jQuery : écrivez moins pour faire plus !

Par tit_toinou

www.siteduzero.com

Dernière mise à jour le 8/01/2011

Sommaire

jQuery : écrivez moins pour faire plus !

jQuery est une **bibliothèque JavaScript** libre très pratique, ayant une syntaxe courte et logique, compatible avec tous les navigateurs courants. jQuery est devenue une référence importante à savoir utiliser : pour preuve, elle est **largement répandue sur la toile**. Microsoft, Twitter, Mozilla, Google Code, Amazon et Wordpress France, pour ne citer qu'eux, l'utilisent.
Vous ne le savez peut-être pas encore, mais jQuery est même utilisé par le Site du Zéro sur la page que vous visitez en ce moment .

Informations sur le tutoriel

Auteur : tit_toinou
Difficulté :
Temps d'étude estimé : 14 jours
Licence :

Pour pouvoir suivre ce tutoriel sur « **jQuery JavaScript Library** », je vous recommande fortement de :

1. Connaître les bases de l'HTML (les balises) ainsi que du CSS (les sélecteurs) ;
2. mais **surtout de connaître le JavaScript** ;
3. en maîtrisant les événements, le DOM, etc. ;
4. ainsi que ses subtilités telles que les closures ou encore les objets.

Tout le long du tutoriel, je vais mettre à disposition les exemples d'utilisation de jQuery sur un site personnel. Il suffit de cliquer sur le titre de l'exemple ("Essayez ce code !") pour accéder à la page.
Chacune de ces pages donnera un lien vers JSBin et un autre vers jsFiddle. Ce sont deux sites bac à sable, où on peut **tester et éditer son propre code JavaScript**.

Il est important que vous n'hésitiez pas à **modifier le code pour vous entraîner** !

Ce tutoriel se propose donc comme alternative à la documentation officielle, en partant de zéro.
Bonne lecture .

Partie 1 : Bases de jQuery

Dans cette première partie nous verrons jQuery Core, c'est-à-dire le script **jQuery nu, sans aucun plugin tierce**. Nous y étudierons la sélection basique des éléments par la fonction principale, la modification et l'accessibilité du contenu des éléments ainsi que de leurs attributs et de leurs style, et pour finir l'animation des éléments et les événements les plus basiques.

Nous **partirons de zéro** et verrons pas à pas comment utiliser chaque méthode avec **un ou plusieurs exemples**. Le QCM de fin de chapitre vous permettra de vérifier vos connaissances tandis que le TP vous entraînera à les **mettre en pratique**, et ne nécessitera que les connaissances acquises au fur et à mesure du tutoriel.

Découvrir jQuery

Vous connaissez certainement jQuery de nom, mais savez-vous vraiment ce que c'est ?

Présentation

Si vous faites un peu de JavaScript, alors vous savez que c'est parfois très difficile et très long d'obtenir ce que l'on veut, et cela peut paraître répétitif.
Les guerres font rage entre les navigateurs (Firefox, Internet Explorer, Opera, Safari, Chrome etc.) qui **se partagent les internautes** : de ce fait, il faudra adapter votre code à chacun de ces navigateurs qui fournissent **des fonctions différentes** pour répondre au même usage.

Par exemple, ce code permet de récupérer le nombre en pixels de défilement vertical de la page web (la barre de défilement à droite) :

Code : JavaScript

```
function avoirDefilementVertical(){
  if(typeof( window.pageYOffset ) == 'number'){
    // Netscape
    return window.pageYOffset;
  } else if( document.body && (document.body.scrollTop) ||
navigator.appName=="Microsoft Internet Explorer") {
    // DOM
    return document.body.scrollTop;
  } else if( document.documentElement &&
(document.documentElement.scrollTop) ) {
     // Internet Explorer 6
    return document.documentElement.scrollTop;
  }
}
```

C'est ici qu'intervient jQuery, une bibliothèque JavaScript libre, dont **la syntaxe est très courte** et dont les noms des fonctions sont intuitifs (en anglais bien sûr ), simplifiant l'AJAX, permettant de **faire des animations**, ayant une **communauté très active**, et contenant à peu près tout ce dont vous rêvez grâce à ses plugins (nous en verrons quelques-uns plus tard).

Et voici la version jQuery du bout de code :

Code : JavaScript

```
function avoirDefilementVertical(){
  return $(document).scrollTop();
}
```

Simple, non  ?

jQuery n'est pas la seule bibliothèque JS disponible, vous pouvez aussi vous intéresser à d'autres comme Mootools, Dojo, Scriptaculous, Prototype, Yahoo UI, Mochikit, ExtJS, et j'en passe...

Installation

Téléchargement

jQuery est tout simplement **un fichier JavaScript**. Il vous suffit donc de le télécharger sur le site officiel.

Là, deux choix s'offrent à vous dans l'encadré à droite :

- « Production » : version compressée du fichier original (incompréhensible mais légère), si vous ne voulez pas regarder le code source et pour la **version publique de votre site**.
- « Development » : **code source lisible**, vous pouvez réduire le poids de jQuery en enlevant ce qui ne vous sert pas, puis en compressant à l'aide de JavaScript Compressor par exemple.

Dans ce tutoriel, on utilise la version 1.4.3 de jQuery.
J'essaie de mettre à jour le tutoriel à chaque nouvelle version de jQuery. Si vous apercevez une incohérence, n'hésitez pas à me le faire savoir .

Intégration à la page web

On va supposer que votre fichier jQuery s'appelle `jquery.js` et qu'il est dans le même répertoire que le fichier qui l'utilise.
jQuery s'intègre comme tout autre fichier JavaScript :

Code : HTML

```
<script type="text/javascript" src="jquery.js"></script>
```

Voici donc le squelette de base de votre page :

L'encodage est en `iso-8859-1`, si vous êtes sur Linux ou Mac, mettez `utf-8`.

Code : HTML

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
  <head>
    <title>Titre</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
  </head>
  <body>
     <!-- le contenu -->
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
      // c'est ici que l'on va tester jQuery
    </script>
  </body>
</html>
```

Hello World

Pour vous montrer la simplicité de jQuery, voici le classique « Hello World » :

Code : HTML - Essayez ce code !

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
  <head>
    <title>Hello World avec jQuery</title>
    <meta http-equiv="Content-Type" content="text/html; charset=iso-
8859-1" />
  </head>
  <body>
      Salut tout le monde !
    <script type="text/javascript" src="jquery.js"></script>
    <script type="text/javascript">
       $('body').html('Hello World');
    </script>
  </body>
</html>
```

Si « Hello World » s'affiche, tout va bien. Si « Salut tout le monde ! » s'affiche, JavaScript est peut être désactivé, *jquery.js* n'est pas dans le même dossier que le code HTML, ou vous avez fait une erreur en recopiant.
En plaçant la balise `<script>` dans `body`, cela réduit les erreurs et accélère le temps d'affichage de la page.

Vous serez en moyen de comprendre ce premier exemple avec la lecture du troisième sous-chapitre  .

Essayer l'exemple

Comme précisé en introduction du tutoriel, vous pouvez essayer certains exemples en cliquant sur leur titre ("Essayer ce code !").
Notez tout de même que les exemples différent parfois, par exemple avec une balise `<div id="contenu">` qui joue le rôle du conteneur principal de l'exemple.

Préférez les serveurs Google aux vôtres

Le script jQuery

Google héberge lui aussi les fichiers jQuery : http://ajax.googleapis.com/ajax/libs/j [...] jquery.min.js pour la version de Production (compressée).

On peut donc facilement intégrer jQuery à sa page en faisant :

Code : HTML

```
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
```

Cette solution apporte plusieurs avantages :

1. Le script de jQuery est **centralisé** : un internaute visite plein de sites qui sont susceptibles d'intégrer jQuery. Si tous ces sites hébergent leurs propre scripts, l'internaute en question va devoir re-télécharger jQuery plein de fois. Alors qu'avec cette solution, le **cache du navigateur** peut être optimisé et ainsi **éviter de re-télécharger** le même script tout le temps.
2. La **bande passante** de votre site est **allégée**, le travail étant transmis aux serveurs de Google.

Toutes les bibliothèques JavaScript

Une autre solution, un peu plus lourde, est d'intégrer les bibliothèques JavaScript et AJAX de Google via http://www.google.com/jsapi :

Code : HTML

```
<script type="text/javascript"
src="http://www.google.com/jsapi"></script>
```

On peut alors intégrer les bibliothèques JavaScript suivantes : jQuery, jQuery UI, Prototype, script.aculo.us, MooTools, Dojo, SWFObject, Yahoo! User Interface, Ext Core.
Il suffit de faire :

Code : JavaScript

```
// Charge la dernière version de jQuery dans 1.xx.
google.load('jquery','1');
// Charge la dernière version de jQuery dans 1.4x.
google.load('jquery','1.4');
// Charge la version 1.4.1.
google.load('jquery','1.4.1');
```

Ces trois lignes chargent la version 1.4.1 de jQuery.

Cette solution apporte aussi son lot d'avantages :

1. Le script est toujours à la **dernière version** en utilisant la première ligne de code.
2. Une flexibilité (négligeable, me direz-vous) puisque l'on peut **intégrer toutes les bibliothèques** qu'on veut sans se soucier de l'URL du script (plus de **liens morts** !).

Appeler un script externe (en l'occurrence chez Google) n'est pas plus lent qu'appeler un script que vous hébergez.

Ce script a d'autres utilités, comme connaître la position de l'internaute, ou encore traduire un texte (voir la documentation).

Fonction principale

Toute jQuery repose autour d'une fonction : `jQuery()` (abrégée `$()` car le dollar est un caractère autorisé pour les noms de fonctions en JavaScript) qui permettra de sélectionner des éléments dans votre page web.
Le premier argument de cette fonction, l'« expression », qu'on lui passe est une chaîne de caractère, représentant un sélecteur CSS, un peu comme dans vos feuilles de styles, mais en plus fort !

Sélecteurs basiques

Voici quelques exemples pour que vous vous rappeliez de ce que sont les sélecteurs CSS :

Expression	Retour
`#titre`	la balise ayant pour `id` "titre"
`h1`	les balises `h1`
`.gras`	les balises qui ont la classe "gras"
`a,h1,h2`	les balises `a`, `h1` et `h2`
`*`	**toutes** les balises

Le sélecteur de l'`id` ne sélectionne pas plusieurs éléments mais un seul (ou aucun). Les autres sélecteurs peuvent sélectionner 0, 1 ou plusieurs éléments.
Le dernier exemple est à utiliser avec précautions : cela renvoie toutes les balises, même la balise `html` !

Une fois que la fonction retourne ton objet jQuery, on en fait quoi ?

Tout (et n'importe quoi) en utilisant des méthodes. Il y a différentes méthodes, et certaines méthodes ont elles-mêmes différents arguments afin de répondre à différents usages. Leurs noms sont intuitifs (en anglais).

Quelques méthodes

À titre d'exemple, pour passer à la pratique, la méthode `html()` permet d'accéder au contenu des éléments si on l'appelle sans aucun argument, et permet de modifier le contenu des éléments si on l'appelle avec comme premier argument une chaîne de caractères (représentant le contenu).

Pour ce code : `<div id="titre">J'aime les frites.</div>` :

Code : JavaScript - Essayez ce code !

```
$('#titre'); // Sélectionne notre balise mais ne fait rien.
alert($('#titre').html()); // Affiche le contenu "J'aime les
frites."
$('#titre').html('Je mange une pomme'); // Remplace le contenu
("J'aime les frites.") par "Je mange une pomme".
$('#titre').html($('title').html()); // Remplace le contenu par le
titre de la page (contenu dans la balise <title>).
```

Deux autres méthodes, `before()` et `after()` permettent d'ajouter du contenu ou un élément de la page (ça dépend de ce qu'on lui passe en paramètre) respectivement avant et après l'élément en question :

Code : JavaScript - Essayez ce code !

```
// Ajoute du contenu après chaque balise textarea.
$('textarea').after('<p>Veuillez ne pas poster de commentaires
injurieux.</p>');
// Ajoute "Voici le titre :" avant la balise ayant comme id "titre".
$('#titre').before('Voici le titre :');
// Ajoute "! Wahou !" après la balise ayant comme id "titre".
$('#titre').after('! Wahou !');
```

Chaînage des méthodes

Il faut savoir qu'on peut chaîner les méthodes, c'est-à-dire que si une méthode agit sur un élément (elle n'est pas censée donner une information sur cet élément mais le modifier), on peut rajouter une autre méthode :

Code : JavaScript - Essayez ce code !

```
// Ajoute du contenu après chaque balise textarea.
$('textarea')
  .after('<p>Veuillez ne pas poster de commentaires injurieux.</p>')
  .after('<p><strong>Merci beaucoup.</strong></p>');
// Ajoute "Voici le titre :" avant et "! Wahou !" après la balise
ayant comme id "titre".
$('#titre')
  .before('Voici le titre :')
  .after('! Wahou !');
```

Comment cela marche-t-il ?

C'est assez simple : les méthodes qui agissent sur des éléments retournent l'objet jQuery en question, tandis que les méthodes censées donner une information sur ces éléments retournent cette information. Le chaînage s'interrompt donc quand on a affaire au second type de méthode (comme `html()` sans aucun argument).

Chargement du DOM

Quand on fait un appel à la fonction principale, il se peut parfois qu'elle ne retourne rien. On a beau placer son code en fin de `body`, **les éléments de la page web ne sont pas encore placés**.
La solution de ce problème en JavaScript est le fameux :

Code : JavaScript

```
window.onload =  function(){
  // Fonctions du genre document.getElementById('balise') qui
marchent,
  // on peut accéder aux éléments.
};
```

jQuery offre une syntaxe à peu près similaire : la fonction doit juste **être donnée à la fonction principale** `jQuery()` ou `$()`.

Code : JavaScript

```
$(function(){
  // On peut accéder aux éléments.
  // $('#balise') marche.
});
```

Le DOM peut alors être manipulé en toute sûreté car vous pouvez être sûrs que tous les éléments existent. Les feuilles de styles sont aussi chargées si elles ont été incluses avant le script.
On peut utiliser cela plusieurs fois, les fonctions seront appelées chronologiquement à partir du moment où le document est prêt.

Désormais, presque tous les bouts de codes jQuery donnés dans ce tutoriel devront **être entourés** de `$(function(){...});` (sauf pour les fonctions qui se déclenchent avec des boutons par exemple).

Conflits entres différentes bibliothèques JavaScript

Le dollar est un caractère souvent utilisé pour **les noms de fonctions de bibliothèques JavaScript**.
La fonction `$.noConflict()` ou `jQuery.noConflict()` (contenues respectivement dans `$` et dans `jQuery`) permet de supprimer l'alias `$`. **Vous devrez utiliser la fonction `jQuery()`** car `$()` ne sera ainsi plus utilisable.

Cependant, avec des closures, on peut aisément récupérer cette utilisation pratique ; exemple de la documentation jQuery :

Code : JavaScript

```
jQuery.noConflict();
(function($) {
  $(function() {
    // Du code qui utilise $() pour jQuery().
  });
})(jQuery);
// Du code utilisant le $() d'autres bibliothèques JS.
```

Si on passe *true* à cette fonction, elle supprime aussi `jQuery`. Dans tous les cas, cette fonction retourne ce `jQuery()` et vous pouvez ainsi **stocker jQuery dans n'importe quelle variable** :

Code : JavaScript

```
var mon_jQuery_a_moi;
mon_jQuery_a_moi = jQuery.noConflict(true);
```

Sélecteurs

Ce que vous avez vu avant peut aller 5 minutes, mais c'est très vite limité, donc voici la liste des **principaux sélecteurs**, ça devrait vous suffire dans la plupart des cas .

Les sélecteurs CSS sont l'œuvre de Sizzle, créé par l'auteur de jQuery, John Resig.

Sélecteurs CSS "classiques"

Expression	Retour
`*`	Toutes les balises.
`elem`	Les balises `elem`.
`#id`	Balise ayant l'`id` "id".
`.class`	Balises ayant la classe "class".
`elem[attr]`	Balises `elem` dont l'attribut "attr" est spécifié.
`elem[attr="val"]`	Balises `elem` dont l'attribut "attr" est à la valeur `val`.
`elem bal`	Balises `bal` contenues dans une balise `elem`.
`elem > bal`	Balises `bal` directement descendantes de balises `elem`.
`elem + bal`	Balises `bal` immédiatement précédées d'une balise `elem`.
`elem ~ bal`	Balises `bal` précédées d'une balise `elem`.

La liste des sélecteurs CSS 3 est très complète (avec explications en français).

Voici les sélecteurs CSS 3 qui ne sont **pas supportés par jQuery** : `:link`, `:visited`, `:active`, `:hover`, `:focus`, `:target`, `::selection`, `:first-letter`, `:first-line`, `:before`, `:after`, `:nth-last-child()`, `:nth-of-type()`, `:nth-last-of-type()`, `:first-of-type`, `:last-of-type`, `:only-of-type`, `:lang(fr)`.

Sélecteurs spécifiques à jQuery

Expression	Retour
`:hidden`	Éléments invisibles, cachés.
`:visible`	Éléments visibles.
`:parent`	Éléments qui ont des éléments enfants.
`:header`	Balises de titres : `h1`, `h2`, `h3`, `h4`, `h5` et `h6`.
`:not(s)`	Éléments qui ne sont pas sélectionnés par le sélecteur `s`.
`:has(s)`	Éléments qui contiennent des éléments sélectionnés par le sélecteur `s`.
`:contains(t)`	Éléments qui contiennent du texte `t`.
`:empty`	Éléments dont le contenu est vide.

:eq(n) et **:nth(n)**	Le n-ième élément, en partant de zéro.
:gt(n) (greater than, signifiant plus grand que)	Éléments dont le numéro (on dit l'« index ») est plus grand que n.
:lt(n) (less than, signifiant plus petit que)	Éléments dont l'index est plus petit que n.
:first	Le premier élément (équivaut à :eq(0)).
:last	Le dernier élément.
:even (pair)	Éléments dont l'index est pair.
:odd (impair)	Éléments dont l'index est impair.

L'index, qui commence à zéro, est **le numéro d'un élément** parmi les autres éléments contenus dans cet objet jQuery. Le premier élément contenu dans l'objet jQuery est le premier élément trouvé par la fonction principale, son index est 0.

Entraînement

Dans ce bout de code de page web :

Code : HTML

```
1 : <p id="premier_texte">
2 :   <span class="texte">
        Salut tout le monde
      </span>
3 :   <img src="photo_1.jpg" alt="Premiere Photo !"
class="superimage" />
    </p>
4 : <p>
5 :   <img src="photo_2.gif" alt="Seconde Photo !" />
6 :   <span class="texte">
        ma Seconde Photo de Vacances !
      </span>
    </p>
7 : <img src="photo_3.jpg" style="display: none;" class="superimage"
/>
```

On a, selon les différents appels à la fonction $() (les numéros désignent les balises) :

Expression	Numéros des éléments sélectionnés
#premier_texte .texte	2.
p > span	2 et 6.
span + img	3.
span > img	Rien, car les balises img ne sont pas contenues dans les balises span.
p	1 et 4.
img[src$=.jpg]	3 et 7 (pas la 5 car l'attribut src se finit par .gif).
img[src*=hoto]	3, 5 et 7 (car ils contiennent tous hoto (photo_ est en commun) dans leur attribut src).
img:visible	3 et 5.
p ~ img	7.

`p:first + img`	Rien, car aucune balise `img` ne suit directement la première balise `p`.
`:hidden`	7, car dans son attribut `style`, `display` est à `none`.
`img:hidden:not(.superimage)`	Rien, car la seule image non-visible a la classe `superimage`
`p:contains('Salut'):has(span)`	1, car contient "Salut" (dans le `span`) et une balise `span`.
`:not(html):not(body):even:not(img)`	2, 4 et 6 (les `:not(html)` et `:not(body)` évitent de récupérer ces balises, `:even` sélectionne les numéros pairs et le `:not(img)` ne change rien).

Variables pratiques

Expression

La variable `selector` d'un objet jQuery est **l'expression utilisée**, c'est-à-dire la chaîne de caractères qui représente le sélecteur passé en paramètre à la fonction `$()`.
Son utilisation est donc `$('p > span').selector` par exemple et qui renvoie "p > span".

Nombre d'éléments

La variable `length` (longueur) d'un objet jQuery est **le nombre d'éléments** contenus dans cet objet jQuery, c'est-à-dire le nombre d'éléments trouvés par l'appel à la fonction `$()`.
Son utilisation est donc `$('img[src~=hoto]').length` par exemple et qui renvoie dans le bout de code plus haut 3.

On peut alors détecter si un élément existe dans la page web :

Code : JavaScript

```
if($('span').length > 0){
  // Il y a un ou plusieurs span dans le document.
}
  // Ou plus simplement :
if($('span').length){
  // Il y a un ou plusieurs span dans le document.
}
```

Note : on peut aussi utiliser la méthode `size()` qui retourne **exactement la même chose**.

Code : JavaScript

```
if($('span').size()){
  // Il y a un ou plusieurs span dans le document.
}
```

Voici les points importants de ce chapitre à retenir :

1. Il faut **absolument connaître le JavaScript** afin de maîtriser jQuery ;
2. jQuery est une bibliothèque JS : elle n'est **pas incluse nativement** (par défaut) dans le JavaScript ;
3. Il faut inclure jQuery soi-même **avant de pouvoir l'utiliser** ;

4. un objet jQuery n'est **pas une référence à un élément du document**, contrairement à ce que retourne `document.getElementById('id')` par exemple ;
5. un objet jQuery **contient une ou plusieurs référence(s)** à un (des) élément(s) du document (tout cela est mieux expliqué au début du troisième chapitre) ;
6. et enfin, il faut toujours entourer son code par le fameux `$(function(){ /* Ici votre code JS */ });` afin que **le document soit chargé**.

Appris dans ce chapitre : fonction principale, intégrer jQuery à sa page web, chaînage des méthodes, chargement du document, sélecteurs CSS et sélecteurs spécifiques à jQuery, variables `selector` et `length`, quelques méthodes : `noConflict()`, `size()`, `html()`, `before()`, `after()`.

Manipuler le contenu

Ce chapitre vous présentera les différentes méthodes qui permettent de manipuler le contenu de votre page. Que ce soit du plus simple, à ce qui aurait pris des dizaines de lignes en JavaScript.

Ainsi, vous serez capables de **récupérer le contenu textuel** d'un élément, de **modifier son code HTML**, de lui **ajouter** un autre élément avant ou après, de lui **incorporer** un autre élément, de l'**envelopper** d'un autre élément, ainsi que de le **copier**, de le **supprimer** ou de le **vider**.

Contenu textuel

Méthode `text()`

Vous connaissez déjà `html()`, qui permet de remplacer le contenu d'un élément. Eh bien voici `text()`, qui manipule le contenu comme du texte et non comme des balises :

- changer le contenu avec `text('contenu')` : les chevrons < et > sont **convertis en entités HTML** : `<` et `>` respectivement (**les balises ne sont donc plus actives**) ;
- récupérer le contenu avec `text()` : renvoie le **contenu « textuel »** (pas les balises), les entités HTML étant **converties en caractères**.

C'est l'équivalent en JavaScript classique de la propriété `textContent`.

Comparaison de `text()` avec `html()`

Récupérer le contenu

Code : HTML

```
<p id="premier">
  <span class="texte">
    Salut tout le monde
  </span>
  <img src="photo_1.jpg" alt="Premiere Photo !" />
</p>
```

`$('#premier').text()` renvoie "Salut tout le monde" (avec tous les retours à la ligne et les espaces).

Alors que `$('#premier').html()` renvoie :

Code : HTML

```
<span class="texte">
    Salut tout le monde
  </span>
  <img src="photo_1.jpg" alt="Premiere Photo !" />
```

Ces méthodes ne récupèrent que **le contenu du premier élément** trouvé par jQuery (dont l'index est 0), exactement comme `html()`.

Définir le contenu

Et lorsque l'on passe du texte en paramètre : `$('#premier').text('<strong>les pommes</strong>')` change le contenu du paragraphe en

Code : HTML - Essayez ce code !

```
&lt;strong&gt;les pommes&lt;/strong&gt;
```

Alors que `$('#premier').html('<strong>les pommes</strong>')` ajoute une balise `<strong>les pommes</strong>` .

Cliquez en haut sur "Essayez ce code !" pour voir les différences entre `html()` et `text()`.

Cas spécial : titre du document

On ne peut pas changer le titre du document avec `html()` ou `text()`, parce que utiliser ces méthodes changera le contenu HTML de la balise `title` mais en aucun cas ne changera le titre tel qu'on le voit en haut de votre onglet ou fenêtre.

Il faut donc utiliser `document.title = chaineDeCaractere;`.
Par contre on peut récupérer le titre du document : `$('title').text()`.

Plusieurs emplois de `text()`

Faites attention avec le double emploi de `text()` : affectation puis récupération.

Cela renvoie du code HTML avec **les balises actives**. Par exemple :

Code : JavaScript

```
$('balise1').html(
  $('balise2')
    .text('<strong>les pommes</strong>')
    .text()
);
```

`balise1` contient alors en gras "les pommes".

La méthode `text()`, pour définir le contenu d'un élément, **vous garantit qu'il n'y aura pas de balises actives** dans cet élément.
Maintenant dans ce cas, on a utilisé `html()` pour définir le contenu.

Remplacer la balise

Première méthode

Les méthodes `html()` et `text()` permettent de changer le **contenu** : `replaceWith()` est une méthode qui permet de remplacer **la balise et son contenu**. On peut lui passer du code html ou encore un objet jQuery qui viendra **remplacer** l'élément en question.

Ainsi faire `$('div').replaceWith('<span>Salut !</span>')` transformera `<div>Bonsoir.. :)</div>` en `<span>Salut !</span>` .

Code : JavaScript - Essayez ce code avec replaceAll() !

```
// Remplace les liens <a>...</a> par <em>censuré</em>.
$('a').replaceWith('<em>censuré</em>');
$('h1').replaceWith($('.titre:first'));
$('#titre').replaceWith('<h1>'+$('#titre').html()+'</h1>');
$('.recherche').replaceWith('<a
href="http://google.com">Google</a>');
```

Seconde méthode

La méthode `replaceAll()` permet de faire l'opération inverse. On peut lui passer un objet jQuery mais encore une expression (comme ce qu'on met dans la fonction principale).

Soit A et B deux objets jQuery.

```
A.replaceWith(B); // B va remplacer A.
A.replaceAll(B): // A va remplacer B.
```

Code : JavaScript

```
// Tous les <h1> vont être remplacés.
$('#titre').replaceAll($('h1'));
// Revient à faire :
$('#titre').replaceAll('h1');
```

Si on donne une chaîne à `replaceAll()`, cela représente une expression (ou un sélecteur jQuery, au choix), tandis que si on donne une chaîne à `replaceWith()`, cela représente du code html, donc faites attention.

Intérieur et extérieur

Insérer à l'intérieur

Les méthodes `html()` et `text()` réécrivent le contenu d'un élément : c'est-à-dire qu'elles les vident dans un premier temps, puis y écrivent.

Les méthodes `prepend()` et `append()` permettent d'ajouter du contenu à celui existant : `prepend()` avant et `append()` après.
`$('span').prepend('balise span » ')` transformera `<span>Hello World !</span>` en `<span>balise span » Hello World !</span>` .
`$('span').append(' « balise span')` transformera `<span>Hello World !</span>` en `<span>Hello World ! « balise span</span>` .

On peut aussi passer un objet jQuery (tous les éléments contenus dans ce dernier seront insérés) en paramètre :

Code : JavaScript - Essayez ce code !

```
$('a').prepend('Lien : ');
$('h1:first').prepend('Premier titre : ');
$('q').append(" (c'était une citation)");
$('#titre').append($('#sommaire'));
$('#basdepage').prepend($('h1'));
```

Tout simplement !

Il existe aussi deux autres méthodes, `prependTo()` / `appendTo()`, qui permettent l'opération inverse :

Soient A et B deux objets jQuery.

```
A.prepend(B); A.append(B); // B va se placer avant / après A.
A.prependTo(B); A.appendTo(B); // A va se placer avant / après B.
```

Le premier argument de `prependTo()` / `appendTo()` peut être soit une chaîne de caractères, représentant un sélecteur, soit un objet jQuery (retourné par `$()`).

Pour ce code : `<span id="nonosse">Nonosse...</span> <div id="chien">Chien !!!</div>` .

Code : JavaScript - Essayez ce code !

```
$('#nonosse').appendTo('#chien');
// Revient à faire :
$('#nonosse').appendTo($('#chien'));
// Ou alors :
$('#chien').append($('#nonosse'));
// Ou alors directement en prenant le texte.
$('#chien').append('Nonosse...');
```

Mais on peut aussi faire plus en créant des balises :

Code : JavaScript - Essayez ce code !

```
$('#contenu p.message_forum').prepend('<em>message du forum :
</em>');
$('#contenu a').append('<span title="avertir">avertissez les liens
morts !</span>');
$('#contenu').append('<div id="basdepage">Tout contenu est sous CC
BY.</em>');
```

Même remarque : attention à la chaîne donnée à `prependTo()` et `appendTo()` qui est un sélecteur et à celle donnée à `append()` et `prepend()` qui est du html.

Insérer à l'extérieur

Les méthodes `append()` / `prepend()` / `appendTo()` / `prependTo()` changent le contenu à l'intérieur, c'est à dire que faire `$('p').prepend('Paragraphe :<br />')` dans `<p>Comment tu vas ?</p>` , change le paragraphe en : `<p>Paragraphe :<br />Comment tu vas ?</p>` .

Les méthodes `after()` / `before()` / `insertAfter()` / `insertBefore()` sont les équivalents de ces méthodes, à l'exception qu'elles ajoutent à l'extérieur des balises. Elles fonctionnent donc de la même manière : on passe en argument un objet jQuery ou du contenu html à `after()` / `before()` et un sélecteur jQuery ou un objet jQuery à `insertAfter()` / `insertBefore()`.

Donc si on fait `$('p').before('Paragraphe : ')` dans `<p>Comment tu vas ?</p>` , cela deviendra `Paragraphe : <p>Comment tu vas ?</p>` .

Code : JavaScript - Essayez ce code !

```
// Tout ce qui suit revient au même :
$('#titre').insertBefore('h1:first');
$('#titre').insertBefore($('h1:first'));
$('h1:first').insertAfter($('#titre'));
$('h1:first').insertAfter('#titre');
$('h1:first').before($('#titre'));
$('#titre').after($('h1:first'));
```

Les méthodes `appendTo()` / `prependTo()` / `insertAfter()` / `insertBefore()` renvoient les objets une fois placés dans le document.

Envelopper

Les méthodes ci-dessous prendront en argument soit un objet jQuery, soit du contenu HTML.

Envelopper à l'extérieur

`wrap()` permet d'« envelopper » n'importe quel élément entre deux balises. Le plus simple est de lui passer une balise ouvrante et une fermante sans contenu (`<em></em>` par exemple).
`$('span').wrap('<b></b>')` transformera **`<span>`**`Hello World !`**`</span>`** en **`<b><span>`**`Hello World !`**`</span></b>`**

Mais on peut aussi rajouter des choses autour tant que la chaine passée commence par une balise ouvrante et finit par une balise fermante.
Cela peut paraître logique, mais on ne peut pas envelopper avec une balise « seule » (par exemple `<img />`, `<input />`, etc.).
Si il y a plusieurs paires de balises ouvrantes / fermantes sans contenu, le contenu sera dupliqué dans chacune d'entre elles.

Code : JavaScript - Essayez ce code !

```
$('#titre').wrap('<h1></h1>');
$('.desactiver').wrap('<pre></pre>');
// Le contenu sera entre les <div></div>.
$('.contenu').wrap('<em>le contenu</em> ira là : <div></div> mais
pas <strong>là</strong>');
// Le contenu sera entre les <em></em> mais aussi les <q></q>.
$('span').wrap('<em id="ici"></em> et là <q></q>');
```

Envelopper à l'intérieur

`wrapInner()` fontionne de la même manière mais entoure le contenu. Tout comme avec `wrap()`, le contenu est dupliqué dans chacun des couples de balises ouvrantes / fermantes sans contenu.
`$('span').wrapInner('<b></b>')` transformera **`<span>`**`Hello World !`**`</span>`** en **`<span><b>`**`Hello World !`**`</b></span>`**

Code : JavaScript - Essayez ce code !

```
$('span.italique').wrapInner('<i></i>');
$('h1:first').wrapInner('<a href="http://google.com"></a>');
$('a').wrapInner('<em>Ceci est un lien : </em><u></u>');
```

Tout envelopper

`wrapAll()` est similaire à `wrap()` à l'exception que si ce dernier enveloppe chaque élément un par un, `wrapAll()` enveloppe tous les éléments d'un coup.

Et si les éléments ne se trouvent pas côte à côte ?

La méthode va tout simplement placer tous les éléments à la suite les uns des autres.

Code : JavaScript - Essayez ce code !

```
$('input.premierFormulaire').wrapAll('<form></form>');
```

```
$('q').wrapAll('<div class="groupeDeCitations"></div>');
```

Déballer

`unwrap()` est une méthode permettant de « désenvelopper » (ou déballer) un élément. C'est-à-dire **enlever la balise parent qui l'enveloppe**.
C'est un peu la méthode inverse de `wrap()` !

Par exemple `$('b > span').unwrap();` sur `<p>Hello <b>Wor<span>ld</span></b></p>` donne `<p>Hello Wor<span>ld</span></p>` .

A la base un plugin pour jQuery, elle a été finalement intégrée dans la version 1.4.

Copier, supprimer et vider

Copier des éléments

`clone()` permet de copier des éléments (de les « cloner »), avec leur contenu, leurs attributs et valeurs (donc par conséquent les événements comme `onclick` etc.).
`clone()` va simplement renvoyer un objet jQuery qui est une copie de l'original, mais il ne va pas l'ajouter à votre page web.

Si on l'appelle en lui passant *true*, la méthode clone aussi les événements associés à ces éléments.
Vous verrez plus tard comment créer des événements avec jQuery.

Code : JavaScript - Essayez ce code !

```
// Multiplie le nombre de boutons par 2.
$('button').clone().appendTo($('body'));
// Revient à faire :
$('body').append($('button').clone());
```

Supprimer des éléments

`remove()` permet de supprimer des éléments de la page web. On peut lui passer un argument qui représente un sélecteur jQuery : il désignera les éléments qui doivent être supprimés parmi tous les éléments contenus dans l'objet jQuery.

Il ne sert à rien d'appeler `remove()` avant d'appeler des méthodes qui insèrent l'élément quelque part dans le document (`prependTo()`, `appendTo()`, `insertBefore()` et `insertAfter()`), car ces méthodes vont supprimer l'élément avant de l'insérer.

Code : JavaScript - Essayez ce code !

```
// Supprime les liens de la page ayant la classe "sites".
$('a').remove('.sites');
// Supprime les balises <strong> dans des <button>.
$('button strong').remove();
```

Vider des éléments

`empty()` permet de vider le contenu de l'élément en question :

Code : JavaScript - Essayez ce code !

```
$('button').empty(); // Vide les boutons.
$('body').empty(); // Vide la page web.
// Revient à faire :
$('body').html('');
```

TP : Condensé

Cahier des charges

Le but sera de manipuler du contenu () en partant de ce bout de page HTML :

Code : HTML

```
<p id="contenu">
  Lorem <span class="vert">ipsum dolor</span> sit amet, <span
class="titre2">
  consectetur adipiscing elit</span>.Etiam <a href="#">facilisis</a>
  <span class="rouge">ultrices</span> dolor, eu <span
class="orange">fermentum
   eros</span> aliquet ac. Aenean varius <span
class="titre2">ultrices nisi
  </span> vel aliquet. Nam eu viverra sem. <span class="gras">Fusce
facilisis
  </span> eros ac <span class="titre1">elit scelerisque
molestie</span>. Morbi
   lacus orci, interdum ac <span class="souligne">faucibus
hendrerit</span>,
   <span class="barre">facilisis vel</span> nunc. <span
class="souligne">Sed in
  </span> <span class="bleu">mauris</span> <span
class="gras">lorem</span>.
  Integer facilisis, <span class="italique">augue et suscipit</span>
molestie,
   <span class="barre">lectus lectus</span> pellentesque mi, <span
class="vert">
  at</span> condimentum <span class="italique">nulla</span> nibh ut
turpis. <span>
  Cum sociis</span> natoque <span class="vert">penatibus et
magnis</span> dis
  <a href="#">parturient montes</a>, nascetur ridiculus mus. Etiam
quis nisl
  metus.<span class="vert">Phasellus</span>ullamcorper posuere augue
quis placerat.
  <span class="titre1">Duis sed quam</span>odio. Donec <span
class="vert">aliquam
   metus</span> a <a href="#">ligula lacinia</a> a tempor leo <span
class="bleu">imperdiet</span>.
   Cras augue purus, <span class="souligne">lobortis eu</span>
scelerisque sed,
   <span class="vert">venenatis ut</span> turpis. Donec <span
class="bleu">quis
   magna sapien</span>. Ut ut diam arcu. <span
class="souligne">Suspendisse nec
   risus</span> id lacus venenatis <a href="#">rhoncus.</a> In vitae
  <span class="vert">justo tellus</span>, <span class="bleu">vitae
lacinia nunc
  </span>. Aliquam <span class="italique">erat</span> <span
class="rouge">volutpat.</span>
</p>
```

Un paragraphe repéré par un `id` est rempli de texte avec certaines parties entourées de `<span>` avec des classes différentes (`vert`, `titre1`, `titre2`, `rouge`, `italique`, `gras`, `souligne`). Il y a aussi des balises liens `<a>` .

Puis l'utilisateur va pouvoir agir sur la page grâce à des boutons : `<button onclick="fonctionJs()">Cliquez ici</button>` .

Vous pouvez attribuer les fonctionnalités que vous voulez aux boutons, mais en voici quelques-unes que je vous propose :

- enlever les liens ;
- enlever le texte en gras ;
- enlever le texte en italique ;
- enlever le texte décoré ;
- vider les boutons ;
- voir le code ;
- transformer les liens en boutons ;
- dupliquer le texte ;
- regrouper les liens ;
- mettre des titres ;
- regrouper les titres ;
- colorer le texte ;
- organiser sémantiquement le texte ;

Essayez d'en faire un maximum et de faire vos propres fonctions, vous pouvez aussi ajouter du contenu à la page, mais aussi prendre le code d'une page d'un site connu, pour ensuite "jouer" avec.

Aide

Si vous ne voyez pas du tout par où commencer, je vous conseille tout d'abord d'écrire la page HTML (sans code JavaScript) en ajoutant les boutons, et le code de base.
Puis ajoutez les événements `onclick` à chaque bouton qui appelle une fonction JavaScript.
Puis écrivez une à une, tout doucement, les fonctions JavaScript.

Solution

N'oubliez pas que :

Citation

> C'est en s'entraînant que l'on apprend.

Voici une solution possible : ce n'est pas parce que votre code est différent qu'il est faux, ni moins bien.

Tout d'abord, voici le code HTML sans le code cité précédemment :

Code : HTML

```
<button onclick="semantique()">Organiser sémantiquement le
texte</button>
<button onclick="colorer()">Colorer le texte</button>
<button onclick="mettreTitres()">Mettre des titres</button>
<button onclick="liensEnBoutons()">Transformer les liens en
boutons</button>
<button onclick="dupliquerTexte()">Dupliquer le texte</button>
<button onclick="regrouperTitres()">Regrouper les titres</button>
<button onclick="regrouperLiens()">Regrouper les liens</button>
<button onclick="viderBoutons()">Vider les boutons</button>
<button onclick="enleverLiens()">Enlever les liens</button>
<button onclick="enleverGras()">Enlever le texte en gras</button>
<button onclick="enleverItalique()">Enlever le texte en
italique</button>
<button onclick="enleverDecor()">Enlever le texte décoré</button>
<button onclick="voirCode()">Voir le code</button>
```

Code : JavaScript - Essayez ce code !

```
function enleverLiens() {
```

```
  $('#contenu a').remove();
}

function enleverGras() {
  $('#contenu strong,#contenu b').remove();
}

function enleverItalique() {
  $('#contenu em,#contenu i').remove();
}

function enleverDecor() {
  $('#contenu *:not(html):not(body):not(p):not(button)').remove();
}

function viderBoutons() {
  $('#contenu button').empty();
}

function voirCode() {
  $('#contenu p').text($('p').html());
}

function liensEnBoutons() {
  $('#contenu a').wrap('<button></button>');
}

function dupliquerTexte() {
  $('#contenu p').clone().appendTo('#contenu');
}

function regrouperLiens() {
  $('#contenu a').wrapAll('<div></div>');
}

function mettreTitres() {
  $('#contenu .titre1').wrap('<h1></h1>');
  $('#contenu .titre2').wrap('<h2></h2>');
}

function regrouperTitres() {
  $('h1').wrapAll('<div></div>');
  $('h2').wrapAll('<div></div>');
}

function colorer() {
  $('#contenu .rouge').wrap('<span style="color:red"></span>');
  $('#contenu .vert').wrap('<span style="color:green"></span>');
  $('#contenu .orange').wrap('<span style="color:orange"></span>');
  $('#contenu .bleu').wrap('<span style="color:blue"></span>');
}

function semantique() {
  $('#contenu .italique').wrap('<i></i>');
  $('#contenu .gras').wrap('<b></b>');
  $('#contenu .souligne').wrap('<u></u>');
  $('#contenu .barre').wrap('<del></del>');
}
```

Voilà, relisez bien pour voir ce que vous n'avez pas compris, mais normalement c'était juste pour vous faire retenir les fonctions de ce chapitre .

Pour les plus feignants d'entre vous, vous pouvez tester le code.

Note

Vous verrez plus tard comment créer des événements avec jQuery. Pour l'instant, vous ferez comme dans ce TP, avec des boutons et des attributs `onclick`.

Maintenant vous n'avez plus d'excuse pour ne pas **optimiser le contenu** de votre page.

Appris dans ce chapitre : comparaison `html()` / `text()`, récupérer le titre du document, méthodes : `replaceWith()`, `replaceAll()`, `prepend()`, `append()`, `prependTo()`, `appendTo()`, `after()`, `before()`, `insertArter()`, `insertBefore()`, `wrap()`, `wrapInner()`, `wrapAll()`, `clone()`, `remove()`, `empty()`.

Modifier la structure

La création d'éléments, la manipulation de leurs attributs, les formulaires ainsi que quelques méthodes bien utiles qui permettent de boucler sur des éléments seront expliqués ici. Vous comprendrez ainsi en quoi la **manipulation du** DOM est d'une simplicité remarquable avec jQuery.

La lecture de la première sous-partie est fortement recommandée : elle introduit quelques notions afin de comprendre **ce qui se passe dans jQuery** ainsi que la structure du document. La seconde sous-partie expliquera comment **passer une fonction en argument** d'une méthode.

C'est dans les sous-chapitres suivants que nous rentrerons au cœur du sujet.

---

jQuery et le document

DOM

La page web **contient des éléments** (des balises), qui elles-mêmes sont susceptibles de contenir d'autres éléments.
Cette **organisation** de la page web forme un « arbre » où un élément parent **contient des fils**, qui eux-mêmes sont susceptibles d'être parents.

Le DOM est un ensemble d'interfaces standardisées, décrivant cet « arbre ». Concrètement, on les utilise en JavaScript grâce à l'objet `document`, qui grâce à ses nombreuses méthodes (`getElementById()`, `createElement()`, que vous connaissez par exemple) nous permettent de récupérer des éléments, et ainsi les modifier grâce à leurs propriétés et à leurs méthodes (`setAttribute()`, `innerHTML`, `addEventListener`, etc...).

Objet jQuery

La fonction principale renvoie un **objet jQuery**. Un objet jQuery contient tout un tas d'éléments de la page web, sur lesquels on peut **appliquer des méthodes** pratiques, qu'on va **apprendre au fur et à mesure** du tutoriel.
Ainsi `$('body')` retourne un objet jQuery contenant un seul élément, `$('a')` retourne un objet jQuery pouvant contenir 0, 1 ou plusieurs éléments.

On peut accéder au n-ième élément contenu dans un objet jQuery en écrivant `[n]` à la suite de cet objet. `[0]`, `[1]` et `[28]` désignent alors respectivement le premier, le second et le vingt-neuvième élément.

`html()` et `after()` s'utilisent bien sûr avec un objet jQuery ; `html()` ne renvoie le contenu que du **premier élément** (`[0]`) de tous les éléments contenus dans l'objet jQuery.

Élément du DOM

`$('body')[0]` **désigne le premier élément** de l'objet jQuery résultant de l'appel à la fonction principale, c'est un élément du DOM.
Cette fois, les méthodes ne sont pas applicables ! Ce code ne fonctionne donc pas : `$('p')[0].html('salut !')` .

Nom de la balise

Les méthodes peuvent être appelées pour un objet jQuery. À l'inverse, des **variables** comme `tagName` ne peuvent être récupérées qu'avec un élément du DOM (**nous ne sommes alors plus dans le domaine de jQuery**, mais du JavaScript en général), donc `$('body').tagName` ne marche pas !

`tagName` renvoie le nom de la balise de l'élément en majuscule, c'est une **chaîne de caractères**.
On peut utiliser `toLowerCase()` sur un objet `String`, donc on peut logiquement utiliser `tagName.toLowerCase()` qui renvoie le nom de la balise de l'élément en minuscule.

Code : JavaScript - Essayez ce code !

```
// Renvoie 'body'.
$('body')[0].tagName.toLowerCase();
```

Mais que contient réellement cet objet jQuery ?

On a l'impression qu'un objet jQuery est un tableau d'éléments. Or, cela n'est pas vrai : comme vous pouvez le constater, un objet jQuery contient la variable `selector`.

Mais alors comment se fait-il que je puisse accéder au premier élément en faisant `[0]` ?

C'est simple, voyez par vous-même dans ce code :

Code : JavaScript

```
// Déclaration d'un objet.
var objet = {};

// Déclaration d'attributs.
objet.variable1 = 'Hello';
objet['variable2'] = 'World';
// Affiche 'Hello World'.
alert(objet.variable1+' '+objet.variable2);

// On ne peut pas utiliser le point
// car une variable ne commence
// jamais par un chiffre.
objet[0] = 'premier élément';
// [0] ou ['0'] reviennent au même.
objet['1'] = document.getElementById('titre');
// Affiche 'premier élément'.
alert(objet[0]);
// Affiche 'titre' si la balise existe.
alert(objet[1].id);

// Ce sont des variables comme les autres
// donc je mets ce que je veux !
objet.length = 'Moi aimer frites !';
objet.selector = 89;
```

Les éléments (`[0]`, `[1]` etc.) contenus dans un objet jQuery sont donc des attributs de cet objet.
On ne peut pas utiliser le point pour les désigner car, comme dit plus haut, une variable ne commence jamais par un chiffre.

Transformer un objet jQuery en tableau

Une méthode pratique, `get()` (ou `toArray()`), permet de **transformer un objet jQuery en tableau**.
Cette méthode, appelée sans rien, renvoie un tableau contenant ce que contient l'objet jQuery, mais sans `selector`. Un vrai tableau.

Une fois que l'objet jQuery est transformé en tableau :

- `length` ne sera plus une variable décidée par jQuery mais une **vraie variable relative aux tableaux** ;
- on peut alors appliquer les méthodes pratiques des objets `Array` (**`join()`, `slice()`** etc. cf. cours JavaScript) ;
- on peut alors **manipuler les éléments normalement** (cf. cours JavaScript).

Parents et enfants

- `parents()` est une méthode qui renvoie l'ensemble des **éléments parents** des éléments en question. Cette méthode peut renvoyer plusieurs éléments par élément concerné par l'appel à cette dernière.
- `parent()` est une méthode qui renvoie **l'élément parent direct** des éléments en question. Cette méthode ne peut renvoyer qu'un seul élément par élément concerné par l'appel à cette dernière.
- `children()` est une méthode qui renvoie **les éléments enfants directs** des éléments en question, c'est-à-dire qu'elle ne renverra pas les enfants des enfants. Cette méthode peut renvoyer plusieurs éléments par élément concerné par l'appel à cette dernière.

Ces trois méthodes peuvent prendre en argument **une expression qui filtrera les éléments** : ceux qui ne conviennent pas au sélecteur seront supprimés des éléments retournés (les méthodes peuvent ne rien retourner !).

Code : JavaScript - Essayez ce code !

```
// Affiche 'HTML'.
alert($('body').parent()[0].tagName);
// Affiche 'HEAD'.
alert($('title').parents()[0].tagName);
```

Passer une fonction aux méthodes

Fonctions anonymes

Désormais, la plupart des méthodes jQuery peuvent prendre des **fonctions anonymes** en argument.

Une fonction anonyme est une fonction qu'on crée « sur le tas ». Elle **sera appelée par la méthode** qui en aura besoin une ou plusieurs fois, les arguments qu'elle prend dépendront donc de l'utilisation que la méthode en fait (tout cela sera précisé).

Code : JavaScript

```
$(expression).methode(function(){
  // Action
});
```

Bien sûr, au lieu de donner en paramètre une fonction anonyme, on peut aussi donner en paramètre une référence à une fonction :

Code : JavaScript

```
function maFonction(){
  // Action
};
$(expression).methode(maFonction);
```

La première utilisation est tout de même beaucoup plus répandue.

Le mot-clé `this` est disponible dans ces fonctions, représentant **un élément du DOM**.

`this` et les fonctions des méthodes

Une autre façon d'appeler la fonction principale est de donner un élément du DOM. La fonction principale **le transforme alors en objet jQuery**, les méthodes sont maintenant applicables.
Par exemple `$(document)` ou encore `$(document.body)` !

La variable `length` (et donc ce que renvoie la méthode `size()`) est alors égale à 1. La variable `selector` est une chaîne de caractères vide.

Donc, dans une fonction passée à une méthode, **il faut utiliser `$(this)`** afin d'utiliser les méthodes jQuery sur les éléments qu'on modifie grâce à la méthode.

Les deux types d'utilisation des méthodes

Nous avons à peine commencé à apprendre quelques méthodes jQuery qu'on s'aperçoit déjà que **certaines méthodes s'utilisent de deux façons différentes**.
Par exemple, pour `html()`, on peut récupérer le contenu, ou le définir.

1. Une méthode qui **définit un paramètre** est appelée « setter »
2. Une méthode qui **récupère un paramètre** est appelée « getter »

La plupart des méthodes ont des utilisations qui la rendent getter et d'autres utilisations qui la rendent setter.

Pour `html()` : l'utilisation `$('#titre').html('Ceci est le titre de la page !');` est setter alors que l'utilisation `alert( $('#titre').html() );` est getter !
Rien de bien compliquer, ce vocabulaire vous sera utile lorsque vous lirez la documentation en anglais  .

Les fonctions en argument dans les méthodes déjà apprises

Une des nouveautés de la version 1.4 de jQuery est de pouvoir **passer des fonctions en argument à toutes les méthodes setter**. Avant la 1.4, seules quelques méthodes (`attr()`, `css()` ...), que nous allons apprendre plus tard, avaient cette fonctionnalité. Le premier argument de ces fonctions est le numéro de l'élément en question dans l'objet jQuery. Le second argument est parfois (ce sera mentionné) la **valeur de ce qu'on définit** (revient au même que le résultat de la méthode utilisée en getter).

Ainsi la plupart des méthodes apprises lors du chapitre précédent acceptent des fonctions en argument ! Ainsi, sont concernées : `html()`, `text()`, `replaceWith()`, `prepend()`, `append()`, `after()`, `before()`, `wrap()`, `wrapInner()`.

Les méthodes du second chapitre dont le second argument de la fonction est la valeur qu'on définit sont : `html()`, `text()`, `append()` et `prepend()`.

Code : JavaScript

```
$('p').prepend(function(i){
  return i + 'ème : ';
});
```

Ce code ajoute à chaque début de paragraphe le combien-ème il est dans la page.

Créer des éléments

Créer des éléments en utilisant le DOM...

Au lieu de donner un sélecteur à la fonction principale, on peut aussi **créer des éléments** : la chaîne de caractères est alors le **nom de la balise comme si elle était simple** (c'est-à-dire le nom de la balise entourée de chevrons ainsi que le slash utilisé afin de fermer les balises avant le chevron droit).

Afin de créer un élément, on utilisait `document.createElement('balise')` ; c'est donc `$('<balise/>')` avec jQuery. Par exemple `$('<h1 />')` crée un titre de niveau 1.

... ou en utilisant `innerHTML` !

On peut aussi **passer du html à la fonction principale** avec des balises, des attributs et leurs valeurs ainsi que du contenu (qui peut être d'autres balises).
Par exemple ce code produit une zone de texte avec ses attributs, son contenu et son style :

Code : JavaScript

```
$('<textarea id="zonedetexte" cols="28" style="border: 2px solid
#ff8;">Ceci est une\nzone de texte !</textarea>');
```

Il y a un gros problème à cette utilisation de jQuery : elle n'utilise pas le DOM, mais se contente tout simplement de créer une `div` remplie du code html en utilisant `innerHTML`.

Cette utilisation de jQuery, bien qu'**utile et fonctionnelle**, montre ses limites en ne **respectant pas les standards du W3C** et en ne faisant pas partie du DOM.

Vérifier qu'on utilise bien le DOM

Je vous parle de tout ça, mais vous n'avez aucune preuve qu'on appelle bien `document.createElement()` quand on fait `$('<balise/>')`. En voici une :

Code : JavaScript - Essayez ce code !

```
document._createElement = document.createElement;
document.createElement = function(balise){
  alert(balise+' créée !');
  document._createElement(balise);
};
```

Le principe est de faire une copie de `createElement()` et ainsi le redéfinir en prévenant avec une `alert()` de la balise créée.
Si quand on appuie un bouton (par exemple `$('<span></span>')`), surgit "div créée", c'est que jQuery fait appel à `document.createElement()` afin de créer une `div` remplie du code html en utilisant `innerHTML` (cf. plus haut).

Placer ces éléments

Sachez que **ces éléments n'existent pas** dans la page web !
Afin de les placer, vous pouvez utiliser les méthodes apprises lors du chapitre précédent (quand il est spécifié « objet jQuery »).

Dans ce code :

Code : JavaScript

```
$('p.message_forum')
  .before(
    $('<div class="alerte">'
       +'Le webmaster ne saurait être tenu responsable des propos
des internautes.'
     +'</div>'));
```

Il ne sert à rien d'utiliser la fonction `$()` dans `before()`. Quitte à **injecter du code html**, autant se passer de l'appel à `$()`.

Les **réelles utilités** de cette utilisation de la fonction principale sont pour les méthodes `prependTo()`, `appendTo()`, `insertBefore()`, `insertAfter()` ainsi que `replaceAll()` (en effet on ne peut pas utiliser les méthodes jQuery avec des chaînes de caractères) :

Code : JavaScript - Essayez ce code !

```
$('<a href="lienmort.php" class="liemort">'
   +'Avertir le webmaster que ce lien est mort.'
 +'</a>')
    .insertAfter('a');

$('<legend>'
   +'Formulaire'
 +'</legend>')
    .prependTo('fieldset');

$('<h1>'
   +$('#titre').html()
 +'</h1>')
  .replaceAll('#titre');
```

L'indentation est là pour vous aider à la lecture.

Attributs

`attr()` est une méthode qui peut s'utiliser de quatre façons différentes.

Récupérer les attributs

`attr('attribut')` renvoie la valeur de l'attribut `attribut`. Le premier paramètre est donc une chaîne de caractères représentant l'attribut en question.

Code : JavaScript - Essayez ce code !

```
// Renvoie l'attribut 'title'
// de l'élément ayant comme id 'titre'.
$('#titre').attr('title');
// Écrit après #titre son attribut 'title'.
$('#titre').after($('#titre').attr('title'));
```

Définir les attributs

attr(attribut, valeur)

`attr('attribut','valeur')` définit la valeur de l'attribut `attribut` à `valeur`. Les paramètres sont donc des chaînes de caractères représentant, pour le premier, l'attribut en question, et le second, sa valeur.

Code : JavaScript - Essayez ce code !

```
$('div.header_gauche img').attr('title','le Site du Zér0');
```

Cette méthode se révèle pratique pour **précharger des images** en JavaScript : quand on a, par exemple, des éléments qui changent d'image arrière-plan lors du survol de la souris (`element:hover{background-url: url(nouvelleImage.png)}`), cette image n'est téléchargée que lorsque la souris survole l'élément (ce qui prend donc du temps).
Pour y remédier, cette fonction précharge une image en la créant puis en lui attribuant sa source, tout ça **sans la placer dans le document** :

Code : JavaScript

```
function prechargerImage(url){
  $('<img />').attr('src',url);
}

// Utilisation :
prechargerImage('logoDeMonSite.png');
```

On peut facilement imaginer une fonction qui prend autant d'arguments qu'on veut et qui précharge toutes les images (voir fin du chapitre pour `each()`) :

Code : JavaScript - Essayez ce code !

```
function prechargerImages(){
  $.each(arguments,function(){
    $('<img />').attr('src',this);
  });
}
```

```
// Utilisation :
prechargerImages(
  'logoDeMonSite.png',
  'imageSurvol.png',
  'imageClique.png');
```

Pour rappel `arguments` est le tableau contenant tous les arguments passés à la fonction.

attr(liste de couples attribut / valeur ou fonction)

Cette utilisation est la même que la précédente, sauf qu'on peut définir plusieurs couples attributs / valeurs à la fois. On utilise donc un objet.

Si vous ne savez pas ce qu'est un objet, il serait temps de relire le tuto JS  .

Donc pour résumer : on met une accolade ouvrante avant et une accolade fermante après, on ne met pas de guillemet à l'attribut, on sépare l'attribut de la valeur par deux points et on sépare chaque couple attributs / valeurs par une virgule.

Code : JavaScript - Essayez ce code !

```
$('img').attr({
  title : 'Mes photos de vacances',
  alt : 'Ceci est une image',
  src : 'vacances.jpg'
});
```

Ce code change tous les attributs `title`, `alt` et `src` de toutes les images.

On peut aussi procéder de cette façon :

Code : JavaScript

```
var attributsImages = {
  title : 'Mes photos de vacances',
  alt : 'Ceci est une image',
  src : 'vacances.jpg'
};

$('img').attr(attributsImages);
```

Il existe une légère différence d'appellation des attributs dans la notation d'objet JavaScript (JSON) :
Quand on désigne un attribut dans une liste de propriétés, si on veut utiliser des tirets, il suffit de mettre des guillemets.

Si on veut les enlever et tout de même mettre des tirets, il suffit de supprimer ces derniers et de mettre une majuscule au caractère qui les suit (par exemple, `position-horizontale` deviendra `positionHorizontale`)
C'est la même différence qu'entre `objet.attribut` et `objet['attribut']` !

attr(attribut, fonction)

On va créer une fonction anonyme. Cette fonction peut avoir un argument qui représente l'indice de l'élément en question ou non (il est facultatif). La variable commence à zéro et peut être nommée comme on le veut.
La fonction doit retourner la valeur de l'attribut.

Code : JavaScript - Essayez ce code !

```
$('img').attr('title',function(i){
  return 'Photo n°'+(i+1)+' : '+$(this).attr('src');
});
```

Notez le `$(this)` qui n'aurait pas été possible sans la fonction anonyme !

Si vous n'avez pas compris ce qu'est le `$(this)`, je vous invite à **relire le second sous-chapitre** !
Ce point est essentiel et doit être absolument acquis !

Pour résumer, `this` est l'**élément du DOM** concerné par la fonction (la fonction sera appelé une fois pour chaque image), donc ici une image, et `$(this)` **l'objet jQuery** produit à partir de cet élément.
On a ici une **nouvelle utilisation de la fonction principale jQuery**, qui permet à partir d'un élément - ou noeud - du DOM, d'associer un objet jQuery avec lequel on pourra utiliser les méthodes jQuery - par exemple ici `attr()` (contrairement à `this` où on ne peut utiliser que les méthodes du JavaScript).

Dans une liste de couples attributs / valeurs, on peut aussi mettre des fonctions anonymes en guise de valeurs. 

Code : JavaScript - Essayez ce code !

```
$('img').attr({
  title : function(i){
    return 'Photo n°'+(i+1)+' : '+$(this).attr('src');
  }
});
```

Autre exemple qui permet d' « anonymiser » votre site des liens que vous publiez :

Code : JavaScript

```
$('a').attr('href',function(i){
  return 'http://anonym.to/?'+$(this).attr('href');
});
```

Ainsi le site cible ne saura pas que son visiteur provient de votre site !

Remarque : depuis la version 1.4, il y a un **second argument** à la fonction de retour, qui est la **valeur de l'attribut qu'on défini** (ce qui rejoint ce qui est dit à la fin du second sous-chapitre).

On peut donc écrire le code ci-dessus plus simplement :

Code : JavaScript

```
$('a').attr('href',function(i,href){
  return 'http://anonym.to/?'+href;
});
```

Supprimer un attribut

`removeAttr(attribut)` est une méthode permettant de supprimer l'attribut en paramètre. Cette méthode ne peut supprimer qu'un seul attribut à la fois.

Code : JavaScript - Essayez ce code !

```
// Tous les éléments de votre page perdront leurs classes.
$('*').removeAttr('class');
// Tous les liens de votre page ne s'ouvriront pas dans une nouvelle fenêtre .
$('a').removeAttr('target');
// Décoche toutes les checkbox et tous les boutons radio de la page.
$(':checkbox').removeAttr('checked');
```

Sélecteurs

Il existe des sélecteurs qui permettent de filtrer les éléments selon leurs attributs (j'en avais déjà donné deux comme exemples dans le premier chapitre sans les expliquer) :

Expression	Retour
`[attr]`	Éléments qui **ont** l'attribut `attr`.
`[attr="val"]`	Éléments dont la valeur de l'attribut `attr` **est égale à** par `val`.
`[attr*="val"]`	Éléments dont la valeur de l'attribut `attr` **contient** `val`.
`[attr!="val"]`	Éléments dont la valeur de l'attribut `attr` **ne contient pas** `val`.
`[attr^="val"]`	Éléments dont la valeur de l'attribut `attr` **commence** par `val`.
`[attr$="val"]`	Éléments dont la valeur de l'attribut `attr` **finit** par `val`.

Code : JavaScript - Essayez ce code !

```
// Change l'attribut target en _top
// de tous les liens qui ont l'attribut target à _blank.
$('a[target="_blank"]').attr('target','_top');
$('img[src$=".png"]').attr('title','cette image est un png');
$('*[id]').attr('title','cet élément a un id');
```

Note : on n'est pas obligé de mettre une étoile afin de désigner tous les éléments (on peut ne rien mettre à la place)

Ces sélecteurs peuvent s'ajouter entre eux :

Code : JavaScript - Essayez ce code !

```
$('.citation[typeQuestion="exclamation"][langueQuestion="espagnol"]')
  .prepend('¡ ')
  .append(' !');
$('.citation[typeQuestion="question"][langueQuestion="francais"]').append('
?');
```

Attribut défini ou non

Si l'attribut n'existe pas ou n'est pas défini, `attr()` (en getter bien sûr) renvoie **`undefined`** .

Formulaires

Sélecteurs

Il existe des sélecteurs spécifiques aux formulaires :

- `:checked` qui permet de sélectionner les `checkbox` cochées ou boutons `radio` sélectionnés.
  => `<input type="checkbox" checked="checked" ,<input type="radio" checked="checked" />`
- `:selected` qui permet de sélectionner les `option` (provenant de `select`) sélectionnés.
  => `<select><option selected="selected"</option></select>`
- `:disabled` qui permet de sélectionner les éléments de formulaires désactivés.
- `:enabled` qui permet de sélectionner les éléments de formulaires actifs.
- `:input` désigne tous les éléments d'un formulaire.
  => `<input />`, `<textarea></textarea>`, `<select></select>`, `<button></button>`
- `:button` désigne tous les boutons.
  => `<input type="button" />` et `<button></button>`
- `:reset` désigne tous les boutons qui remettent à zéro le formulaire.
  => `<input type="reset" />` et `<button type="reset"></button>`
- `:submit` désigne tous les boutons qui envoient le formulaire.
  => `<input type="submit" />` et `<button type="submit"></button>`
- `:checkbox` désigne toutes les cases à cocher.
  => `<input type="checkbox" />`
- `:radio` désigne tous les boutons radio.
  => `<input type="radio" />`
- `:text` désigne tous les champs de texte.
  => `<input type="text" />`
- `:password` désigne tous les champs de mots de passes.
  => `<input type="password" />`
- `:file` désigne tous les champs d'envoi de fichier.
  => `<input type="file" />`
- `:image` désigne tous les champs image.
  => `<input type="image" />`
- `:hidden` désigne aussi les éléments du formulaire cachés.
  => `<input type="hidden" />`

Récupérer les valeurs

`val()` renvoie la valeur d'une balise de formulaire. La fonction peut renvoyer différents types d'information.

- **`<input type="text" />`**, **`<textarea></textarea>`**, **`<input type="radio" />`** et **`<select></select>`** : renvoie une chaîne de caractères représentant l'attribut `value`, pour les `text` et `radio` (celle sélectionnée), le contenu de la balise `textarea` pour cette balise, et le contenu de la balise `option` sélectionnée pour les `select`.
- **`<input type="checkbox" />`**, **`<select multiple="multiple"></select>`** : renvoie un tableau de chaînes de caractères représentant les attributs `value` des `checkbox` cochées, et les contenus des balises `option` sélectionnées.

J'appellerai les formulaires de la première puce des formulaires "simples" et les formulaires de la seconde puce des formulaires "compliqués".

Formulaires simples

Dans ce bout de code :

Code : HTML

```
<input type="text" id="texte" value="Salut!" />
```

```
<textarea id="zonetexte">Ceci est une
zone de texte !</textarea>
<input type="radio" name="choixradio" value="Radio 1" />Radio 1
<input type="radio" name="choixradio" value="Radio 2" />Radio 2
<input type="radio" name="choixradio" value="Radio 3" />Radio 3
<select id="choixselect">
  <option>Select 1</option>
  <option>Select 2</option>
  <option>Select 3</option>
</select>
```

On peut utiliser pour récupérer les valeurs de ces éléments :

Code : JavaScript - Essayez ce code !

```
// Renvoie 'Salut!'.
$("#texte").val();

// Renvoie 'Ceci est une\nzone de texte !'.
$("#zonetexte").val();

// Renvoie l'attribut value de la balise sélectionnée.
// Par exemple 'Radio 1'.
$('input[name="choixradio"]:checked').val();

// Renvoie le contenu de la balise option sélectionnée.
// Par exemple 'Select 1'.
$('#choixselect').val();
```

Explication du troisième :
Il y a plusieurs boutons radio. On les repère alors grâce à leur attribut `name`. Or `val()` est fait pour un élément, il va donc prendre le premier qu'il trouve et renvoyer la valeur donc « Radio 1 ». Pour sélectionner celui qui est couramment sélectionné, on utilise le sélecteur `:checked` !

Formulaires compliqués

Dans ce bout de code :

Code : HTML

```
<input type="checkbox" name="choixcheck" value="Choix CheckBox 1" />
Choix Checkbox 1
<input type="checkbox" name="choixcheck" value="Choix CheckBox 2" />
Choix Checkbox 2
<input type="checkbox" name="choixcheck" value="Choix CheckBox 3" />
Choix Checkbox 3
<select id="choixmultiple" multiple="multiple">
  <option>Select Multiple 1</option>
  <option>Select Multiple 2</option>
  <option>Select Multiple 3</option>
</select>
```

On peut utiliser pour récupérer les valeurs de ces éléments :

Code : JavaScript - Essayez ce code !

```
// checkbox sera différent selon que les checkbox sont cochées ou
```

```
non.
// Par exemple :
// '<br />Choix CheckBox 1 : Non Cochée' ;
// '<br />Choix CheckBox 2 : Cochée' ;
// '<br />Choix CheckBox 3 : Non Cochée'.

var checkbox = "";
$('input[name="choixcheck"]').each(function(){
  checkbox +=
    '<br />'+$(this).attr('value')+' : '+
    ($(this).attr('checked') == true ? '' : 'Non ')+'Cochée';
});

// ALTERNATIVE :
// checkbox contiendra une liste de toutes les checkbox cochées.
// Par exemple "Choix CheckBox 1,Choix CheckBox 2".

var checkbox = "";
$('input[name="choixcheck"]:checked').each(function(i){
  checkbox += (i>0 ? ',' : '')+$(this).attr('value');
});

// Renvoie les contenus des balises option sélectionnées.
// Par exemple "Select Multiple 1,Select Multiple 2,Select Multiple
3".
// Si elles sont toutes les trois sélectionnés.
$('#choixmultiple').val();
```

Voir à la fin de ce chapitre pour comprendre `each()`.

Définir les valeurs

`val(valeur)` permet de définir la ou les valeur(s) des éléments d'un formulaire.

- `valeur` doit être une chaîne de caractères pour les **<input type="text" />**, **<textarea></textarea>** et **<select></select>**.
- `valeur` doit être un tableau de chaînes de caractères pour les **<input type="checkbox" />**, **<select multiple="multiple"></select>**.

Exemples :

- `<input type="text" />` => `$('#texte').val('Bonjour!');`
- `<textarea></textarea>` => `$('#zonetexte').val('Ceci est une grande\nzone de texte !');`
- `<select></select>` => `$('#choixselect').val('Select 2');`
- `<input type="checkbox" />` => `$('input[name="choixcheck"]').val(['Choix CheckBox 1','Choix CheckBox 2']);`
- `<select multiple="multiple"></select>` => `$('#choixmultiple').val(['Select Multiple 1','Select Multiple 3']);`

On ne peut pas sélectionner un bouton radio avec `val('valeur')` ! Si vous essayez, cette méthode changera l'attribut `value` du radio et ne le sélectionnera pas.

Pour changer les valeurs des `<input type="radio" />`, on utilise `attr('checked','checked')` sur un élément :

Code : JavaScript

```
// Sélectionne le second.
$('input[name="choixradio"]:eq(1)').attr('checked','checked');
```

Vous pouvez tester ces codes définissant les valeurs des formulaires sur cette page.

Boucler sur des éléments

each() sur un objet jQuery

`each()` est une méthode qui permet **d'itérer (de boucler) sur les éléments d'un objet jQuery**. C'est un peu l'équivalent en plus simple d'une boucle `for`.

On lui passe en argument une fonction anonyme qui va être appelée **autant de fois qu'il y a d'éléments** dans l'objet jQuery. La fonction sera donc un bout de code qui **s'exécutera pour chaque élément**. Si cette fonction retourne *false*, `each()` **s'arrête d'itérer**.

Cette fonction peut avoir un argument qui représente l'indice de l'élément en question (c'est le combien-ème ?) ou non. La variable commence à zéro et peut être nommée comme on le veut.
`this` représentera l'élément en question donc `$(this)` **l'objet jQuery en question**.

Code : JavaScript - Essayez ce code !

```
$('a').each(function(i) {
  $(this)
    .prepend('<small>Lien n°'+(i+1)+' »</small> ')
    .append(' <small>« '+$(this).attr('href')+'</small>');
});
```

Nous ne sommes pas obligés de nommer la variable `i` !

Code : JavaScript - Essayez ce code !

```
$('#checkboxs :checkbox').each(function(i){
  if(!$(this).attr('checked')){
    alert(i+" checkboxs cochées d'affilée depuis le début");
    return false;
  }
});
```

Voir aussi les codes plus haut dans le chapitre beaucoup plus intéressants qui nécessitent `each()` afin de les comprendre (ici et là).

N'oubliez pas que, comme dit au début du chapitre, la plupart des méthodes simples apprises au chapitre d'avant peuvent prendre une fonction en argument depuis la version 1.4.
On peut donc **éviter parfois d'utiliser `each()` !**

Donc le premier exemple pourrait s'écrire :

Code : JavaScript

```
$('a')
  .prepend(function(i){
    return '<small>Lien n°'+(i+1)+' »</small> ';
  })
  .append(function(i){
    return ' <small>« '+$(this).attr('href')+'</small>';
  });
```

A vous de choisir ce que vous préférez. 

each() avec des données

`$.each()` est une fonction (contenue dans l'objet jQuery) dérivée de la méthode `each()`, la différence étant qu'elle permet d'itérer sur un tableau ou un objet, passé en premier argument.
Son utilisation est assez simple :

- si la variable passée en paramètre est un **tableau**, le premier argument de la fonction est l'**index** de l'élément courant dans le tableau et le second argument (ainsi que le mot-clé `this`) est la **valeur** de l'élément ;
- si la variable passée en paramètre est un **objet**, le premier argument de la fonction est un **attribut** et le second argument (ainsi que le mot-clé `this`) est la **valeur** de cet attribut.

Dans le second cas, faites attention à ne pas mettre un attribut `length` à l'objet passé en paramètre à `$.each()`, sinon cette fonction va agir comme si l'objet était un tableau (et ainsi n'agir que sur les éléments `0`, `1` ... jusqu'à `length-1`).

Exemples avec des tableaux

Code : JavaScript - Essayez ce code !

```
$.each([
  0,1,1,2,3,5,8,13,21
],function(n){
  $('body').append(n+'ème nombre de Fibonacci : '+this+'<br />');
});
```

Code : JavaScript - Essayez ce code !

```
var tableau = [
  'Bonjour, comment vas-tu ?',
  "Aujourd'hui, il fait beau !",
  $('q:first').text()
]
,chaine = '';

$.each(tableau,function(n,texte){
  chaine += 'Texte n°'+(n+1)+' : '+texte+'\n';
});

$('<pre></pre>')
  .html(chaine)
  .appendTo('#contenu');
```

Exemples avec des objets

Code : JavaScript - Essayez ce code !

```
var objet = {
  span: $('#contenu span'),
  listes: $('#contenu ul,#contenu ol'),
  tableaux: $('#contenu table'),
  'expression compliquée': $('#contenu h5 >
div:has(blockquote):contains("Bonjour") + strong')
}
,chaine = ''
,fonction = function(balise,requete){
  chaine += 'Nombre de '+balise+'
(<code>'+requete.selector+'</code>) : '+requete.size()+'\n';
};
```

```
$.each(objet,fonction);

$('<pre></pre>')
  .html(chaine)
  .appendTo('#contenu');
```

Notez bien que `$(expression).each(fonction)` est équivalent à `$.each($(expression),fonction)` .

each() : passer des arguments à la fonction de retour

Il peut arriver qu'on passe à `$.each()` une fonction déclarée avant l'appel à la méthode en question.
Ce dernier ne peut donc pas accéder aux variables déclarées dans le bloc qui contient l'appel à cette méthode.

Pour palier ce problème, un troisième argument, facultatif, existe pour `$.each()` (second pour `each()`) : un tableau.
Les valeurs de ce tableau passé en paramètre seront **les arguments de la fonction** de retour passée.
Cette fonction n'a donc plus les deux premiers arguments cités plus haut. On ne peut donc plus **connaître l'index** ou encore **le nom de l'attribut**.

Code : JavaScript - Essayez ce code !

```
function faireSurTexte(fonction){
  var elems1   = $('p');
  var elems2   = $('span');
  var longueur = elems1.length + elems2.length;
  var couleur  = 'rgb(255,192,0)';
  elems1.each(fonction,[longueur,couleur]);
  elems2.each(fonction,[longueur,couleur]);
}

$(function(){

faireSurTexte(function(longueur,couleur){
  $(this).attr('style','color:'+couleur);
  $(this).append(" nombre d'éléments : "+longueur);
});

});
```

Cet exemple est vraiment mauvais, mais peut-être un jour aurez-vous besoin de cette petite astuce.

Ceci n'est pas marqué dans la documentation de jQuery mais ne vous inquiétez pas, cela marche.

map() récupère les informations des éléments d'un objet jQuery

`map()` et `$.map()` sont des dérivés de `each()` et `$.each()`. La méthode boucle sur l'objet jQuery et la fonction boucle sur les données (passés en paramètres).
`map()` renvoie l'objet jQuery où chaque élément est remplacé par ce qu'a retourné la fonction (**`return`**).

Le gros problème est qu'on ne peut pas utiliser cet objet jQuery comme un tableau ;
exécuter par exemple `join(', ')` afin de séparer les valeurs et transformer le tableau en chaîne de caractères.

La réponse est bien évidemment la méthode `get()` apprise au début du chapitre.

Code : JavaScript - Essayez ce code !

```
// Liste tous les liens de la page.
alert(
  $('a').map(function(){
    return $(this).attr('href');
  }).get().join('\n'));

alert(
  $('#contenu *').map(function(){
    return this.tagName;
  }).get().join(', '));
```

`$.map()` renvoie un tableau où chaque élément est remplacé par ce qu'a retourné la fonction. On peut passer un objet, à condition de spécifier un attribut `length`, et la fonction va parcourir les éléments `0`, `1` ... jusqu'à `length-1`.

A l'inverse de `$.each()`, `$(expression).map(fonction)` **n'est pas équivalent** à `$.map($(expression),fonction)` ; les différences sont **les arguments de la fonction** de retour ainsi que le mot-clé `this` :

- Pour `map()`, le premier argument est l'**index** et le second ainsi que `this` l'**élément courant**.
- Pour `$.map()`, le premier argument est la valeur du tableau et le second l'index. `this` n'est **pas précisé**, donc est égale à la variable `window`.

TP : Sommaire automatique

Code de base

On part du principe que votre page web, qui est votre explication de quelque chose (par exemple un tutoriel !), ne contient que des **titres de niveau 1 et 2** afin de diviser et structurer le texte.
Tout le texte suivant un titre de niveau 1 est contenu dans une `div` de classe `chapitre` (par exemple), si il y a des titres de niveau 2.
Tout le contenu est dans une `div` qu'on peut repérer grâce à son id qui est, pour faire original, `contenu` .

Par exemple :

Secret (cliquez pour afficher)

Code : HTML

```
<div id="contenu">

  <p>
    Introduction
  </p>

  <h1>Chapitre Un</h1>
  <div class="chapitre">
    <h2>a)</h2>
    <p>Paragraphes.</p>

    <h2>b)</h2>
    <p>Paragraphes.</p>
  </div>

  <h1>Chapitre Deux</h1>
  <p>Paragraphes.</p>

  <h1>Conclusion</h1>
  <div class="chapitre">
    <h2>Que retenir ?</h2>
    <p>Paragraphe.</p>

    <h2>Remerciements</h2>
    <p>Paragraphe.</p>
  </div>

</div>
```

Votre but

Votre but est d'indexer ces titres afin de créer un **sommaire**. Ce sommaire contiendra des listes avec des **liens vers les ancres** des titres (leur attribut `id`) : les titres de niveau 1 sont la **première liste** et dans chaque élément de cette liste il y a une **nouvelle liste** qui sont les titres de niveau 2 (si ils existent).

Allez voir le sommaire de n'importe quelle page assez longue de Wikipédia : le résultat doit être à peu près celui que vous verrez .

Le code final produit avec le code d'exemple en haut peut ressembler à ceci (sans les explications bien sûr) :

Code : HTML

```
<div id="sommaire">
  <ol class="niveau1">
```

```
        <!-- premier h1 trouvé -->
        <li><a href="#ancre0">Chapitre Un</a>
          <ol class="niveau2">
            <li>
              <!-- premier h2 trouvé dans la première div.chapitre -->
              <a href="#ancre0-1">a)</a>
            </li>
            <li>
              <a href="#ancre0-2">b)</a>
            </li>
          </ol>
        </li>
        <!-- second h1 trouvé -->
        <li><a href="#ancre1">Chapitre Deux</a></li>
        <!-- troisième h1 trouvé -->
        <li><a href="#ancre2">Conclusion</a>
          <ol class="niveau2">
            <li>
              <!-- premier h2 trouvé dans la troisième div.chapitre -->
              <a href="#ancre2-1">Que retenir ?</a>
            </li>
            <li>
              <a href="#ancre2-2">Remerciements</a>
            </li>
          </ol>
        </li>
      <ol>
    </div>
    <div id="contenu">
      <!-- le contenu de votre choix avec des balises
    <h1> et <h2> (contenues dans des div.chapitre)
    + les ancres créées par le JavaScript
    -->
    </div>
```

Kappisch  ?
Pour rappel, on accède à une ancre en faisant `<a href="#ancre"></a>` . Ce `ancre` peut être l'attribut `name` d'un autre lien, mais aussi **l'attribut `id` de n'importe quelle élément de la page**.
Il faudra donc **attribuer un id à chaque titre**.

Solution

Voici le code JavaScript avec toutes les explications nécessaires :

Code : JavaScript - Essayez ce code !

```
// On crée le sommaire tout en haut
$('#contenu').before('<div id="sommaire"><ol
class="niveau1"></ol></div>');

// Pour chaque titre <h1>
$('#contenu h1').each(function(numTitre1,titre1){
  // numTitre1 est le numéro du h1 en question, partant de 0
  // titre1 est l'élément h1 en question, donc on peut le manipuler
avec $(titre1)
  // $('h1:eq('+numTitre1+')') est donc la requête permettant
d'acceder a cet h1

  // On ajoute l'id au titre, pour l'ancre
  $(titre1).attr('id', 'ancre-'+numTitre1);

  // On ajoute une ligne avec le texte du h1
  // + le lien de vers son ancre attache
```

```
        $('#sommaire .niveau1').append(
             '<li id="sommaire-'+numTitre1+'">'
               +'<a href="#ancre-'+numTitre1+'">'+$(titre1).text()+'</a>'
           +'</li>');
        // il ne faut pas oublier d'attribuer un id a cet li
        // pour pouvoir y rajouter des h2 plus tard


        // Si cet h1 a des h2
        if($('#contenu h1:eq('+numTitre1+') + div.chapitre').length == 1){
          // On crée une liste de sous-parties
          $('#sommaire-'+numTitre1).append('<ol class="niveau2"></ol>');

          // Pour chaque h2 dans le h1 en question
          $('#contenu h1:eq('+numTitre1+') + div.chapitre >
    h2').each(function(numTitre2,titre2){
            // On ajoute l'id au sous-titre, pour l'ancre
            $(titre2).attr('id', 'ancre-'+numTitre1+'-'+numTitre2);

            // On ajoute une ligne avec le nom de ce h2 avec un lien
            // + le lien de l'ancre vers le sous-titre
            $('#sommaire-'+numTitre1+' ol').append(
                 '<li>'
                  +'<a href="#ancre-'+numTitre1+'-
    '+numTitre2+'">'+$(titre2).text()+'</a>'
              +'</li>');
          });
        }
    });
```

Améliorations

Pour vous entraîner, voici quelques améliorations que vous pourriez programmer :

- Prévoir le script pour des `h3`, `h4`, `h5` et `h6` ;
- Rendre visible les ancres pour accéder au sommaire (ou en haut de la page) en cliquant dessus ;

Ce chapitre étant le plus dur de cette partie, c'est sûrement celui où vous avez appris le plus de choses !

Appris dans ce chapitre : variable `tagName`, ce que contient un objet jQuery, transformer un objet jQuery en tableau, méthodes getter et méthodes setter, passer une fonction en argument aux méthode du chapitre précédent, variable `this` dans ces fonctions, créer un élément, placer un élément créé, sélecteurs des attributs, sélecteurs des formulaires, choisir un bouton radio, méthodes : `attr()`, `removeAttr()`, `val()`, `each()`, `$.each()`, `map()`, `$.map()`.

Décorer la page

L'**accessibilité et la modification du style** des éléments de votre page, ainsi que leurs classes, leurs dimensions, leurs ascenseurs et enfin leurs différentes positions seront étudiés dans ce très court chapitre.

N'oubliez pas que dans les méthodes que l'on va apprendre, vous pourrez **passer des fonctions en argument** (en setter) !

Style

`css()` peut s'utiliser de **plusieurs façons différentes**, les mêmes que `attr()`. Pour résumer :

- `css('attribut')` permet de **récupérer la valeur d'un attribut** CSS
- `css('attribut',valeur)` permet de **définir un attribut** CSS
- `css({attribut1: valeur1, attribut2: valeur2})` permet de **définir plusieurs attributs** CSS
- `valeur` dans les deux lignes précédentes peut être une fonction qui **retourne la valeur** de l'attribut CSS.

N'oubliez pas que dans le troisième cas, l'attribut `border-left-style` par exemple, **s'écrira `borderLeftStyle`** ou `'border-left-style'`.

A retenir :

1. `float` est un mot clé du langage JavaScript, il est donc préférable de l'entourer de guillemets.
2. L'attribut `opacity`, un nombre (ou chaîne de caractères) compris entre 0 et 1, peut être utilisé et marche sous Internet Explorer.

Code : JavaScript - Essayez ce code !

```
// Couleur de fond cyan.
$('body').css('background-color','#0ff');

// Formatage des liens.
$('a')
  .css({
    borderBottom: '2px dashed #88d',
    color: '#44b',
    paddingLeft: '4px',
    'float': 'left',
    margin: '4px',
    'letter-spacing': '2px',
    textDecoration: function(){
      return $(this).attr('decoration');
    },
    opacity: 0.8
  })
  .attr('title',function(){
    return 'couleur : '+$(this).css('color')
  });
```

Classes

Enlever et ajouter des classes

Au même titre que `removeAttr()`, `removeClass()` vous permettra d'**enlever une ou plusieurs classes** et `addClass()` vous permettra d'**ajouter une ou plusieurs classes**.

Ces deux méthodes ne peuvent prendre qu'un seul argument.
Pour les utiliser avec plusieurs classes différentes, il suffit de les séparer par des espaces.

Code : JavaScript - Essayez ce code !

```
$('span:first').addClass('premier');
$('a[href*="moi.free.fr"]')
  .addClass('lienInterne')
  .removeClass('lienExterne');
```

Interchanger des classes

`toggleClass()` vous permettra de **jongler entre la présence et l'absence d'une ou de plusieurs classes**.

Code : JavaScript - Essayez ce code !

```
function interchangerClasseSurligner(){
  $('a').toggleClass('surligner');
}

function interchangerClasseRouge(){
  $('a').toggleClass('rouge');
}
```

Son deuxième paramètre, facultatif, est un *booléen*, qui à *false* fait l'effet d'un `removeClass()`, et qui à *true* fait l'effet d'un `addClass()`.

Pour ces méthodes (`addClass()`, `removeClass()`, et `toggleClass()`), le second argument de la fonction qu'on passe en argument est **la classe de l'élément en question**.

Vérifier la possession d'une classe

`hasClass()` est une méthode qui permet de **déterminer si un élément a une classe** (renvoie *true*) ou non (renvoie *false*).

Ne marche pas avec plusieurs classes.

Code : JavaScript - Essayez ce code !

```
// Si le premier paragraphe a la classe 'important' :
// changer son attribut title en 'Ce paragraphe est important.' ;
// sinon changer son attribut title en "Ce paragraphe n'est pas
important.".
$('p:first').hasClass('important')
  ? $('p:first').attr('title','Ce paragraphe est important.')
  : $('p:first').attr('title',"Ce paragraphe n'est pas important.");

// Renvoie forcément faux.
```

```
        $('*:not(.salut)').hasClass('salut');
```

Dimensions

Récupérer les dimensions

Un élément a **une seule** largeur et hauteur, mais il y a plusieurs façons de les définir : on compte **les bordures, les marges, le padding** (marges intérieures) ou pas ? C'est ce qu'on appelle le « box model ».
Ainsi, on différencie **quatre couples de méthodes** (le dernier est dérivé du troisième), dans l'ordre croissant du nombre donné :

Méthodes afin de récupérer les dimensions

Hauteur	Largeur	Padding ?	Bordures ?	Marges ?
`height()` (*hauteur*)	`width()` (*largeur*)	**non**	**non**	**non**
`innerHeight()` (*hauteur intérieure*)	`innerWidth()` (*largeur intérieure*)	**oui**	**non**	**non**
`outerHeight()` (*hauteur extérieure*)	`outerWidth()` (*largeur extérieure*)	**oui**	**oui**	**non**
`outerHeight(true)` (*idem*)	`outerWidth(true)` (*idem*)	**oui**	**oui**	**oui**

Code : CSS - Essayez ce code !

```css
#rectangle
{
  position: relative;
  outline: 7px dotted #fdd;
  border: 4px dashed #ddf;
  margin: 5px;
  padding: 3px;
  background-color: #dfd;
}

#rectangle div
{
  margin: 24px 2px 0 5px;
  padding-left: 83px;
  background-color: #999;
}

#commentaires
{
  font-family: monospace;
}
```

Définir les dimensions

Seul le premier couple de méthodes vous permettra de **définir ces dimensions** (en passant en argument un nombre représentant cette hauteur ou largeur).

Code : JavaScript - Essayez ce code !

```javascript
// Pour que le logo prenne presque toute la page.
// (Math.round(x) retourne l'entier le plus proche de x.)
$('img#logo').width(Math.round($('body').width()*0.9));
```

Ascenseurs

`scrollTop()` renvoi le nombre en pixels de défilement de l'ascenseur **vertical**.
`scrollLeft()` renvoi le nombre en pixels de défilement de l'ascenseur **horizontal**.

Ces méthodes permettent aussi de **définir ce défilement** quand on leur passe un entier positif en argument représentant le décalage désiré.

Ces méthodes ne marchent pas seulement pour le `document`, elles sont aussi valables pour les éléments dont l'`overflow` n'est pas à `visible`.

Code : JavaScript - Essayez ce code !

```
function afficherDefilements(){
  alert( [ $('#contenu div').scrollTop() , $('#contenu
div').scrollLeft() ] );
}

function definirAscenseurs(){
  $('#contenu div').scrollTop(Math.random()*$('#contenu div
p').height());
  $('#contenu div').scrollLeft(Math.random()*$('#contenu div
p').width());
}
```

Positions

Absolue

`offset()` est une méthode qui permet de récupérer la position X et Y d'un élément **par rapport à la page** (ces valeurs sont donc absolues). Elle renvoie un objet contenant `left` et `top` :

- `offset().left` renvoie la **position horizontale (X)** d'un élément par rapport à la page ;
- `offset().top` renvoie la **position verticale (Y)** d'un élément par rapport à la page.

Relative

`position()` est une méthode qui permet de récupérer la **position X et Y** d'un élément **par rapport à son parent** (ces valeurs sont donc relatives). Elle renvoie un objet contenant `left` et `top` :

- `position().left` renvoie la position horizontale d'un élément par rapport à son parent ;
- `position().top` renvoie la position verticale d'un élément par rapport à son parent.

Cette méthode ne prend pas en compte les **marges**.

Notes :

- `offset()` et `position()` ne marchent qu'avec des éléments **visibles**.
- Les valeurs retournées prennent comme **origine le coin en haut à gauche** de la page.

Définir la position absolue

Depuis la version 1.4 de jQuery, on peut **définir la position absolue d'un élément**, grâce à la méthode `offset()`.
Il suffit de lui **passer en paramètre soit un objet** contenant les propriétés `left` et / ou `top`, **soit une fonction** qui va renvoyer cet objet.

Le premier argument de la fonction de retour est l'index de l'élément en question, et le second est, comme vous vous l'attendiez, la position absolue de cet élément (objet contenant les propriétés `top` et `left`, soit le résultat de la méthode `offset()` sur cet élément).

Code : JavaScript

```
$('#boite').offset({
    left : 28
  , top  : 45
});


var hauteur = $('div.liste:first').outerHeight(true) + 2;

$('div.liste').offset(function( i , o ){
  return {
      top  : i * hauteur
    , left : o.left
  };
});
```

Si l'élément en question a sa propriété CSS `position` à `static`, elle sera alors redéfinie en `relative`, **pour pouvoir**

changer sa position par rapport au document.

La décoration de votre page n'a plus de secret pour vous, mais il ne faut pas trop en abuser au risque de faire perdre de l'accessibilité à votre site.
En effet, les utilisateurs ayant désactivé JavaScript ne verront les modifications apportées par jQuery.

Dans les versions antérieures de jQuery, on ne pouvait récupérer des informations telles que la taille et la position d'un élément.
On devait faire appel à un plugin nommé jQuery Dimensions. Il a été intégré à jQuery depuis la version 1.2.6.

Appris dans ce chapitre : méthodes : `css()`, `addClass()`, `removeClass()`, `toggleClass()`, `hasClass()`, `height()`, `innerHeight()`, `outerHeight()`, `width()`, `innerWidth()`, `outerWidth()`, `scrollTop()`, `scrollLeft()`, `offset()`, `position()`.

Animer des éléments

Une animation jQuery est le **changement du style CSS** d'un élément en **un temps donné**.
Dans ce chapitre, vous allez apprendre à **animer des éléments**, à gérer ces animations, ainsi qu'à utiliser les **animations prédéfinies** de jQuery.

Mais avant tout cela, il faut d'abord introduire la notion d'**évènements en jQuery** : en effet, ceux-ci sont assez simples en jQuery. Il est assez important de les maîtriser, et ils ont largement leur place dans la première partie du tutoriel.

Équivalents des événements

Un événement en JS est une action de l'utilisateur (mouvements de la souris, clic, etc...) ou même une action interne au navigateur (chargement de la page par exemple).

jQuery dispose de méthodes simples pour **attacher des événements à des fonctions** (ou « écouter un événement »). Nous allons donc passer en revues ces méthodes, avec lesquelles vous pourrez enfin dynamiser votre page !

Introduction

Code : JavaScript

```
element.addEventListener('evenement',function(){
  // Action
});
// ou
element.onevenement = function(){
  // Action
};
```

Ça, c'est ce que vous utilisiez avant. Eh bien, jQuery offre une **syntaxe à peu près similaire**, très facile à retenir.

Les événements avec jQuery seront créés grâce à des **méthodes** ayant pour nom le type de l'événement, l'argument étant la **fonction de retour**.

Code : JavaScript

```
// Écoute d'un événement
elements_jQuery.evenement(function(){
  // Action
});
```

Toutes les méthodes que nous allons voir ici peuvent aussi être appelées sans argument : elles servent alors à **déclencher l'action par défaut du navigateur** ainsi que celle que **vous avez définie** (grâce aux méthodes avec fonction de retour).
Les méthodes qui ne permettent pas de faire cela seront signalées.

Pour faire simple :

Code : JavaScript

```
elements_jQuery.evenement(function(){
  // Ce qu'il faut faire
  alert('Action !');
});

// Action !
elements_jQuery.evenement();
```

Résumé des équivalences en jQuery :

Formulaires

Sélection

`select` est déclenché lorsque du texte est sélectionné dans un `<input type="text" />` ainsi que dans un `<textarea></textarea>`.

Code : JavaScript

```
$(':text,textarea').select(function(){
  alert($(this).val());
});
```

Changement

`change` est déclenché lorsque le texte d'un `<input type="text" />`, d'un `<input type="password" />`, d'un `<textarea></textarea>` ainsi que le choix d'un `<select></select>` est **changé** (si après édition il est le même qu'avant édition, l'événement ne sera pas déclenché), ainsi que quand un `<input type="checkbox" />` ou un `<input type="radio" />` est cliqué.

Code : JavaScript

```
$(':input').change(function(){
  alert($(this).val());
});
```

Code : JavaScript

```
// Déclenche l'action du navigateur par défaut
// ainsi que l'action que vous avez définie.
$(':input').change();
```

Soumission du formulaire

`submit` est déclenché lorsqu'un formulaire est soumis (soit par l'intermédiaire d'un `<input type="submit" />` mais aussi grâce au JavaScript). La fonction passée en paramètre peut renvoyer *false*, et alors le formulaire n'est pas soumis. Cette méthode s'applique donc aux balises `<form></form>`.

Code : JavaScript

```
$('form[name="inscription"]').submit(function(){
  if($('form[name="inscription"] :password:first').val().length <
6){
    alert('Veuillez rentrer au moins 6 caractères dans votre mot de
passe');
    return false;
    }
});
```

Code : JavaScript

```
// Soumet tous les formulaires
// et déclenche l'action du navigateur par défaut
// ainsi que l'action que vous avez définie.
```

```
$('form').submit();
```

Focalisation

`focus` est déclenché lorsqu'un élément d'un formulaire est « focalisé » par l'utilisateur, soit en cliquant dessus, soit grâce aux raccourcis du navigateur qui permettent de parcourir la page (les tabulations par exemple). On dit qu'il **obtient le « focus »**.
`blur` est déclenché lorsqu'un élément d'un formulaire **perd le « focus »**.

Ces deux méthodes marchent avec tous les éléments formulaires (sauf `<input type="hidden" />`) ainsi qu'avec les `<iframe>` et la page web (`document`).

Code : JavaScript

```
$(':input').focus(function(){
  $(this).css('background-color','#00f');
});
$(':input').blur(function(){
  $(this).css('background-color','#f00');
});
```

Code : JavaScript

```
// Ne jamais pouvoir accéder
// à une balise formulaire.
$(':input').focus(function(){
  $(this).blur();
});

// Ou encore ne jamais pouvoir
// sortir d'une balise formulaire.
$(':input').blur(function(){
  $(this).focus();
});
```

Quand la méthode `focus()` sans aucun argument est utilisée, la fonction qu'on a définie **n'est pas appelée**.

Touches

L'appui sur une touche se décompose en trois étapes successives :

1. `keydown` : **enfoncement** de la touche ;
2. `keypress` : **maintien** de la touche enfoncée ;
3. `keyup` : **relâchement** de la touche.

Le premier argument de la fonction de retour passée en paramètre est un objet contenant des informations sur la touche appuyée.
Un attribut `which` ou `keyCode` (dépend du navigateur) désigne le numéro de la touche appuyée. Le problème est que chaque navigateur renvoie un nombre différent.

Code : JavaScript

```
$(document).keypress(function(evenement){
  // Si evenement.which existe, codeTouche vaut celui-ci.
```

```
  // Sinon codeTouche vaut evenement.keyCode.
  var codeTouche = evenement.which || evenement.keyCode;
  alert(codeTouche);
});
```

`evenement.which ? evenement.which : evenement.keyCode` est équivalent à (mais plus long) `evenement.which || evenement.keyCode`.

Code : JavaScript

```
// Déclenche l'action keypress du navigateur par défaut
// ainsi que l'action keypress que vous avez définie
// sur un textarea qui gagne le focus.
$('textarea').focus(function(){
  $(this).keypress();
});

// Vous pouvez vérifier que cela marche :
$('textarea').keypress(function(){
  alert($(this).val());
});
```

Souris

Clics de souris

Un clic de souris se décompose en trois étapes successives :

1. `mousedown` : **enfoncement** de la souris ;
2. `mouseup` : **relâchement** de la souris ;
3. `click` : **clic** de la souris.

Le premier argument de la fonction de retour passée en paramètre est un objet contenant des informations sur la touche appuyée, `pageX` pour la position X et `pageY` pour la position Y.

On peut aussi **double cliquer** grâce à l'événement `dblclick`, les trois fonctions ci-dessus seront appelées deux fois avant le déclenchement de cet événement.

Code : JavaScript - Essayez ce code !

```
$(document).mousedown(function(clic){
  $('#posX').text('Position X : '+clic.pageX);
  $('#posY').text('Position Y : '+clic.pageY);
});
```

Code : JavaScript - Essayez ce code !

```
// On ne voit pas la couleur bleue car click
// est appelé directement après mouseup.
$('*')
  .mousedown(function(){
    $('#contenu')
   .css('background-color','#f00')
      .append('<span style="color: #f00"> Down!</span>');
  })
  .mouseup(function(){
```

```
    $('#contenu')
   .css('background-color','#00f')
      .append('<span style="color: #00f"> Up!</span>');
  })
  .click(function(){
    $('#contenu')
   .css('background-color','#f0f')
      .append('<span style="color: #f0f"> Clique!</span>');
  })
  .dblclick(function(){
    $('#contenu')
   .css('background-color','#0ff')
      .append('<span style="color: #0ff"> Double Clique!</span>');
  });
```

Code : JavaScript

```
// Déclenche l'action click du navigateur par défaut
// ainsi que l'action click que vous avez définie.
$('a').click();

// Vous pouvez vérifier que cela marche en attribuant une fonction
qui va se déclencher :
$('a').click(function(){
  alert('Vous allez vers : '+$(this).attr('href'));
});
```

`$(expression).dblclick()` marche aussi.

Mouvements de souris

La souris peut **entrer au dessus** d'un élément, **bouger** sur cet élément et enfin **partir** de cet élément.

On distingue donc :

- `mouseenter` : la souris entre au-dessus de l'élément ;
- `mouseleave` : la souris quitte l'élément ;
- `mouseover` : la souris entre au-dessus de l'élément ou un de ses enfants ;
- `mouseout` : la souris quitte l'élément ou un de ses enfants ;
- `mousemove` : la souris bouge sur l'élément.

`mousemove` sera appelée plein de fois en comparaison avec les autres.

Le premier argument de la fonction de retour passée en paramètre est un objet contenant des informations sur la touche appuyée, `pageX` pour la position X et `pageY` pour la position Y.

Code : JavaScript - Essayez ce code !

```
$(document).mousemove(function(clic){
  $('#posX').text('Position X : '+clic.pageX);
  $('#posY').text('Position Y : '+clic.pageY);
});
```

Ces méthodes ne peuvent être appelées sans la fonction de retour.
`$(document).mousemove()` ne marche pas par exemple.

Fenêtre

Seule la dernière méthode pourra être appelée sans argument.

Défilement

`scroll` est déclenché lorsque l'utilisateur utilise un ascenseur horizontal ou vertical.

Code : JavaScript

```
$(document).scroll(function(){
  alert('Arrête de me défiler !');
});
```

`$(expression).scroll();` ne marche pas.

Redimensionnement

`resize` est déclenché lorsque l'utilisateur redimensionne la fenêtre en utilisant les poignées par exemple, mais aussi en réduisant sa fenêtre.
Pour l'utiliser sur la fenêtre principale du navigateur il faut appliquer la méthode sur `$(window)` et non `$(document)` .

Code : JavaScript

```
$(window).resize(function(){
  alert('Arrête de me redimensionner !');
});
```

`$(expression).resize();` ne marche pas.

Chargement

`load` est déclenché lorsque l'élément en question a complètement fini de se charger : la page, les cadres, les `iframes`, les objets mais surtout **les images**.
La méthode doit être définie avant que l'élément en question soit chargé sinon la fonction de retour ne sera pas appelée.

Code : JavaScript

```
$('.header_gauche img').load(function(){
  $(this).attr('title','Ceci est le logo du SdZ !');
});
```

`$(expression).load();` ne marche pas.

On peut, grâce à cette méthode, vérifier que notre méthode du chapitre 3 effectuant un préchargement des images, marche :

Code : JavaScript - Essayez ce code !

```
function prechargerImages(){
  $.each(arguments,function(){
    $('<img />')
      .attr('src',this)
.load(function(){
```

```
        alert('Image chargée : '+$(this).attr('src')+' !');
        });
          });
        }

        prechargerImages(
          'logoDeMonSite.png',
          'imageSurvol.png',
          'imageClique.png');
```

Quand le document est prêt

Passer une fonction à `$()` est le raccourci de `$(document).ready`.
`ready` est un événement global, l'argument à la fonction principale importe peu (mais en général on met `document`). Il est déclenché lorsque la page est complètement prête (cf. premier chapitre).
Le premier argument de la fonction de retour est la variable qui contient tout jQuery (`jQuery` ou `$`). `this` dans la fonction anonyme est la variable `document`.

Code : JavaScript

```
$(document).ready(function(){
  alert('Le document est prêt !');
});
```

`$(expression).ready();` ne marche pas.

Départ

`unload` est déclenché lorsque l'utilisateur quitte la page. Vous ne pouvez pas faire grand chose à ce stade là au niveau de l'utilisateur à part envoyer une alerte.

Code : JavaScript

```
$(document).unload(function(){
  alert('Au revoir et à bientôt !');
});
```

`$(expression).unload();` ne marche pas.

Erreurs

`error` est déclenché lorsqu'une erreur se produit : soit lors de l'exécution d'un script JavaScript, soit par une image dont l'adresse est mauvaise, ou alors qu'une image est corrompue.

Code : JavaScript

```
$('.header_gauche img').error(function(){
  alert("Le logo du SdZ ne s'est pas chargé !");
});

$(document).error(function(){
  alert("Une erreur s'est produite !");
});
```

Code : JavaScript

```
// Déclenche l'action du navigateur par défaut
// ainsi que l'action que vous avez définie.
$('p').error();
```

Raccourcis jQuery

Mix entre mouseover et mouseout

`hover()` est une méthode permettant d'appeler une fonction quand la souris va au-dessus de l'élément en question (**mouseover**) et quand elle s'en va (**mouseout**).
C'est donc un petit raccourci de **mouseover** (première fonction passée en paramètre) et de **mouseout** (seconde fonction passée en paramètre).

Code : JavaScript - Essayez ce code !

```
// Soulignage d'un lien quand la souris passe dessus.
$('a').hover(function(){
  $(this).css('text-decoration','underline');
},function(){
  $(this).css('text-decoration','none');
});
```

Rafales de clics

`toggle()` est une méthode qui prend autant de fonctions en paramètre qu'on veut ; elles seront appelées successivement à chaque fois que l'élément est cliqué et lorsque la dernière est appelée, la suivante sera la première.

Code : JavaScript - Essayez ce code !

```
$('button').toggle(function(){
  $(this).css('background-color','#ff0');
},function(){
  $(this).css('background-color','#f0f');
},function(){
  $(this).css('background-color','#0ff');
},function(){
  $(this).css('background-color','#00f');
});
```

La couleur d'arrière-plan d'un bouton qu'on clique sera successivement jaune, puis magenta, puis cyan et enfin bleu.

Créer ses animations

`animate()` est une méthode qui permet d'**animer le style CSS de vos éléments au cours du temps**. Plus on voudra définir de **manière précise le déroulement de notre animation**, plus il faudra donner des informations à cette méthode. De ce fait, elle peut s'appeler de deux façons différentes.

Méthode classique : plein d'arguments

`animate(style, [duration], [easing], [complete])` peut s'appeler avec ces arguments.
Voici une liste les décrivant, avec en premier le nom anglais de la variable, puis entre parenthèses et en italique son type, puis entre parenthèses sa "traduction" en français, et pour finir sa description :

1. `style` (*objet contenant des couples attribut/valeur CSS*) : **le style de l'élément** à la fin de l'animation.

2. `duration` (*entier* ou *chaîne de caractères*) (durée) : un entier positif qui est le nombre de millisecondes représentant **la durée de l'animation**, ou alors "`slow`" (600 millisecondes), "`normal`" (400 millisecondes) qui est la valeur par défaut, ou enfin "`fast`" (200 millisecondes).

3. `easing` (*chaîne de caractères*) (évolution) : détermine la façon dont les propriétés vont **changer au cours du temps** : "`swing`" par défaut, il y a aussi "`linear`".

4. `complete` (*fonction*) (fonction de retour) : une fonction (peut bien sûr être anonyme ou une variable) qui va être **appelée quand l'animation d'un élément est finie** . Elle va être appelée autant de fois qu'il y a d'éléments concernés par l'animation.

Seul le premier argument est obligatoire, les autres sont **facultatifs** (c'est ce que signifient les crochets).
Ainsi, on peut appeler `animate()` avec le style et la durée, ou avec le style et la fonction de retour, ou encore avec le style, l'évolution et la fonction de retour. Bref, si un argument facultatif n'est pas donné, on peut quand même donner le suivant dans la liste.

Code : JavaScript - Essayez ce code !

```
// animation avec juste le style
$('p')
  .css('width','400px')
  .animate({
    width : '500px'
  });

// animation avec le style, la durée et l'évolution
$('p span').animate({
    padding : '50px'
  , opacity : 0.4
},'slow','linear');

// animation avec le style, la durée et la fonction de retour
$('p strong').animate({
    fontSize : '2em'
  , paddingLeft : '50px'
},2000,function(){
  alert('fini ! ');
});
```

Méthode plus complète : un objet comme second argument

Une autre façon d'appeler la fonction est d'avoir en **second argument un objet** : les propriétés sont `duration` (durée), `easing` (évolution), `complete` (fonction de retour) et deux autres, qui ne sont **disponibles qu'avec cette manière** d'appeler `animate()` :

- `step` (*fonction*) (fonction étape par étape) : fonction qui sera appelée **à chaque étape de l'animation**, autant de fois

qu'il y a d'attributs de style et d'éléments concernés.

- `queue` (*booléen*) (empiler les animations) : détermine si la prochaine animation sur les éléments concernés **devra attendre que la première soit finie** ou non. Par défaut à *true*. Si à *false*, les animations se déroulent **en même temps**. C'est-à-dire que les attributs de style des éléments touchés par plusieurs animations **évolueront chacun de leur côté**, selon l'animation qui leur sera affectée.

Tous les attributs de l'objet en question sont bien sûr facultatifs.

Code : JavaScript - Essayez ce code !

```
$('div.rouge')
  .css('top','0px')
  .animate({
    top : '448px'
  },{
      duration : 2000
    , easing   : 'linear'
    , queue    : false
  });

$('div.bleu')
  .css('left','0px')
  .animate({
    left : '80%'
  },{
      duration : 2500
    , queue    : true // ici peu importe sa valeur
    , complete : function(){
      $(this).append('<br />finie !');
    }
  });
```

Dans la première animation, on a mis `queue` à *false*, donc **toutes les animations rajoutées après se déroulent en même temps**. D'où le fait que dans le second appel à `animate()`, peu importe la valeur qu'on affecte à `queue`.

Ajouts sur le style CSS

Les attributs CSS passés en argument de `animate()` présentent quelques améliorations.

Valeurs relatives

Chose intéressante dans les attributs CSS : **on peut utiliser des valeurs relatives** !
Je m'explique : au lieu de mettre `left: '45px'`, on peut écrire `left: '+=40px'`, ce qui signifie « ajouter 40 pixels ».
Le préfixe += ajoute donc et le préfixe -= enlève.

Code : JavaScript - Essayez ce code !

```
$('#plus').click(function(){
  $('#rectangle').animate({
    width : '+=32px'
  },1000,'linear');
});

$('#moins').click(function(){
  $('#rectangle').animate({
    width : '-=32px'
  },1000);
});
```

Animations des ascenseurs

Autre chose pratique : on peut faire des animations avec les ascenseurs des éléments ! En effet, il suffit de spécifier un attribut `scrollTop` et / ou `scrollLeft` afin de **faire défiler le contenu de l'élément**.
Ceci ne marche bien sûr qu'avec des éléments qui ont un ascenseur et qui peuvent défiler jusqu'au nombre de pixels désiré.

Code : JavaScript - Essayez ce code !

```
// un défilement vertical doux de 800px en 5 secondes
// (si il y a 800 pixels à défiler, sinon s'arrête avant)
$('body').animate({
  scrollTop : '800px'
},5000);
```

Maintenant, nous allons voir quelques précisions sur les animations, qui répondront (je l'espère) à vos interrogations .

Animations des couleurs

On ne peut pas faire varier les couleurs avec jQuery (exemple `.animate({color: '#f00'})`).
Pour faire joujou avec les couleurs, il faudra inclure le plugin jQuery UI (pas tout le plugin, juste le cœur : `effects.core.js`) qu'on étudiera plus tard ou le plugin color !

Évolution d'une animation

L'évolution de l'animation détermine comment **les attributs de styles évoluent au cours du temps**.

`linear` et `swing` sont donnés par défaut

Voici leurs caractéristiques :

1. pour `linear` : les attributs évoluent **proportionnellement par rapport au temps**,
2. et pour `swing` : **les attributs démarrent de façon moins brusque** que `linear` pendant la première moitié de l'animation, **puis ils arrivent aussi de façon moins brusque** vers leurs valeurs finales. L'**animation est donc un peu plus douce** (une des traductions d'easing d'ailleurs) et plus agréable visuellement.

J'ai traduis easing de l'anglais par évolution, mais le terme d'« accélération » (trouvé sur Alsacréations, mais on dit aussi ce mot en anglais en jQuery pour parler d'easing) peut aussi vous donner une idée de ce que c'est.

Observer leurs différences

Pour être un peu plus rigoureux et précis, si on trace la courbe de la valeurs des attributs en fonction du temps, on obtient **une droite** pour `linear` et **une demi-période de sinusoïde** (fonction cos ou sin) pour `swing`.

Sur ces graphiques :

- le point `(0,0)` correspond à la valeur initial de l'attribut en question au début de l'animation ;
- et le point `(1,1)` sa valeur final en fin d'animation.

Les abscisses correspondent au temps, tandis que les ordonnées correspondent à la valeur de l'attribut qui change progressivement vers sa valeur finale.
Voici le lien vers le graphique contenant ces deux courbes (pour pouvoir mieux comparer).

Vous pouvez aussi observer la différence entre `linear` et `swing` avec ce code :

Code : JavaScript - Essayez ce code !

```
$('#linear')
  .css('width','0px')
  .animate({
    width : '90%'
  },4000,'linear');


$('#swing')
  .css('width','0px')
  .animate({
    width : '90%'
  },4000,'swing');
```

Ajouts

Des plugins sont nécessaires afin d'en rajouter d'autres, par exemple jQuery UI (qui sera étudié plus tard).

En attendant, vous pouvez regardez sur ces sites ce que vous pourrez avoir plus tard grâce à des plugins : http://hosted.zeh.com.br/tweener/docs/ [...] nsitions.html et http://www.robertpenner.com/easing/easing_demo.html.

Étapes de l'animation

Petite précision sur la fonction étape par étape présente quand on passe un objet à `animate()`. Elle est appelée à chaque fois qu'**un élément modifie un de ses attributs**. Notez bien : elle est appelée **une fois par élément** et **une fois par attribut**.

Par exemple, si une animation change la taille des bordures et la taille des marges (**deux attributs différents**) et **concerne cinq éléments**, elle va être appelée `5*2=10` fois à chaque fois que la fonction `animate()` **avance d'un cran** en direction du style CSS final à obtenir.

Faites attention à cette fonction car **elle est souvent appelée énormément de fois** (plusieurs centaines), selon la fluidité de l'animation.

Arguments de la fonction de retour

Le premier argument de cette fonction est **la valeur de l'attribut** en question. Supposons que les marges varient entre `1px` et `3px` et que l'animation dure `1` seconde. Au bout de environ (tout n'est pas parfait) `500` millisecondes, le premier argument de la fonction qui concerne les marges sera environ de `2` (après il y a les fonctions qui concernent la taille des bordures !).

Code : JavaScript - Essayez ce code !

```
$('a')
  .css('letter-spacing','0px')
  .animate({
    letterSpacing: '10px' // Ajouter 10 pixels
  },{
    duration: 3000, // Dure 3 secondes.
    step: function( i ){ // i part de 0 et arrive à 10
      $('body').append( '<br />' + i + ' : ' + $(this).text() );
    }
  });
```

Animation temporelle : durée définie ou non

Il y a une chose à savoir sur les animations jQuery : **la durée de l'animation est toujours définie** (sinon c'est "`normal`", soit 400 millisecondes). Une animation ne correspond pas à l'ajout progressif de la valeur d'un attribut (par exemple la hauteur

d'une boîte pour la dérouler), mais **le changement de cet attribut en un temps donné** .

Pour continuer avec l'exemple de la boîte, plus celle-ci est grande et plus **l'animation sera rapide** pour le même temps. Au contraire, plus la boîte sera petite plus **la vitesse de l'animation diminuera**.

Réaliser une animation « progressive »

Si vous voulez réaliser une animation « de progression », il faut alors appeler `animate()` (ou les effets que nous allons voir plus tard) en définissant **la durée de l'animation en fonction de la valeur de l'attribut** en question.

Ainsi, si vous voulez dérouler votre boîte d'une vitesse de 150 pixels par seconde (soit 0,15 pixels par millisecondes), il faudra que la durée de l'animation soit égale à la différence de hauteur (réalisée à la fin de l'animation) de la boîte divisée par la vitesse (ici `0.15`).

Vous pouvez aussi programmer cela en JavaScript de façon classique, en utilisant (par exemple) `setInterval()`.

Exemple avec un embryon de menu

Voilà le cas général pour créer une animation progressive :

Code : JavaScript - Essayez ce code !

```
// cas général
var valeurDebut = elem.css( 'valeur' );

element.animate({
  valeur : valeurFin
}, Math.abs( valeurDebut - valeurFin) / vitesse );
// il faut absolument que la durée soit positive
// si vous ne savez pas la plus grande valeur entre valeurDebut et
valeurFin
// utilisez simplement une valeur absolue
```

Cliquez sur « Essayez ce code ! » pour avoir un exemple concret d'un commencement de menu en jQuery. Vous pourrez y observer **la différence entre animation classique et progressive** en cliquant sur les boutons.

Voilà, c'était pas très important, pas obligatoire, mais ça prend cinq lignes et ca peut pas faire de mal .

Évolution par attribut

Une des nouveautés de la version 1.4 est de pouvoir **attribuer une évolution à chaque attribut CSS**. Les éléments dont l'évolution n'est pas précisée utiliseront **l'évolution précisée dans les arguments**, sinon si elle n'est pas spécifié, celle par défaut.
Ceci peut se réaliser de deux façons :

Dans le style (en premier argument)

L'objet passé en premier argument **contient des couples attributs CSS / valeurs** décrivant le style final de l'élément.
Au lieu de mettre directement la valeur de l'attribut CSS en question, on peut aussi **donner un tableau**, dont le premier élément est la valeur, **et le second élément l'évolution** que cet attribut doit utiliser durant l'animation.

Ceci marche bien sûr avec les deux types d'appels à `animate()`.

Dans l'objet (en second argument)

Lorsque vous passez un objet en second argument, vous pouvez aussi **spécifier la propriété `specialEasing`** qui est un objet contenant des couples attributs / évolution, attribuant donc **à chaque attribut CSS sa propre évolution**.

Ceci ne marche donc qu'avec le second appel à `animate()`.

Code : JavaScript

```
$('p')
  .css('width','400px')
  .animate({
      width   : [ '800px' , 'linear' ]
    , opacity : 0.5
  }, 'linear');

$('p span').animate({
    padding  : '50px'
  , fontSize : '28px'
},{
    duration      : 'slow'
  , easing        : 'swing'
  , specialEasing : {
    fontSize : 'linear'
  }
});
```

Utiliser les effets

Les effets de jQuery sont des **animations prédéfinies simples d'utilisation** mais peu configurables qui suffisent parfois à réaliser ce que l'on cherche.
Les méthodes expliquées ici ne prennent que des arguments facultatifs (sauf la dernière) : le premier est la **durée de l'animation** et le second la **fonction de retour**.

Tous les effets utilisent l'évolution `swing`.

Visibilité

- `show()` permet d'**afficher** les éléments en question.
- `hide()` permet de **cacher** les éléments en question.
- `toggle()` permet de jongler entre la présence ou l'absence de l'élément (**si l'élément est caché, l'afficher, sinon le cacher**).

Code : JavaScript

```
// Utilisé pour afficher / cacher une balise secret.
$('blockquote.secret').toggle('normal');
```

Ces trois méthodes peuvent ne prendre aucun argument (à ce moment là, il n'y a pas d'animation et tout se fait **de façon brute**) ou les arguments classiques cités plus haut : à ce moment là, la hauteur, la largeur et l'opacité changent **de manière progressive**.

On peut utiliser `toggle()` avec les arguments classiques ou avec un seul argument, un booléen qui, à *true* affiche l'élément avec `show()`, mais qui à *false* cache l'élément avec `hide()`.

Code : JavaScript

```
$('a').toggle( true ); // affiche les liens
$('textarea').toggle( false ); // cache les zones de texte

// affiche ou cache les paragraphes selon que la checkbox soit
cochée ou non
$('p').toggle( $('#afficherParagraphes').attr( 'checked' ) );
```

Glissement

- `slideDown()` permet de **dérouler** verticalement les éléments en question.
- `slideUp()` permet d'**enrouler** verticalement les éléments en question.
- `slideToggle()` permet de jongler entre la présence ou l'absence de l'élément (**si l'élément est caché, le dérouler, sinon l'enrouler**).

Disparition

- `fadeIn()` permet de faire **apparaître** les éléments en question en modifiant l'opacité de manière progressive.
- `fadeOut()` permet de faire **disparaître** les éléments en question en modifiant l'opacité de manière progressive.
- `fadeTo()` s'utilise avec les arguments classiques, excepté que la fonction de retour est en troisième position et en seconde position se trouve un nombre compris entre 0 et 1, qui est l'**opacité** à laquelle les éléments doivent être, toujours de manière progressive.

Code : JavaScript

```
$('div').fadeIn(2000);
$('a').fadeTo('slow',0.5);
```

S'entraîner

Le meilleur moyen de voir les effets en action, c'est de faire joujou avec le code.
Rendez-vous sur JSBin.com et n'hésitez surtout pas à modifier le script à volonté pour vous entraîner !

Autres ajouts sur le style CSS dans `animate()`

Directement inspiré des effets, les valeurs des attributs CSS donnés à `animate()` peuvent aussi être "`show`", "`hide`" ou encore "`toggle`".
Ces **raccourcis pratiques** sont disponibles pour les attributs qui concernent **l'affichage graphique de l'élément**, et respectivement affichent, cachent et inversent l'affichage (si c'est affiché, le cacher, et vice-versa).

Code : JavaScript - Essayez ce code !

```
$('div.boite').animate({
    width : 'toggle'
  , height : '200px'
},5000);

$('div.boite').animate({
    width : 'toggle'
  , height : 'toggle'
},5000);
```

Cela marche pour les attributs `width`, `height` ainsi que `opacity`.

Contrôler les animations

Nous allons passer en revue dans ce sous-chapitre tout ce qui nous permet de manipuler les animations.

Sélecteur d'animations

Le sélecteur `:animated` permet de filtrer les éléments qui sont **animés au moment où la fonction s'exécute**.

Code : JavaScript - Essayez ce code !

```
// Toutes les balises div qui sont
// en ce moment animées ont la classe 'anime'.
$('div:animated').addClass('anime');

// Toutes les balises div qui ne sont pas en ce moment
// animées n'ont pas (ou plus) la classe 'anime'.
$('div:not(:animated)').removeClass('anime');
```

Désactiver les animations

Depuis la version 1.3, on peut **désactiver toutes les animations** ainsi que tous les effets avec la simple ligne `$.fx.off = true;` (ou bien sûr `jQuery.fx.off = true;`).
Ceci n'améliore pas les performances de jQuery, juste que lorsqu'on utilise un effet, l'animation n'a pas lieue, mais les éléments ont **directement leur style final** (et la fonction de retour est exécutée).

On peut réactiver les animations et les effets avec `$.fx.off = false;` .

Code : JavaScript - Essayez ce code !

```
function desactiverAnimations(){
  $.fx.off = true;
}

function activerAnimations(){
  $.fx.off = false;
}
```

Comme vous pouvez vous en apercevoir en jouant un peu avec l'exemple (appuyer plusieurs fois sur "Animer!", puis désactiver les animations), une animation qui a été programmée lorsque les animations étaient activées, marche normalement, même si les animations sont maintenant désactivées.

Fluidité des animations

Depuis la version 1.4.3, on peut changer la fluidité des animations en changeant la valeur de `$.fx.interval` (ou `jQuery.fx.interval`).
Ce nombre représente **le nombre de millisecondes à attendre entre deux changements de style CSS** dans une animation.

Il est par défaut à 13 millisecondes, soit environ 77 images par secondes (`1/0.013`).

Augmenter cet intervalle **baisse la fluidité de l'animation mais améliore les performances** (le navigateur est alors moins gourmand), tandis que le réduire améliore la fluidité.

Cela ne change en aucun cas la vitesse de l'animation ni sa durée !

Files d'attente jQuery

Voilà le point le plus important du cours concernant le contrôle des animations !

Une file d'attente (en anglais `queue`) d'un élément est **un tableau de fonctions associées à cet élément** : la première fonction du tableau est exécutée en première sur l'élément en question, lorsqu'elle a fini de s'exécuter, elle s'enlève du tableau pour **laisser place à la suivante** qui va s'exécuter.

Pour commencer, on va considérer que ces fonctions sont des animations.

Stopper les animations

La méthode `stop([ viderLaQueue, allerALaFin ])` vous permet d'**arrêter l'animation d'un élément**. Elle peut prendre aucun, un ou deux arguments :

- `viderLaQueue` (*booléen*) :
  - Si à *true* : stoppe l'animation en cours et celles à venir (qui **sont présentes dans la queue**).
  - Si à *false* : (par défaut, donc aussi quand on appel sans argument), stoppe seulement la **première animation de la queue**. Les autres animations dans la queue **commenceront tout de suite** .
- `allerALaFin` (*booléen*) :
  - Si à *true* : ne stoppe pas l'animation, mais la fait **aller à sa fin** : l'élément concerné aura donc son style final et la fonction de retour sera appelée.
  - Si à *false* (par défaut) : stoppe l'animation et ne fait rien de spécial.

A retenir :

Code : JavaScript - Pour résumer

```
// Ces trois lignes sont équivalentes :
$('div:animated').stop();
$('div:animated').stop(false);
$('div:animated').stop(false,false);
// Ces deux lignes sont équivalentes :
$('div:animated').stop(true);
$('div:animated').stop(true,false);

// Donc les seules vraies utilisations de stop() sont :
$('div:animated').stop(); // Stopper la première animation.
$('div:animated').stop(true); // Stopper toutes les animations.
$('div:animated').stop(false,true); // Stopper la première animation
et la fait aller jusqu'à sa fin.
$('div:animated').stop(true,true); // Stopper toutes les animations
à venir et fait aller la première jusqu'à sa fin.
```

Notez **le dernier cas un peu ambigu** : on aurait pu croire que toutes les animations à venir sur un élément se verraient aller jusqu'à leurs fins (style CSS et appel de la fonction de retour), mais elles sont en fait juste annulées. Seule l'animation en cours se voit aller jusqu'à sa fin.

Code : JavaScript - Essayez ce code !

```
function stop1(){
  $('#contenu div:animated').stop();
}

function stop2(){
  $('#contenu div:animated').stop( true );
}
```

```
function stop3(){
  $('#contenu div:animated').stop( false , true );
}

function stop4(){
  $('#contenu div:animated').stop( true , true );
}
```

Le dernier cas ambigu peut se voir facilement avec l'exemple : il suffit d'animer plusieurs fois un bloc, puis d'appuyer sur le quatrième bouton.
Le bloc va aller directement à l'endroit où il est censé aller, et son nombre (affiché en dessous du bouton) va **décrémenter une seule fois** (il ne sera donc pas égal à 0).

Manipuler les files d'attente

Maintenant on va étudier le cas général : même si étroitement lié, ce qu'on va dire sort un peu du cadre des animations.
On peut **manipuler les files d'attente** avec deux méthodes : `queue()` ainsi que `dequeue()`.

queue()

Les utilisations de `queue()` sont :

- `queue()` appelée sans argument **renvoie la file d'attente** (sous forme d'un tableau de fonctions) des éléments concernés.
- `queue()` peut prendre en argument une fonction qui va **s'ajouter à la queue**. Cette fonction ne doit pas forcément être une animation, vous pouvez **exécuter n'importe quelle action**.
- `queue()` peut aussi prendre en argument un tableau de fonctions qui viendra **remplacer la file d'attente existante**.

dequeue()

La méthode `dequeue()` **doit être utilisée à la fin de toute fonction** présente dans la queue, et ceci afin que les autres fonctions de la queue s'exécutent.

Pourquoi ?

`dequeue()` supprime de la queue la fonction finie (en cours), puis exécute la suivante dans la queue (on passe à la suivante).
Son utilisation à l'intérieur d'une fonction de retour est tout simplement `$(this).dequeue();` .

Par exemple, dans une animation, avant d'appeler la fonction de retour définie, jQuery utilise cette méthode.

Exemple :

Code : JavaScript - Essayez ce code !

```
$('#carre')
  .text('commencé !')
  .animate({
        top  : '100px'
      , left : '128px'
    } , 2000
    , function(){
      alert('fini !');
      $(this).text('fini !')
  })
  // ajout à la queue d'une fonction
```

```
        .queue(function(){
          alert('blabla !');
          $(this).dequeue();
        })
        .css('color','#f00')
        .queue(function(){
          alert('blabla 2 !');
        });
```

Voici **dans l'ordre** ce que fait le code :

1. changement du contenu textuel,
2. ajout d'une animation avec fonction de retour "fini !",
3. ajout d'une fonction à la queue "blabla !", avec `dequeue()` à la fin,
4. changement de la couleur de la police,
5. ajout d'une autre fonction à la queue "blabla 2 !", sans `dequeue()`.

Voici maintenant ce qui se passe **dans le temps** :

1. changement du contenu textuel,
2. commencement de l'animation,
3. changement de la couleur de la police,
4. ...progression de l'animation...
5. fin de l'animation : **`dequeue()` est exécutée**..
6. ..ce qui a pour effet d'exécuter la première fonction ajoutée à la queue.
   "blabla !" s'affiche. `dequeue()` est exécutée..
7. ..donc la seconde fonction est appelée.
   "blabla 2 !" s'affiche.
8. la fonction de retour de l'animation est appelée.
   "fini !" s'affiche.

A la ligne 14 : ne pas mettre `$(this).dequeue();` empêche "blabla 2 !" de s'afficher. Mais cela n'empêche pas "fini !" de s'afficher !
En effet, à l'intérieur d'`animate()`, il y a un `$(this).dequeue();` (je l'ai déjà dit deux fois, mais je le répète ), suivie d'un appel à la fonction de retour.

Note : si on avait passé un second objet à `animate()` avec `queue` à *false*, il n'y aurait pas eu de `$(this).dequeue();` avant d'appeler la fonction de retour.
Les fonctions ajoutées par la suite avec `queue()` se seraient quand même déclenchées, mais **pendant le déroulement de l'animation** (et pas à sa fin).

Plusieurs files d'attente différentes

Vous verrez dans la seconde partie qu'il existe d'autres files d'attente que celle pour les animations.
En attendant, dites-vous que la file d'attente que vous manipulez pour l'instant s'appelle `fx`.

`clearQueue()`

La méthode `clearQueue()`, introduite depuis la version 1.4, est un raccourci de `stop(true)`, mais utilisable pour d'autres files d'attente que `fx`.
Elle sera vue plus en détail dans la seconde partie.

Retarder une file d'attente

Créée spécialement pour les effets et animations, et introduite depuis la version 1.4, `delay()` est une méthode qui permet de **retarder pendant un temps donné une file d'attente** d'un élément.
Le premier argument est le nombre en millisecondes avant que **l'élément suivant dans la file d'attente ne soit exécuté** .
Le second argument désigne la file d'attente (par défaut `fx`).

Code : JavaScript

```
$('#logo')
   .slideDown(500)
   .delay(2000)
   .fadeOut(800)
;
```

TP : Menu déroulant

Votre but

Vous devez réaliser un **menu déroulant à deux niveaux**.
Le premier niveau sera disposé à l'horizontale. Quand on survolera un des éléments, le second niveau (s'il y en a un) se déroulera **verticalement vers le bas** (animation de la hauteur).

Code de base

Code : HTML

```
<ul id="menu">
  <li class="premier" id="premierElement">
    <a>Élément du menu sans sous-menu</a>
  </li>
  <li class="premier" id="secondElement">
    <a class="">Élément du menu avec sous-menu</a>
    <ul class="sousmenu">
      <li><a>Élément du sous-menu</a></li>
      <li><a>Élément du sous-men</a></li>
    </ul>
  </li>
</ul>
```

La **balise principale de notre menu** est une balise liste non-ordonnée `<ul>` .

A l'intérieur se trouvent des balises élément de liste `<li>` (affectées d'une classe `premier`) qui contiennent **obligatoirement un lien** `<a>` et, s'il y a un sous-menu, encore une liste (`<ul>` de classe `sousmenu`, et à l'intérieur des `<li>`).

Il faut bien sûr **préciser un attribut `href`** aux liens du menu (non fait dans le bout de code ci-dessus).

Chacun des éléments `<li>` de la liste principale sont **affectés d'un attribut d'identification**.

Aides

Style CSS

Pour mettre la première liste à l'horizontale, il suffit que ses éléments `<li>` aient `float: left;` .
Pour que les sous-menus **modifient la hauteur de leur élément parent** (`<li>`) et qu'ils soient cachés en fonction de sa hauteur, il ne faut pas qu'ils aient `position: absolute;` .
Pour qu'on ne puisse pas voir le sous-menu (caché) qui déborde, il faut que les éléments `<li>` du menu principale aient `overflow: hidden;` .

Hauteur du sous-menu

Un sous-menu `<ul>` va changer la hauteur de son élément parent `<li>` (élément de la liste principale).
Il va donc falloir **enregistrer la hauteur** de chaque sous-menu (c'est pour cela qu'il y a des `id` spécifiés), pour la ressortir lors

de l'animation de déroulement.

On utilise alors un objet, qui associe à chaque `id` d'un `<li>` , sa hauteur.

Le résultat ressemblera à l'exemple sur les animations progressives de "Créer ses animations".

Empêcher l'enchaînement d'animations

Une fois votre menu fini, vous rencontrerez sûrement le problème suivant : lorsque qu'on passe la souris très rapidement et plusieurs fois sur le menu, un élément déclenche une animation, l'animation n'aura pas le temps de se terminer que la souris sera déjà sur un autre élément.. ce qui va déclencher **encore une autre animation**.

Au final, votre menu se met à bouger énormément, alors que la souris ne se trouve plus sur le menu. Le résultat est **très inesthétique**..

La solution, c'est d'utiliser `stop()` sur les éléments avant de les animer .

Solution

Code : JavaScript - Essayez ce code !

```
var tailles = {}, tailleMax = 0, tailleCourante;

$('#menu li.premier')
  .each(function(){
    // enregistrer la hauteur du menu déroulé complètement
    tailles[ $(this).attr('id') ] = tailleCourante =
$(this).height();
    // redéfinir la hauteur (par défaut) pour cacher le menu
    // Note : juste pour ceux qui ont JavaScript activé
    // donc ceux qui n'ont pas JS activé verront le menu déroulé et
non animé
    $(this).height( 20 );

    // enregistrer la taille maximale au fur et à mesure
    if( tailleCourante> tailleMax ){
      tailleMax = tailleCourante;
    }
    // pour ne pas déborder sur le contenu (position:relative et pas
absolute)
    $('#menu').height( tailleMax );
  })
  // la souris rentre..
  .mouseenter(function(){
    $(this).stop().animate({
      // hauteur du menu déroulé complètement
      height: tailles[ $(this).attr('id') ]
    },500);
  })
  // ..et sort
  .mouseleave(function(){
    $(this).stop().animate({
      height: '19px' // taille par défaut
    },500);
  })
;
```

Note

Ce TP est un entraînement. Le menu n'est pas parfait, entre autres parce qu'il déborde sur le contenu qui suit (donc s'il y a

énormément d'éléments dans les sous-menus, le contenu sera bien après dans la page).

Vous verrez dans la seconde partie des **méthodes indispensables** pour créer des menus mieux et radicalement plus simples. Par exemple la méthode `find()` nous aurait permis dans ce cas-là de nous affranchir d'une contrainte...

Vous pourrez plus tard (dans la seconde partie) vous amuser à créer des menus de toutes sortes, et dont le code sera, je l'espère, **nettement plus léger** que celui-ci.

Améliorations

Voici quelques améliorations que vous pourriez implémenter pour vous entraîner :

- Mieux décorer le menu : prévoir des icônes à côté du texte, ou encore attribuer des images de fond esthétiques et répétitives aux éléments par exemple ;
- Prévoir des éléments du menu qui ne sont pas des liens (donc pas forcément de balise `<a>`) ;
- Simplifier le code, en essayant de supprimer des classes par exemple ;
- Réaliser des animations progressives (cf. "Créer ses animations") pour dérouler les sous-menus (deux lignes de codes à changer) ;
- Faire un menu entièrement vertical, ou entièrement horizontal.

Vous êtes donc maintenant capables d'**animer des éléments** en utilisant les effets, et même en créant **vos propres animations**.

C'est ici que s'achève le cours de la première partie du tutoriel.

Appris dans ce chapitre : méthodes événements (`select`, `change`, `submit`, `focus`, `blur`, `keydown`, `keypress`, `keyup`, `mousedown`, `mouseup`, `click`, `dblclick`, `mouseenter`, `mouseleave`, `mouseover`, `mouseout`, `mousemove`, `scroll`, `resize`, `load`, `ready`, `unload`, `error`, `hover`, `toggle`), méthodes `animate()`, `show()`, `hide()`, `toggle()`, `slideDown()`, `slideUp()`, `slideToggle()`, `fadeIn()`, `fadeOut()`, `fadeTo()`, `fadeIn()`, sélecteur `animated`, variable `$.fx.off`, méthodes `stop()`, `queue()`, `dequeue()`.

Vous connaissez maintenant les **principales méthodes de jQuery** qui vous seront utiles à tout moment.

Partie 2 : Plus loin dans jQuery

Ici, nous finirons de voir **entièrement jQuery Core**. Vous y étudierez les événements, le parcours du document par des méthodes, l'AJAX, comment créer ses propres plugins ainsi que bien sûr tous les petits utilitaires mis à notre disposition par jQuery.

Les explications seront **un peu moins fournies en exemples** par rapport aux chapitres de la partie précédente, mais nous comptons bien sûr sur vous pour couvrir **tous les aspects d'une méthode** de façon autonome. Par contre, les QCM et les TPs seront toujours au rendez-vous !

Gérer les événements

Le JavaScript dispose d'événements. Ceux sont des appels à des fonctions lorsque qu'**une action est déclenchée** : elle peut être due à l'utilisateur ou au navigateur (c'est ce que vous avez vu dans la première partie).
Mais ces actions et ces appels à des fonctions peuvent aussi être créés artificiellement, **pour les besoins de votre programme**.

Nous allons voir dans ce chapitre comment les méthodes de jQuery, qui constituent un vrai plus, permettent **de créer et de manipuler les événements**.

Malgré tout cela, jQuery ne fait que le relais entre le JavaScript et vous : si vous voulez bien gérer les événements comme les mouvements de la souris ou les clics, **des connaissances en événements JS sont requises**.

Écouter et supprimer

Vocabulaire des événements

Quand un événement est déclenché sur un élément, et qu'il faut **associer une fonction à appeler lors de ce déclenchement**, on dit qu'on « écoute » un événement.

On lie, ou on attache (« to bind ») une fonction à un type d'événement bien déterminé sur un élément. L'élément en question est alors « écouté » sur ce type d'événement.
On supprime, on délie ou on détache (« to unbind ») une fonction à un type d'événement sur un élément lorsqu'on enlève l'appel à la fonction lors du déclenchement de l'événement sur cet élément.

La fonction de retour est appelée un « écouteur » d'événement (`addEventListener()` signifie « ajouter un écouteur d'événement » !).

Écouter

Les méthodes apprises au début du chapitre sur les animations, par exemple `click()` ou `mouseenter()`, permettent de **lier un événement à une fonction** : on appelle ça un écouteur d'événement.
`bind()` permet de généraliser cela en lui passant en premier argument une chaîne de caractères représentant tous les événements (séparés par des espaces) **à lier avec la fonction passée en paramètre**.

Code : JavaScript - Essayez ce code !

```
$('p,textarea')
  .bind('mouseenter focus',function(){
    $(this).css('border-color','#222');
  })
  .bind('mouseleave blur',function(){
    $(this).css('border-color','#bbb');
  });

$('q').bind('dblclick',function(){
  // Si l'attribut auteur existe...
  if($(this).attr('auteur')){
    // ... l'afficher.
    alert("L'auteur de cette citation est "+$(this).attr('auteur')+'
!');
  }
});
```

L'avantage de cette méthode n'est peut être pas évident au début, mais vous le comprendrez vite avec les méthodes qu'on va apprendre ensuite  .

La fonction false

Depuis la version 1.4.3, au lieu de spécifier une fonction, on peut mettre un booléen *false*.
Il désigne alors la fonction qui renvoie *false*.

Code : JavaScript

```
$('a').bind('onclick',false);
// est équivalent à
$('a').bind('onclick',function(){
  return false;
});
```

Dans ce cas-là, cela supprime le comportement par défaut du navigateur qui consiste à ouvrir le lien (attribut `href` de la balise `<a>`).

Supprimer

`unbind()` permet **d'enlever l'écouteur d'événement** créé préalablement avec `bind()`.
On peut lui passer divers arguments qui effectuent sur les éléments concernés :

- aucun argument (`unbind()`) : supprime **tous les écouteurs** ;
- nom de l'événement (`unbind('click')`) : supprime tous les écouteurs **du type de l'événement** (il peut y avoir plusieurs fonctions pour un même événement) ;
- nom de l'événement et fonction (`unbind('click',fonction)`) : supprime juste **un écouteur bien précis** pour ce type d'événement.

On peut, comme avec `bind()`, spécifier plusieurs événements.

Code : JavaScript - Essayez ce code !

```
// Affichera l'adresse du lien qu'une seule fois.
$('#contenu a:not(.interne)').click(function(){
  alert('adresse du lien : '+$(this).attr('href'));
  $(this).unbind('click');
  // le lien ne sera pas suivi
  return false;
});

$('#contenu textarea')
  .focus(zoneTexte)
  .focus(function(){
    $(this).after('<br />Clic !');
  })
  .blur(function(){
    // couleur jaune clair
    $(this).css('background-color','#ff8');
  })
  // Quand on double-cliquera, enlèvera tous les événements, sauf le 'Clic !'.
  .dblclick(function(){
    $(this).after('<br />Double Clic !');
    $(this)
      .unbind('focus',zoneTexte)
      .unbind('blur');
  });

function zoneTexte(){
  // couleur magenta clair
  $(this).css('background-color','#f8f');
}

function unbindZoneTexte(){
  // enlève tout
  $('#contenu textarea').unbind();
}
```

Référence à une fonction

On peut appeler `unbind()` avec le nom de l'événement, puis **une référence à une fonction**. Notez bien la distinction entre une fonction et sa référence.

Une fonction est une variable :

Code : JavaScript

```
var maFonction = function(){
```

```
  // faire quelque chose
  alert('Blabla !');
}
```

Deux fonctions identiques (qui font la même chose) peuvent êtres déclarées séparément :

Code : JavaScript

```
var maFonction2 = function(){
  // faire quelque chose
  alert('Blabla !');
}
```

Cependant, même si le JavaScript est un langage de script, on ne va pas s'amuser à comparer des fonctions pour voir si elles sont identiques.

Une fonction est identique à une autre si l'une est une référence à l'autre :

Code : JavaScript

```
var maFonction3 = maFonction;

// maFonction et maFonction2 ne pourront jamais être égales
// elles ont été créées séparément en mémoire..
alert(maFonction == maFonction2); // false

// ..mais maFonction3 est une référence à maFonction
alert(maFonction == maFonction3); // true
```

Dans ce code, écrire le nom de la variable (sans parenthèses pour appeler la fonction) désigne la variable contenant la fonction en elle-même et non la fonction.

Ainsi pour `unbind()` il faut donner une référence à la fonction passée à `bind()`, comme dans l'exemple un peu plus au-dessus.

Enlever la fonction false

Toujours depuis la version 1.4.3, il suffit de donner *false* à `unbind()` pour enlever la fonction qui retourne *false*.

Les événements uniques

`one()` a les mêmes propriétés que `bind()` (et les mêmes arguments) à l'exception que **l'événement ne se déclenche qu'une et une seule fois**.
Tout comme avec `bind()`, on utilise `unbind()` afin de délier un événement à l'appel de la fonction.

Code : JavaScript - Essayez ce code !

```
$('a:not(.interne)').one('click',function(){
  alert('adresse du lien : '+$(this).attr('href'));
  return false;
});
```

Attacher plusieurs événements en même temps

Une des nouveautés de la version 1.4 est de pouvoir donner un objet en argument à `bind()`, contenant des **couples nom de l'événement / fonction de retour attachée**.

Par exemple :

Code : JavaScript

```
$('h1')
  .bind({
      mouseenter: function(){
        $(this).css('color','#000');
      }
    , mouseleave: function(){
        $(this).css('color','#00f');
      }
    , click: function(){
        $(this).css('color','#ff0');
      }
  });
```

Déclencher

Le déclenchement d'un événement n'est pas réservé à l'utilisateur : vous pouvez **déclencher virtuellement des événements sur des éléments sans avoir pour autant l'aval de l'utilisateur**.
Il se peut même que votre code n'ait rien à voir avec l'interface utilisateur. Les événements sont une manière de programmer.

trigger()

`trigger()` permet de **déclencher un événement**, produisant l'action par défaut du navigateur ainsi que **celle que vous avez définie**.
Les méthodes utilisées plus haut qui pouvaient être appelées sans argument sont donc **le raccourci d'un appel à `trigger()`**

Par exemple `$(expression).submit()` est équivalent à `$(expression).trigger('submit')` .

Code : JavaScript - Essayez ce code !

```
$(':text')
  .focus(function(){
    $(this).trigger('click');
  })
  .click(function(){
    $(this).after(' clic !');
  });
```

triggerHandler()

`triggerHandler()` agit presque de la même manière que `trigger()` sauf qu'elle **n'actionne pas le mécanisme par défaut du navigateur** (par exemple quand un champ gagne le « focus », le texte du champ est sélectionné et on peut écrire), mais appelle seulement la fonction qu'on a définie.

Autre particularité, `triggerHandler()` ne fait son effet **que sur le premier élément** de tous les éléments contenus dans l'objet jQuery.

Code : JavaScript - Essayez ce code !

```
$('#contenu :text')
  .mouseover(function(){
    // Appelle la fonction que vous avez définie
    // mais ne met pas le curseur de texte.
    $(this).triggerHandler('focus');
  })
  .focus(function(){
    $(this).after(' focus !');
  });
```

Résumé des événements déclenchables ou non

Événements déclenchables :

- blur ;
- change ;
- click ;
- dblclick ;
- error ;
- focus ;

- keydown ;
- keypress;
- keyup ;
- select ;
- submit.

Événements non déclenchables :

- load ;
- mousedown ;
- mouseout ;
- mouseover ;
- mousemove ;
- mouseup ;
- resize ;
- unload.

Complément d'informations sur `bind()` : nos propres événements !

Autre aspect des événements, qui renforce l'intérêt des méthodes apprises ici (parce que dans les cas simples on peut se débrouiller sans elles), est le fait qu'on peut **créer nos propres événements** :

Code : JavaScript - Essayez ce code !

```
$('a')
   .bind('lien',function(){
     alert('Vous aller vers '+$(this).attr('href')+' !');
   })
   .click(function(){
     if(!$(this).hasClass('interne')){
       $(this).trigger('lien');
     }
   });
```

Bien évidemment, cela va de pair avec l'utilisation de trigger().
Notez que ce code peut tout aussi bien s'écrire comme cela depuis la version 1.4 :

Code : JavaScript - Essayez ce code !

```
$('a').bind({
     lien : function(){
       alert('Vous aller vers '+$(this).attr('href')+' !');
       }
   , click : function(){
       if(!$(this).hasClass('interne')){
         $(this).trigger('lien');
     }
});
```

Passer des arguments aux écouteurs

On peut donner des informations à `trigger()`, `triggerHandler()`, `bind()`, `one()` ainsi qu'aux méthodes événements (`onclick()` par exemple), des données qui seront alors accessibles dans les écouteurs (déclenchées lors d'un événement).

On distingue deux cas :

1. Le premier, simple, correspond à ce qu'on pouvait faire avec `each()` (cf. fin de "Modifier la structure").

Lors de la création d'un événement sur un élément, on peut **transmettre des données qui seront exploitables dans l'écouteur** ;
2. Le second cas, nettement plus pratique, permet de **déclencher un événement d'une manière plus précise**.
Ainsi dans l'écouteur, l'exploitation de ces données permettra concrètement de savoir dans quelles circonstances on a déclenché l'événement.

Voici comment on fait :

avec `bind()`, `one()` et les méthodes événements

- Au lieu d'appeler `bind()` et `one()` avec deux arguments (le nom de l'événement, puis la fonction), on peut rajouter un troisième argument, `donnees`, qui vient **se glisser entre le premier et le deuxième** :
`bind('mouseover','salut',maFonction)` ;
- Au lieu d'appeler les méthodes événements (les méthodes apprises au début du chapitre sur les animations, `mouseover()` par exemple) avec en premier argument la fonction de retour, on **peut insérer juste avant** (donc la fonction de retour est deuxième) un argument `donnees` : `mouseover('salut',maFonction)` .

Le second cas est apparu avec **la version 1.4.3**.
Si vous n'avez pas cette version, utilisez `elems.bind('evenement',donnees,fonction)`, qui est équivalent à `elems.evenement(donnees,fonction)`.

Le premier argument de la fonction appelée lors du déclenchement d'un événement, **L'objet événement**, (que l'on va étudier plus tard) **contient un membre `data`** égale à `donnees` :

Code : JavaScript

```
function maFonction(e){
  // salut !
  alert( e.data + ' !' );
}
```

Bien évidemment, `donnees` peut être de n'importe quel type, le plus pratique étant un objet.

Exemple :

Code : JavaScript - Essayez ce code !

```
function alerte(e){
  alert( 'Vous êtes sur ' + e.data.type + ' !' );
}

$('a').mouseenter({
  type : 'un lien'
},alerte);

$('img').bind('mouseenter',{
  type : 'une image'
},alerte);
```

avec `trigger()` et `triggerHandler()`

Une autre particularité de `trigger()` et de `triggerHandler()` : vous pouvez **passer des arguments** aux fonctions de retour passées à `bind()` / `one()` / `live()` !
Le second argument de ces deux méthodes est un tableau qui représente **les arguments à passer aux fonctions de retour**.

Code : JavaScript - Essayez ce code !

```
$('a').bind('afficherInfosLien',function(e,url,description,classe){
  alert( 'URL : ' + url
       + '\nDescription :' + description + '\n'
       + ( classe ? ( 'classe du lien : ' + classe ) : '' )
       );
});

$('a').click(function(){
  $(this).trigger('afficherInfosLien',[
    $(this).attr('href'),
    $(this).text(),
    $(this).attr('class')
  ]);
  return false;
});
```

J'espère que les deux exemples vous ont bien fait comprendre comment cela s'utilisait, même si **l'utilité n'est pas évidente au premier abord**.

Événements vivants

Méthode `live()`

`live()` a les mêmes propriétés que `bind()` à l'exception d'une subtile différence : si **des éléments correspondant à la requête jQuery** (la variable `selector` de l'objet jQuery) venaient à se créer plus tard, ils se verraient eux aussi **liés au même événement**, à la même fonction.

Donc :

- `$(this).live(...)` n'a aucun intérêt : l'événement ne sera pas vivant,
- si vous clonez un élément, même si l'argument n'est pas à *true*, **les événements créés avec `live()` seront copiés**.

De toute évidence, des exemples s'imposent :

Code : JavaScript - Essayez ce code !

```
var i = 1;

$('#contenu :text')
  .live('mouseover',function(){
    $(this).css('background-color','#fbf');
  })
  .live('mouseout',function(){
    $(this).css('background-color','#fff');
  })
  .live('click',function(){
    $(this)
      // Cloner sans copier les événements mais
      // qui copie les événements live quand même.
      .clone()
      .val(i++)
      .insertAfter(this);
  });
```

Dans les versions ultérieures à la 1.4, `live()` ne marche pas avec les événements suivants : blur, focus, mouseenter, mouseleave, change, submit.

Méthode `die()`

`die()` est à `live()` ce que `unbind()` est à `bind()`

`die()` délie tous les évènements liés avec `live()`, donc ceux sur les éléments semblables !
`unbind()` et `die()` sont donc bien différents.

Code : JavaScript - Essayez ce code !

```
var i = 1;

$('#contenu :text')
  .live('mouseover',function(){
    $(this).css('background-color','#fbf');
  })
  .live('mouseout',function(){
    $(this).css('background-color','#fff');
  })
  .live('click',function(){
```

```
        $(this)
          .clone()
          .val(i++)
          .insertAfter(this);
      });

    function die(){
      $('#contenu :text').die();
    }
```

`live()` étant **une méthode assez coûteuse**, utilisez-là avec parcimonie !

Espaces de noms

Les **espaces de noms** (`namespaces` en anglais) permettent de créer des sortes de **classes pour les événements**.
Il suffit de mettre **un point** après le nom de l'événement **lors d'un appel** à `bind()`, `unbind()`, `one()`, `trigger()`, `triggerHandler()`, `live()` ainsi que `die()`.

On peut alors créer **plusieurs appels à différentes fonctions pour un même type d'événement**, et mieux manipuler ces différents écouteurs d'événements.
L'utilité de ces méthodes est redoublée (contrairement à juste utiliser `click()`, `mouseover()` etc..) ; l'usage de `trigger()` et de `triggerHandler()` est alors utile pour déclencher nos événements avec un espace de nom bien particulier.

Quelques règles

À noter que :

- On peut **définir plusieurs noms en même temps**, par exemple `bind('focus.hello.world')` qui définit `focus.hello` et `focus.world` :
  Code : JavaScript - Essayez ce code !

```
$('input').bind('focus.hello.world',function(){ /* action ...
*/ });


$('#focusHello').click(function(){
  $('input').trigger('focus.hello');
  // action !
});

$('#focusWorld').click(function(){
  $('input').trigger('focus.world');
  // action !
});
```

- Mais **on ne peut pas déclencher plusieurs noms en même temps**, par exemple `trigger('focus.hello.world')` qui ne déclenche ni `focus.hello`, ni `focus.world` :
  Code : JavaScript - Essayez ce code !

```
$('input').bind('focus.hello',function(){ /* action 1 */ });
$('input').bind('focus.world',function(){ /* action 2 */ });


$('#focusHelloWorld').click(function(){
  $('input').trigger('focus.hello.world');
  // rien du tout !
});
```

  Par contre ceci marche avec `unbind()` !

- Si on ne **précise pas de noms**, tous les noms sont déclenchés, par exemple `trigger('focus')` qui déclenche `focus.hello` et `focus.world` :
  Code : JavaScript - Essayez ce code !

```
$('input').bind('focus.hello',function(){ /* action 1 */ });
$('input').bind('focus.world',function(){ /* action 2 */ });


$('#focus').click(function(){
  $('input').trigger('focus');
  // action 1 et 2 !
```

```
});
```

- Si on **précise un nom sans type d'événement, rien n'est déclenché**, par exemple `trigger('.salut')` qui ne déclenche ni `focus.salut` ni `blur.salut` :

Code : JavaScript - Essayez ce code !

```
$('input').bind('focus.salut',function(){ /* action 1 */ });
$('input').bind('blur.salut',function(){ /* action 2 */ });


$('#salut').click(function(){
  $('input').trigger('.salut');
  // rien du tout !
});
```

- Vous pouvez bien sûr utiliser **vos propres types événements avec des noms**, par exemple `trigger('hello.world')` :

Code : JavaScript - Essayez ce code !

```
$('input').bind('hello.world',function(){ /* action ... */ });


$('#hello').click(function(){
  $('input').trigger('hello');
  // action !
});

$('#helloWorld').click(function(){
  $('input').trigger('hello');
  // action !
});
```

Utilité ?

Pour ceux que ça intéresse, je vais montrer rapidement en quoi les espaces de noms ne sont pas indispensables. Ce court passage n'est pas obligatoire  .

Situation et problème

Imaginez que vous avez **deux événements différents** qui attribuent au `click` une fonction :

Code : JavaScript

```
$('input').click(function(){
  alert('Vous avez cliqué sur un élément répondant au sélecteur
"input"');
});

$(':text').click(function(){
  alert('Vous avez cliqué sur un élément répondant au sélecteur
":text"');
});
```

Maintenant, vous voulez pouvoir à tout moment **retirer un des événements**. Il faudrait alors **contenir les deux fonctions dans des variables**, comme ceci :

Code : JavaScript

```
function clickTexte(){
      alert('Vous avez cliqué sur un élément répondant au sélecteur
"textarea,:text"');
}

function clickChamp(){
  alert('Vous avez cliqué sur un élément répondant au sélecteur
":text"');
}


// premier événement
$('textarea,:text').click(clickTexte);
// second événement
$(':text').click(clickChamp);

// enlever le premier événement
$('textarea,:text').unbind('click',clickTexte);
// enlever le second événement
$(':text').unbind('click',clickChamp);
```

Vous voyez qu'il n'y a pas vraiment de problème, il faut juste **stocker les fonctions** (étant donné qu'on peut attribuer plusieurs fonctions pour un événement). On n'a pas utilisé ici de fonctions anonymes.

Déclarer des fonctions sur le tas

Notez que ce code marche :

Code : JavaScript

```
// variable indéfinie
var nombre;
// affecte 8 à nombre, puis affiche 8
alert( nombre = 8 );
// affiche 8
alert( nombre );
```

Donc si vous voulez absolument garder la technique consistant à **déclarer la fonction sur le tas** (et ainsi garder la description de la **fonction à côté de son attachement à l'événement**), vous pouvez le faire :

Code : JavaScript

```
var clickTexte, clickChamp;
// premier événement
$('textarea,:text').click(clickTexte = function(){
      alert('Vous avez cliqué sur un élément répondant au sélecteur
"textarea,:text"');
});
// second événement
$(':text').click(clickChamp = function(){
  alert('Vous avez cliqué sur un élément répondant au sélecteur
":text"');
});

// enlever le premier événement
$('textarea,:text').unbind('click',clickTexte);
// enlever le second événement
$(':text').unbind('click',clickChamp);
```

Et donc ?

Tout ça pour quoi ?

Je voulais juste illustrer le fait que **les espaces de noms ne sont pas indispensables** : on peut très bien contourner les problèmes qu'ils résolvent. Mais s'ils ont été créés, c'est justement pour contourner proprement tous ces problèmes ! 

L'utilité des espaces de noms n'est pas évidente au premier abord, mais s'il vous arrive d'en avoir besoin (quand on commence à **manipuler beaucoup d'événements**), c'est toujours **utile de savoir s'en servir**.
C'est simplement beaucoup plus utile et agréable à utiliser. Ils nous facilitent la vie, et leur présence reste dans l'esprit de jQuery !

Sous-Chapitres restants : "Objet événement" et le TP.

Appris dans ce chapitre : espaces de noms pour les événements, méthodes : `bind()`, `unbind()`, `one()`, `trigger()`, `triggerHandler()`, `live()`, `die()`.

Vous êtes maintenant capables de **maîtriser l'ensemble de cette bibliothèque** JavaScript.

Le tutoriel est loin d'être fini. D'autres chapitres seront rédigés.

Userscripts

Maintenant que vous maîtrisez suffisamment cette bibliothèque, vous pouvez aisément vous essayez à **coder des «userscripts »** (ou même des bookmarklets) pour les sites qui utilisent jQuery, par exemple **le Site du Zéro lui-même**.
Les userscripts sont des petits scripts codés par des utilisateurs lambda qui s'exécutent après le chargement d'une page, afin de **la modifier** et ainsi **ajouter des fonctionnalités**. Les userscripts les plus connus pour le SdZ sont les ZppScripts. N'hésitez pas à jeter un coup d'œil !

jQuery hors web

Il faut aussi savoir que l'utilisation du JavaScript ne se limite pas aux pages web : on peut facilement coder des **petites extensions** pour Firefox par l'intermédiaire de Jetpack (module pour Firefox) avec lequel on peut utiliser jQuery (cf. documentation). D'ailleurs, certains navigateurs commencent à migrer vers un système incluant le JavaScript afin de coder des extensions.
Enfin, vous pouvez vous servir de jQuery avec Adobe AIR afin de créer des **applications exécutables** sur tous les systèmes d'exploitations.

Mot de la fin

Voilà... j'espère que vous aussi vous êtes fans de jQuery et que vous continuerez à découvrir sa puissance ainsi que celle de ses **plugins**.
Si jQuery ne vous plaît pas : vous pouvez **apprendre un autre « framework »** comme Prototype ou Yahoo UI.
Maintenant que vous connaissez une bibliothèque JavaScript, ce sera plus facile d'en apprendre une autre en **lisant sa documentation**.

- **une erreur dans le tutoriel** : signalez une erreur pour faire appel à un validateur qui va la corriger. Vous pouvez aussi nous envoyer un message personnel.
- **une question sur le tutoriel, ou une envie d'y participer** : n'hésitez pas à commenter le tutoriel ou poster sur **topic**

officiel du tuto. Vous pouvez aussi y donner vos exemples, ou encore proposer des questions de Q.C.M.
- **un problème** : direction le forum JavaScript en postant un sujet commençant par `[jQuery]`.

Je tenais à remercier les zCorrecteurs, ainsi que MisterDo pour son aide sur la première partie du tutoriel : il m'a beaucoup aidé et m'a aussi suggéré une organisation du tutoriel bien plus efficace !
N'hésitez pas à commenter le tutoriel, à bientôt !