', margin:'2px'} );
```

Pour récupérer les valeurs CSS déjà attribuées, la méthode CSS() peut être également employée. Elle renvoie un tableau de données (array) contenant les propriétés demandées ainsi que leurs valeurs.

```
var properties = $('img').css(['border-left', 'margin']);

console.log( properties['margin'] ); // Affiche 2px
```

Traitement des valeurs

La méthode css() offre un autre intérêt particulier qui est de pouvoir ajouter des valeurs numériques aux tailles définies en pixels.

```
var value = 2;

$('img').css(['border-width':(value + 1)]); ; // Affiche 3px
```

Les animations

L'un des intérêts particuliers de jQuery est la facilité avec laquelle les animations peuvent être introduites aux éléments de la page. La méthode `animate()` exprime bien la facilité avec laquelle les animations peuvent être produites. Elle emploie la même syntaxe que la méthode `css()` pour identifier les propriétés CSS qui créeront l'animation. Un second paramètre permet de rectifier la vitesse de l'animation définie en millième de seconde.

```
$('img').animate({'opacity':0.5, 'margin':2}, 500);
```

jQuery propose des méthodes supplémentaires permettant d'attribuer des animations prêtes à l'emploi. C'est le cas par exemple de `slideUp()`, `slideDown()`, `fadeIn()` ou `fadeOut()`.

```
$('ul').slideDown(500);
```

La méthode stop()

Lors du déclenchement successif et à répétition d'une animation par un événement, `hover()` par exemple, jQuery conserve les animations à reproduire et les effectue à la chaîne. Ceci peut donner un résultat non désiré d'une animation qui se répète autant de fois que l'événement a été déclenché. La méthode stop() permet de définir s'il est utile ou pas de la conserver en mémoire et reproduire les événements.

Voici un exemple d'utilisation concrète de la méthode `stop()`. Le survol d'un élément déclenche une animation à répétition. Dès le déclenchement de l'animation suivante, elle annulera la précédente et achèvera l'animation en cours.

```
$(window).load(function(){
	$('ul').hover(function(){
		$(this).stop( true, true ).slideDown();
	},function(){
		$(this).slideUp();
	});
});
```

Les méthodes de sélection

jQuery dispose d'une série de méthodes de sélection qui permettent d'efficacement repérer des éléments dans le DOM.

Ces méthodes s'utilisent lorsqu'un sélecteur a été défini et qu'il est utile de repérer d'autres éléments à partir de celui-ci. Exemple avec la méthode find() qui permet de repérer un ou des éléments contenus dans un élément sélectionné.

```
$('ul').click(function(){
	$(this).find('li.selected').slideDown();
});
```

Voici quelques méthodes de sélection qui méritent d'être connues. Toutes peuvent contenir un sélecteur comme paramètre.

```
$(this).prev();           // L'élément précédent.
$(this).prevAll();        // Tous les éléments précédents.
$(this).prevUntil();      // Éléments précédents jusqu'au sélecteur indiqué.
$(this).next();           // L'élément suivant.
$(this).nextAll();        // Tous les éléments suivants.
$(this).nextUntil();      // Éléments suivants jusqu'au sélecteur indiqué.
$(this).parent();         // L'élément parent direct.
$(this).children();       // Les éléments enfants directs.
```

D'autres méthodes encore complètent cette liste par les possibilités de filtrage qu'elles proposent. C'est le cas de la méthode `not()` qui permet d'éliminer des éléments sélectionnés.

```
$( 'li' ).not( '.green, .blue' ); // Equivalent à $( 'li:not(.green, .blue)' )

$( 'li' ).not( $('li.green') );
```

Une autre méthode pratique dans le filtrage d'éléments est `eq()` qui permet d'identifier un élément en fonction de sa position dans un tableau de données (array).

```
$( 'li' ).eq( 0 );  // Premier élément
$( 'li' ).eq( 2 );  // Troisième élément
$( 'li' ).eq( -1 ); // Dernier élément
```

Enfin l'exploitation unitaire des éléments peut se faire grâce à la méthode `each()` qui effectue une boucle parmi les éléments sélectionnés.

```
$( 'li' ).each( function( index ){  // index indique la clé dans le array
      $(this).css({ marginTop:index });
});
```

Écrire dans le DOM

jQuery dispose également de méthodes qui facilitent l'introduction de contenus et éléments dans la page. Le méthode html() permet d'introduire du contenu. Celui-ci peut contenir des balises HTML.

```
$( 'li' ).html( '<a href="page.html">Lien </a>' );
```

La méthode html() remplace tout contenu existant dans l'élément sélectionné. S'il est plutôt question d'ajouter du contenu à celui existant, les méthodes `append()`, `prepend()` ou `before()` peuvent s'avérer plus adéquats. Elles permettent respectivement d'ajouter à la fin, au début ou avant l'élément sélectionné.

```
$( 'li a' ).append( '<span>Web</span>' );
/* Resultat : <a href="page.html">Lien <span>Web</span></a> */
```

De la même façon il est possible de supprimer du contenu ajouté en faisant référence à un sélecteur avec la méthode `remove()`.

```
$( 'li a' ).remove( '<span>' );
```

jQuery dispose également de la méthode `attr()` utile pour définir ou récupérer les valeurs des attributs.

```
var url = $( 'li a' ).attr( 'href' ); // Récupère la valeur de l'attribut href

$( 'li a' ).attr( 'href', url + '#anchor' ); // Définit une valeur à href
```

Une astuce avec classe

Définir un attribut classe à un élément est très simple en jQuery. Il suffit d'associer à un élément la méthode `addClass('nomClasse')` et indiquer le nom de la classe comme paramètre.

```
$( 'li' ).addClass( 'selected' );
```

Ceci permet d'appliquer les styles se référant au nom de la classe donnée comme cela serait défini dans la feuille de styles CSS, ce pourquoi d'ailleurs cette méthode a été conçue.

Il existe toutefois une autre utilité pratique à l'ajout de classes car il s'agit d'une méthode pour identifier l'état changeant d'un élément au fil des événements. Un exemple connu est l'animation « accordéon » qui ouvre un bloc de contenu en cliquant sur un titre tout en refermant le bloc ouvert d'une autre portion. Sa structure HTML pourrait se présenter comme ceci :

```
<dl>
      <dt>Titre 1</dt><dd>Contenu du premier bloc</dd>
      <dt>Titre 2</dt><dd>Contenu du deuxième bloc</dd>
      <dt>Titre 3</dt><dd>Contenu du troisième bloc</dd>
</dl>
```

Il est utile d'employer des classes dans ce cas pour identifier le bloc qui est ouvert. Cela permet notamment d'éviter que se ferme le bloc déjà ouvert dans le cas où un clic se fait sur le titre qui lui correspond. Ceci est possible en combinant les méthodes `hasClass()` pour vérifier qu'un élément dispose d'une classe et

`removeClass()` qui permet de retirer le nom d'une classe à un élément et ainsi modifier la référence qui lui est associée. Voici l'exemple complet de l'animation de cet « accordéon ».

```
$(window).load(function(){
	$('dd').hide();
	$('dt').click(function(){
		if( !$(this).hasClass('active') ){ // Si l'élément n'a pas la classe
			$('dt.active').removeClass('active');
			$('dd').slideUp();
			$(this).addClass('active');
			$(this).next().slideDown();
		}
	});
});
```

AJAX

Les outils que propose jQuery concernant AJAX sont multiples. Là encore le principe d'échange asynchrone de données, tel que détaillé à la page 43 de ce document, est simplifié.

La fonction `$.ajax()` est d'initier le processus d'échanges de données. Dans sa version courte mais suffisante cette fonction s'utilise comme ceci :

```
$.ajax( 'ajax.php' ).done( function( data ){
	$( 'p.reponse' ).html( data );
});
```

Dans l'exemple, est indiqué comme paramètre le nom du fichier distant à récupérer, puis jointe la méthode `done()` qui permet de préciser ce qui se passera lorsque les contenus seront récupérés. Ce processus est intégré dans une fonction anonyme qui dispose d'une variable (en l'occurrence *`data`*) qui permet de récupérer et transférer les données du serveur à la page.

La fonction `$.ajax()` met à disposition une quantité de paramètres qui permettent d'apporter des précisions notamment quant aux transferts de données, au format et au contenu reçu. Ceux-ci sont transmis dans un objet.

```
$.ajax({
	method:'POST',			// Methode GET ou POST
	url:'ajax.php',			// Fichier distant
	data:{ name:'John', age:32}, // Données transmises au fichier distant
	dataType:'json'			// Format des contenus reçus
```

```
}).done( function( json ){
	$( 'p.reponse' ).html( json.data );
});
```

De la fonction $.ajax() ont été déclinées les fonctions $.get() et $.post(). Il s'agit de fonctions dont la méthode est clairement indiquée.

```
$.get({
	url:'ajax.php',				// Fichier distant
	data:{ name:'John', age:32}, // Données transmises au fichier distant
	dataType:'json'				// Format des contenus reçus
}).done( function( json ){
	$( 'p.reponse' ).html( json.data );
});
```

Chargement de fichiers et de leur contenu

La fonction `load()` de jQuery (il ne s'agit pas de l'événement qui porte le même nom…) permet de gérer le chargement de fichiers et leur contenu de manière asynchrone. Il s'agit d'une méthode utile notamment pour le chargement de contenus devant être partagés par différentes pages.

Dans l'exemple qui suit il est question de récupérer le contenu de la balise `<ul id="menu">` du fichier « menu.html » pour l'introduire dans la balise `<nav>` du fichier courant.

```
$('nav').load( 'menu.html ul#menu' );
```

Récupérer les données d'un formulaire

jQuery propose la méthode `serialize()` pour recueillir automatiquement les données provenant des balises `<input>`, `<textarea>` et `<select>` d'un formulaire. Cette opération aurait autrement dû se faire en recueillant les contenus un à un à l'aide de la méthode `val()`.

```
$.ajax({
	method:'POST',
	url:$('form').attr('action'),
	data:$('form').serialize(),
	success:function( data ){
		$( 'p.reponse' ).html( data );
	}
});
```

Les « callbacks »

Les « callbacks » ou « fonctions de rappel » sont utiles pour différer des opérations dans le temps. Elles ne sont pas propres à jQuery. Il demeure qu'elles sont directement introduites dans les méthodes d'animations et d'AJAX et permettent ainsi de séquencer les opérations.

Les « callbacks » s'introduisent sous la forme d'une fonction anonyme qui contient les opérations à effectuer une fois l'animation ou l'appel en AJAX terminée.

```
1  $('img').animate({'opacity':0.5}, 500, function(){
       $('img').animate({'margin':2}, 1500);
   });
```

Dans l'exemple qui précède, le « callback » est introduit dans la première méthode `animate()` par la fonction anonyme déclarée. Elle fait appel à une seconde animation qui ne se déclenchera qu'une fois la première terminée.

Créer un « plugin »

Il existe une quantité de « plugins » jQuery qui proposent des animations et fonctionnalités prêtes-à-l'emploi, telles que des slideshows, parallax, accordéons,... L'intérêt du « plugin » est de faciliter l'intégration d'animations et de le personnaliser sans avoir à coder à nouveaux les fonctionnalités du « plugin ». Il peut également s'agir d'un moyen pour récupérer des développements qui ont pu être laborieux à mettre en place.

Un « plugin » peut être disposé dans un fichier à part. Ceci facilite sa réutilisation et ne le mélange pas au reste. La déclaration d'un « plugin » se fait de la manière suivante :

```
1  (function( $ ){
2      $.fn.myPluginName = function(){
3          // Opérations
4      };
5  }( jQuery ));
```

`(function( $ ){ /*...*/ }( jQuery ));` définit l'espace réservé au « plugin ». Le symbole `$` est une variable qui fait référence au module. Il permet d'y attacher des méthodes objets spécifiques, c'est ce qui permet notamment d'indiquer un nom au « plugin » avec `$.fn`. Dans l'exemple, `$.fn` permet d'ajouter le nom du « plugin », en l'occurrence « *myPluginName* » a qui est attribué une fonction anonyme qui contiendra les opérations.

L'utilisation du « plugin » se fait par une attribution de la nouvelle méthode à un(des) élément(s) de la page.

```
$('div').myPluginName();
```

Pour intervenir sur le ou les élément(s) associé(s) au « plugin » le mot-clé `this` peut être utilisé. Par ailleurs, il est recommandé que soit retourné `this` afin de préserver la possibilité d'enchaîner les méthodes associées à un élément.

```
(function( $ ){
	$.fn.colorize = function(){
		this.css({color: '#000000'});
		return this;
	};
}( jQuery ));

$('div').colorize().addClass('colorized');
```

Pour donner de la modularité à un « plugin », des paramètres peuvent être transmis lors de l'association du « plugin » à un élement. La fonction $.extend permet d'initialiser des paramètres et définir leurs valeurs par défaut.

```
(function( $ ){
	$.fn.colorize = function( options ){
		var settings = $.extend({
			color: '#000000',
			backgroundColor: '#ffffff'
		}, options );

		return this.css({
			color: settings.color,
			backgroundColor: settings.backgroundColor
		});
	};
}( jQuery ));

$('div').colorize({ color:'#ff0000'});
```