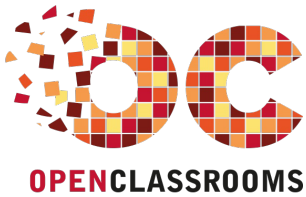
# Développez votre propre plug-in jQuery !

**Par Adrien Guéret (KorHosik)**




www.openclassrooms.com


*Dernière mise à jour le 15/05/2011*

# Sommaire

# Développez votre propre plug-in jQuery !

Par  Adrien Guéret (KorHosik)

Mise à jour : 15/05/2011
Difficulté : Facile Durée d'étude : 1 heure, 30 minutes



**jQuery** est une bibliothèque **JavaScript** très populaire qui est de plus en plus utilisée. De nombreux sites l'ont en effet adoptée pour ses effets dynamiques qui donnent un aspect très professionnel.

Une des forces majeures de **jQuery** est la possibilité d'y ajouter ses propres plug-ins, nous permettant ainsi de réutiliser rapidement et facilement nos codes, mais aussi de les proposer de façon pratique au reste de la communauté.

Le but de ce tuto est de vous apprendre à développer vos propres plug-ins : vous verrez que ça n'a rien de sorcier !

Il va de soit que pour développer un plug-in **jQuery**, il est nécessaire d'avoir un minimum de connaissances en... **jQuery**
Si ce n'est pas votre cas, je vous invite à lire ce tuto, signé tit_toinou.

Sommaire du tutoriel :



## Un... plug-in ?

Commençons par le commencement : qu'est-ce qu'un plug-in concrètement ? De façon globale, il s'agit d'un module, d'un petit logiciel qui ajoute des fonctionnalités à un programme donné afin que ce dernier réponde à des besoins plus précis.

Pour l'anecdote, *plug-in* est un terme anglais qui signifie *brancher*. Cela résume bien leur rôle : on peut très bien imaginer un plug-in comme étant une drôle de boîte que l'on branche sur un appareil pour l'améliorer. En français, il est plutôt recommandé de parler de *module d'extension*, mais le terme anglais est bien plus utilisé, alors c'est celui-ci que nous allons utiliser.

Revenons à ce qui nous intéresse : **jQuery** ! Ici, un plug-in n'est pas un gros logiciel tout compliqué à développer... Non non, un plug-in **jQuery**, c'est une simple fonction JavaScript !

Pour être exécutée, cette drôle de fonction devra être appelée en tant que méthode de nos éléments, comme ceci :

**Code : JavaScript**

```
$('p').notreSuperPlugIn();
//Là, tous les paragraphes de notre page verront notre plug-in
appliqué sur eux !
```

Il ne reste plus qu'à coder notre super plug-in .

# Le code minimum de notre plug-in

## La fonction *jQuery.fn*

Avant de s'attaquer à quelque chose de complexe, commençons par voir les bases : comment ajouter un plug-in à **jQuery** ?

Nous allons nous baser sur la fonction jQuery.fn. Celle-ci est en quelque sorte la fonction mère de **jQuery**. C'est elle qui nous permet d'écrire `$('#element').show()` , `$('#element').fadeTo()` etc... Et c'est donc elle qui nous permettra d'écrire notre plug-in.

Mais comment ? Ça commence plutôt mal car il y a plusieurs façons de faire !

La première étend notre fonction jQuery.fn avec une notation JSON contenant notre plug-in. L'avantage est que nous pouvons ainsi définir plusieurs plug-ins d'un coup :

**Code : JavaScript**

```
jQuery.extend(jQuery.fn,
{
    notreSuperPlugIn: function(parametres)
    {
        //Code de notre plug-in ici
    },
    secondPlugIn: function(parametres)
    {
        //Code du second plug-in ici
    }
});
```

Cette pratique est cependant déconseillée : il est préférable de n'avoir qu'un plug-in par fichier, c'est plus propre, plus facile à maintenir, ça fait des fichiers moins gros... Cela dit, si vous avez en projet une bibliothèque de plug-ins étant liés les uns avec les autres, cette solution vous conviendra .

Mais pour l'instant, évitons de voir trop grand et préférons la seconde méthode : celle-ci ajoute simplement notre plug-in à jQuery.fn, de la même manière que n'importe quelle fonction :

**Code : JavaScript**

```
jQuery.fn.notreSuperPlugIn=function(parametres)
{
   //Code de notre plug-in ici
};
```

Nous adopterons donc cette dernière solution dans la suite de ce tuto : elle a l'avantage d'être plus claire, plus explicite mais, surtout, plus rapide à écrire.

## Enfermons notre plug-in !

Cependant, si vous utilisez **jQuery** depuis quelque temps, vous devez avoir l'habitude d'utiliser dans vos codes le **$** plutôt que le mot **jQuery**. Pour ce faire, nous allons mettre la définition de notre plug-in à l'intérieur d'une fonction anonyme et d'une closure, de cette manière :

**Code : JavaScript**

```
(function($)
```

```
{
    $.fn.notreSuperPlugIn=function(parametres)
    {
        //Code de notre plug-in ici
    };
})(jQuery);
```

Cette syntaxe peut paraître curieuse aux premiers abords, mais elle est toute logique : on met entre parenthèses notre fonction anonyme et on l'appelle aussitôt avec l'argument **jQuery**. Ainsi, dans toute notre fonction, **$** aura la valeur... **jQuery**  !
C'est une technique pour éviter les conflits avec les autres librairies JavaScript utilisant aussi le **$**.

## Récupérer les éléments utilisant le plug-in

Il est temps désormais de récupérer les différents éléments sur lesquels on doit appliquer notre super plug-in. Pour cela, tentons déjà de comprendre ce qui se passe lorsqu'on fait appel à ce dernier. C'est-à-dire, comme on l'a vu au chapitre précédent, quand on écrit ceci :

**Code : JavaScript**

```
$('p').notreSuperPlugIn();
```

Ici, nous voulons appliquer notre plug-in sur tous les paragraphes. En fait, cela revient à appeler une méthode de nos paragraphes. Si vous avez des notions en *POO*, vous savez que pour accéder à l'élément appelant la méthode, il faut utiliser le mot-clé **this**. Si vous n'avez pas de notions en *POO*, sachez simplement que pour accéder à l'élément appelant la méthode, il faut utiliser le mot-clé **this**.

Ainsi, le mot-clé **this** dans notre plug-in fait référence à l'objet **jQuery** appelant notre plug-in ! De petits exemples pour que vous compreniez bien :

**Code : JavaScript**

```
$('div').notreSuperPlugIn(); //this fera référence à l'objet jQuery
$('div')
$('#header').notreSuperPlugIn(); //this fera référence à l'objet
jQuery $('#header')
$('table.commentaire').notreSuperPlugIn(); //this fera référence à
l'objet jQuery $('table.commentaire')
```

... Bref, je crois que vous avez compris le principe  .

Vous savez probablement qu'un objet **jQuery** peut faire lui-même référence à plusieurs éléments de notre page Web. Ainsi, `$('div')` fait référence à toutes les ** ** de notre page !

Et donc, pour pouvoir appliquer notre plug-in sur tous les éléments de notre objet **jQuery**, nous utiliserons la fonction each() :

**Code : JavaScript**

```
(function($)
{
    $.fn.notreSuperPlugIn=function(parametres)
    {
        this.each(function()
        {
            //Code de notre plug-in ici
        });
    };
```

```
})(jQuery);
```

On retrouve notre cher mot-clé **this**, qui représente notre objet **jQuery** appelant le plug-in. Et c'est dans la fonction each() que l'on écrira notre code.

## Ne cassons pas la chaîne !

Il reste toutefois un petit quelque chose à ajouter pour avoir une bonne base. Vous n'êtes pas sans savoir que **jQuery** a pour particularité de chaîner ses fonctions. Comprenez par là que l'on peut faire ceci sans problème :

**Code : JavaScript**

```
$('p').css('color','red').appendTo($('div')).show();
```

Cela est possible car chacune des fonctions, que ce soit css(), appendTo(), show() ainsi que la grande majorité des fonctions de la bibliothèque, renvoient en valeur de retour l'objet **jQuery** qui les a appelées.

Pour respecter cette philosophie, notre plug-in doit permettre de s'ajouter dans cette chaîne, comme ceci :

**Code : JavaScript**

```
$('p').css('color','red').notreSuperPlugIn().appendTo($('div')).show();
```

Pour ce faire, rien de plus simple, il suffit de renvoyer l'objet **jQuery** appelant notre plug-in... Vous savez, notre fameux mot-clé **this**.

**Code : JavaScript**

```
(function($)
{
    $.fn.notreSuperPlugIn=function(parametres)
    {
        this.each(function()
        {
            //Code de notre plug-in ici
        });

        return this;
    };
})(jQuery);
```

Bien entendu, la fonction each() renvoie elle aussi l'objet **jQuery** l'ayant appelé. Ainsi, il est également possible d'écrire ceci :

**Code : JavaScript**

```
(function($)
{
    $.fn.notreSuperPlugIn=function(parametres)
    {
        return this.each(function()
        {
            //Code de notre plug-in ici
        });
    };
```

```
})(jQuery);
```

Ceci sera donc notre code de base pour tous nos futurs plug-ins. Ce n'est pas bien compliqué en fait, il faut juste le savoir, c'est tout .

# Notre premier plug-in !

## Débutons par un plug-in basique

Maintenant qu'on a le code de base, le reste va vite couler de source : ça ne sera que du code **jQuery** traditionnel !

Commençons par un plug-in simple. On va l'appeler *hoverFade* il abaissera lentement l'opacité de nos éléments au passage de la souris dessus. Une fois l'opacité à 0, il remettra rapidement l'opacité de l'élément à la normale. Cela peut donner un effet sympa pour nos liens par exemple, ça changera d'un simple changement de couleur .

On aura donc besoin des fonctions fadeOut() et fadeIn() .

Essayez de le faire vous-même avant de regarder le code ci-dessous, c'est en essayant qu'on apprend !

**Code : JavaScript**

```
(function($)
{
 $.fn.hoverFade=function()
 {
    return this.each(function()
    {
  //On veut que l'élément change au passage de la souris, on
utilise donc mouseover() !
   $(this).mouseover(function()
   {
    //On diminue donc l'opacité lentement
    $(this).fadeOut('slow',function()
    {
     //Une fois l'élément invisible, on le fait réapparaitre
rapidement !
     $(this).fadeIn('fast');
    });
   });
    });
 };
})(jQuery);
```

Il me semble inutile d'expliquer ce code, les commentaires sont assez explicites je pense. Vous pouvez cependant vous demandez pourquoi j'utilise `$(this)` et non notre mot-clé **this**.
C'est tout simplement parce que nous codons dorénavant dans la fonction each(), et non plus directement dans notre plug-in ! Et, dans each(), pour accéder à notre élément on utilise... `$(this)` .

## Et si on testait ?

Ça serait mieux, non ? Commençons par enregistrer le plug-in dans un fichier : pour ce faire, on utilise généralement deux méthodes différentes, selon la version de votre fichier :

- La version de développement : c'est celle qu'on a actuellement, avec le code bien présenté, indenté et commenté. Il est facile à maintenir et permet aux autres développeurs de comprendre ce que vous avez fait. On le nomme généralement de cette façon : **jquery.nomDuPlugIn.js**.

- La version de production : c'est bien évidemment le même code que la version de développement, mais sans l'indentation, sans les commentaires et les retours à la ligne, et qui tient le plus souvent sur une seule ligne ! Le fichier n'est alors plus du tout lisible et le maintenir est un enfer... Mais il a l'avantage d'être plus léger, ce qui le rend idéal pour l'intégrer dans les sites, d'où le terme de "production". Pour transformer notre fichier de développement en version de production, on utilise généralement des outils faits pour, comme le site jscompress. On nomme généralement la version de production de cette façon : **jquery.nomDuPlugIn-min.js**.

Bien entendu, il est préférable de conserver les deux versions. De plus, si vous voulez proposer votre plug-in au reste du monde, il est préférable de mettre à disposition les deux versions également .

Bon, alors, nommons notre fichier **jquery.hoverFade.js** et créons une page HTML pour le tester !

**Code : HTML**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitionnal//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
 <head>
  <title>Notre super plug-in jQuery !</title>
  <meta http-equiv="Content-Type" content="text/html; charset=charset=iso-
8859-1" />
  <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
  <script type="text/javascript" src="jquery.hoverFade.js"></script>
  <script type="text/javascript">
   $(function()
   {
    $('a, h1').hoverFade().css('color','red');
   });
  </script>
 </head>
 <body>
  <h1>Lorem ipsum dolor sit amet</h1>
                <p>
                   Lorem ipsum dolor sit amet, <a href="#">consectetur
adipiscing</a> elit. Praesent sit amet libero odio.
                   Maecenas auctor varius lorem at commodo. Praesent
egestas consectetur hendrerit. Pellentesque
                   condimentum dictum enim, scelerisque convallis erat
mollis quis. Ut cursus tellus sed odio feugiat
                   aliquam. Aenean vel orci nunc. Cras pharetra tortor a
elit laoreet <a href="#">non ornare magna fermentum</a>. Fusce
                   convallis tincidunt rhoncus. Nunc ac elit in tellus
facilisis euismod. Donec vel mi urna. Aliquam in
                   urna dolor, <a href="#">non blandit sapien</a>. Proin
consequat, arcu non dignissim semper, nibh tellus condimentum
                   turpis, vel ornare ipsum massa rhoncus eros.
                </p>
 </body>
</html>
```

Tester !

N'oubliez pas bien sûr d'inclure le plug-in via une balise `<script>` dans le `<head>` ! Veillez aussi à inclure **jQuery**, sans ça notre plug-in ne pourra pas fonctionner .
J'ai choisi pour ma part d'inclure le fichier hébergé par Google.

Enfin, une troisième balise `<script>` a été ajoutée afin d'appliquer notre plug-in sur les éléments que l'on souhaite. Dans l'exemple ci-dessus, les liens seront donc animés au passage de la souris, ainsi que les titres de premier niveau. L'appel à la fonction .css() n'est là que pour vérifier que le chaînage fonctionne bien .

Ce n'est donc pas si compliqué, n'est-ce pas ?

# Un peu de personnalité avec les paramètres

## Les paramètres simples

Notre plug-in ci-dessus est bien sympathique, quoique vraiment primaire.

Il serait intéressant de proposer à l'utilisateur de le paramétrer. Par exemple, notre plug-in *hoverFade()* impose actuellement une vitesse d'animation. Ça pourrait être sympathique de paramétrer cette vitesse non ?

La façon de procéder à laquelle vous pensez sans doute est la suivante :

**Code : JavaScript**

```
(function($)
{
 $.fn.hoverFade=function(vitesseFadeOut, vitesseFadeIn)
 {
    return this.each(function()
    {
  //On veut que l'élément change au passage de la souris, on
utilise donc mouseover() !
   $(this).mouseover(function()
   {
    //On diminue donc l'opacité lentement
    $(this).fadeOut(vitesseFadeOut,function()
    {
     //Une fois l'élément invisible, on le fait réapparaitre
rapidement !
     $(this).fadeIn(vitesseFadeIn);
    });
   });
    });
 };
})(jQuery);
```

Pour ensuite l'appeler de cette façon :

**Code : JavaScript**

```
$('p').hoverFade(2500,2000);
```

Cette solution fonctionne, même si dans l'idéal il faudrait vérifier les valeurs envoyées. Cependant, elle n'est pas des plus pratique.

Pourquoi ? Imaginons la plug-in suivant :

**Code : JavaScript**

```
$.fn.unSuperPlugIn=function(vitesse, couleur, largeur, hauteur,
position_x, position_y, cible){};
```

Peu importe ce que fait ce plug-in, intéressons uniquement aux paramètres. Imaginons que nous souhaitons ne renseigner que la position de notre élément. Nous serions obligés d'indiquer tous les paramètres précédents, à savoir la vitesse, la couleur et les dimensions... Pas très pratique n'est-ce pas ?

Imaginons de plus que vous mettiez plus tard à jour votre plug-in en ajoutant un paramètre : tous les anciens scripts l'utilisant ne fonctionneront plus, la rétro-compatibilité est inexistante !

Ceci est clairement un problème... Mais heureusement, il existe une solution simple pour le résoudre .
L'idée est de n'admettre qu'un seul paramètre à notre plug-in. Ce paramètre sera une sorte de "super variable" qui en contiendra plusieurs autres "normales". Ce paramètre sera ce qu'on appelle un **objet littéral**. Nous l'avons déjà un peu évoqué dans la première partie de ce tuto, si vous vous rappelez .

Cependant, si on veut permettre de ne préciser que quelques paramètres, il nous faut des valeurs par défaut ! Celles-ci seront elles aussi stockées dans un objet littéral. Si on en reste là, on a donc deux objets différents : un fourni par l'utilisateur, et un autre contenant nos valeurs par défaut. Mais il faut les mettre en commun pour savoir comment traiter les paramètres ! Pour cela, on utilise la fonction jQuery.extend(), qui permet de "fusionner" des objets en un seul .

**Code : JavaScript**

```
(function($)
{
 $.fn.hoverFade=function(options)
 {
    //options est donc un objet littéral, ne l'oublions pas !

            //On définit nos paramètres par défaut
            var defauts=
            {
                "vitesseFadeOut": "slow",
                "vitesseFadeIn": "fast"
            };

            //On fusionne nos deux objets ! =D
            var parametres=$.extend(defauts, options);

            return this.each(function()
    {
  //On accèdera à la vitesse du fadeOut via
parametres.vitesseFadeOut
                //et à celle du fadeIn via parametres.vitesseFadeIn

                //On veut que l'élément change au passage de la
souris, on utilise donc mouseover() !
                $(this).mouseover(function()
                {
                    //On diminue donc l'opacité lentement

$(this).fadeOut(parametres.vitesseFadeOut,function()
                        {
                            //Une fois l'élément invisible, on le fait
réapparaître rapidement !
                            $(this).fadeIn(parametres.vitesseFadeIn);
                        });
                });
    });
 };
})(jQuery);
```

Pour appeler notre plug-in, il est possible de préciser un des paramètres, les deux, ou aucun .

**Code : JavaScript**

```
$('a').hoverFade();
$('h1').hoverFade({"vitesseFadeIn": 2350});
$('#test').hoverFade({"vitesseFadeIn": 2350,"vitesseFadeOut":
"normal"});
```

## Des fonctions en paramètre !

En JavaScript, une variable peut très bien stocker une fonction entière. A ce titre, il est possible de passer des fonctions en paramètre à nos plug-ins ! Tentons d'ajouter un *callback* à notre plug-in, ou si vous préférez une fonction qui sera appelée une fois l'animation finie. Ajoutons déjà notre nouveau paramètre dans les valeurs par défaut :

**Code : JavaScript**

```
var defauts=
{
    "vitesseFadeOut": "slow",
    "vitesseFadeIn": "fast",
    "callback": null
};
```

A la fin de l'exécution de notre plug-in, il faut donc lancer cette fonction si elle existe :

**Code : JavaScript**

```
$(this).mouseover(function()
{
     //On diminue donc l'opacité lentement
     $(this).fadeOut(parametres.vitesseFadeOut,function()
     {
          //Une fois l'élément invisible, on le fait réapparaitre
rapidement !
          $(this).fadeIn(parametres.vitesseFadeIn,function()
          {
               //Si le parametre callback ne vaut pas null, c'est
qu'on a précisé une fonction !
               if(parametres.callback)
               {
                  parametres.callback();
               }
          });
     });
});
```

Ainsi, si nous appelons notre plug-in de cette manière :

**Code : JavaScript**

```
$('a, h1').hoverFade({"callback": function()
{
    alert('Animation finie !');
}});
```

Tester !

Une pop-up s'affichera à la fin de l'animation  .

## Des paramètres pour nos fonctions-paramètres

Là, ça commence à devenir tordu . Pourquoi ne pas donner de paramètres aux fonctions que l'on passe en paramètre ? De cette façon :

**Code : JavaScript**

```
$('a, h1').hoverFade({"callback": function(data)
{
    alert('Voici ce que contient data : '+data);
}});
```

Bien entendu, si on teste ce code on verra que data vaut **undefined**. Et pour cause, on ne l'a pas défini . Rappelez-vous comment nous avons procédé tout à l'heure : nous avons appelé la fonction dans le code du plug-in, via l'instruction suivante :

**Code : JavaScript**

```
if(parametres.callback)
{
    parametres.callback();
}
```

Là, vous devriez commencer à comprendre le principe : puisque c'est là qu'on appelle notre fonction, c'est ici qu'on lui fournit la liste des paramètres !

**Code : JavaScript**

```
if(parametres.callback)
{
    parametres.callback("toto",23,$(this).text());
}
```

Lorsqu'on appelle notre plug-in, nous pouvons accéder à ces variables en leur donnant n'importe quel nom (même si ce n'est évidemment pas conseillé, autant faire un code clair et compréhensible ) :

**Code : JavaScript**

```
$('a, h1').hoverFade({"callback": function(data,radiateur,chocolat)
{
    alert("Voici ce que contient data : " + data + "\nEt radiateur :
" + radiateur + "\nEt enfin chocolat : " + chocolat);
}});
```

Tester !

Si vous décidez de mettre des paramètres à vos fonctions-paramètres, sachez qu'il n'est pas nécessaire de les indiquer lorsque vous appelez votre plug-in. La fonction se lancera sans erreur et fonctionnera tout de même, vous ne pourrez juste pas accéder aux variables envoyées par le plug-in .

**Code : JavaScript**

```
$('a, h1').hoverFade({"callback": function()
{
    //On peut mettre ce que l'on veut ici, on ne peut juste pas
accéder aux trois variables qu'on l'on a défini dans notre plug-in
```

```
=)
    alert("Voici ce que contient data : " + data + "\nEt radiateur :
" + radiateur + "\nEt enfin chocolat : " + chocolat);
    //Puisqu'on essaye d'accéder aux variables, le script plante,
alors attention !
}});
```

Bien évidemment, pour notre petit plug-in *hoverFade*, les paramètres dans les fonctions-paramètres ne sont pas très utiles. Afin de conclure ce tuto, nous allons donc développer ensemble un plug-in bien plus complexe que celui-ci .

## TP : une galerie d'images

Afin de vous faire un peu exercer et appliquer tout ce qu'on a vu, nous allons faire un plug-in plus complexe que *hoverFade*. Je vous propose donc de réaliser une petite galerie de défilement d'images .

### Ce que nous allons réaliser

Notre plug-in agira sur un élément contenant des images, et uniquement des images. Ces images s'afficheront dans cet élément une par une, en s'effaçant progressivement pour faire apparaître la suivante. L'élément affecté deviendra donc une petite galerie d'images à lui tout seul .

Éventuellement, le plug-in ajoutera un lien sur ces images pour les rendre cliquables.

### Les paramètres du plug-in

Bien entendu, le plug-in sera personnalisable à travers les paramètres suivants :

- **interval** : Intervalle entra chaque image, en millisecondes. *Défaut : 5000.*
- **width** : Largeur de la galerie. *Défaut : '300px'.*
- **height** : Hauteur de la galerie. *Défaut : '150px'.*
- **scaleWidth** : Doit-on adapter la largeur de l'image ? *Défaut : true.*
- **scaleHeight** : Doit-on adapter la hauteur de l'image ? *Défaut : true.*
- **makeLinks** : Doit-on créer des liens ? *Défaut : false.*
- **callback** : Fonction appelée à chaque nouvelle image. *Défaut : null.*

Les paramètres *scaleWidth* et *scaleHeight* indiquent si le plug-in doit adapter la taille des images aux dimensions de l'élément contenant (définies par *width* et *height*).

Le paramètre *callback* sera une fonction appelée (si elle est définie bien sûr !) à chaque fois qu'une image apparaît. Cette fonction aura comme paramètre l'image qui vient d'apparaître.

*makeLinks* indique si les images seront cliquables ou non. Si oui, on admettra que le plug-in récupérera le lien via l'attribut **alt** des images.

### Le code HTML

Trouvez ci-dessous le code HTML sur lequel vous pourrez tester votre plug-in (que j'ai nommé très originalement *imageSlide* ) :

**Code : HTML**

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitionnal//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
 <head>
  <title>Le plug-in imageSlide !</title>
  <meta http-equiv="Content-Type" content="text/html; charset=charset=iso-
```

```
8859-1" />
  <script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
  <script type="text/javascript" src="jquery.imageSlide.js"></script>
  <script type="text/javascript">
   $(function()
   {
    $('#galerie').imageSlide(); //Appel par défaut du plug-in
   });
  </script>
 </head>
 <body>
  <div style="text-align: center;">
   <h1>Le plug-in imageSlide !</h1>
   <p id="galerie" style="margin: auto;">
    <img src="1.jpg" title="Zèbre"
alt="http://fr.wikipedia.org/wiki/Z%C3%A8bre" />
    <img src="2.jpg" title="Pont" alt="http://fr.wikipedia.org/wiki/Pont"
/>
    <img src="3.jpg" title="Plage" alt="http://fr.wikipedia.org/wiki/Plage"
/>
    <img src="4.jpg" title="Fleurs"
alt="http://fr.wikipedia.org/wiki/Fleur" />
   </p>
   <p id="description">Zèbre</p>
   <p style="tetx-align: right;">Images provenant de <a
href="http://www.freephotobank.org/">http://www.freephotobank.org/</a>.</p>
  </div>
 </body>
</html>
```

J'ai ajouté un petit paragraphe d'id **description** pour le test du plug-in : celui-ci contiendra le **title** de l'image qui vient d'apparaître. Il vous faudra donc utiliser le paramètre *callback* pour ça .
Les images que j'ai utilisées proviennent du site http://www.freephotobank.org/ (d'où les crédits dans le code ), vous pouvez bien évidemment utiliser les vôtres .

## Quelques conseils avant de commencer

Le premier conseil que je peux vous donner est de ne pas foncer tête baissée dans le code : c'est le meilleur moyen pour aller droit dans le mur !
Analysez bien la problématique du TP et procédez par étape. Par exemple, commencez par coder le plug-in dans sa façon de fonctionner la plus globale, comme s'il n'avait pas de paramètres, en ne prenant en compte que leurs valeurs par défaut. Une fois que cette version du plug-in fonctionne correctement, et donc que vous avez une base saine, vous pouvez vous attaquer aux paramètres .

Voici également quelques pistes qui vous seront sans doute utiles :

- Dans le cas où les images sont plus grandes que la galerie et qu'elles ne doivent pas s'adapter à la taille de ce dernier, il serait préférable de fixer la propriété CSS **overflow** de la galerie à **hidden**.
- Pour ne pas vous mélanger trop avec tous les `$(this)` qui risquent d'y avoir dans votre code, je vous conseille de stocker l'élément principal dans une variable. Ça sera plus clair et plus facile à maintenir .
- Pour parcourir les images, vous aurez probablement à récupérer le nombre d'images de la galerie. Un compteur vous sera également utile, en particulier combiné avec le sélecteur eq() de **jQuery** .
- Afin d'adapter les dimensions des images à la taille de la galerie, il suffit simplement de mettre les propriétés CSS **width** et/ou **height** à **100%** .
- Pour faire propre, vérifiez que le paramètre *interval* n'est pas trop petit. S'il vaut moins de 1000 millisecondes, fixez-le à 1000 : en effet, si les images défilent trop vite ça risque non seulement de faire mal aux yeux, mais aussi de faire chauffer un peu trop le pauvre moteur JavaScript de l'utilisateur .

Bon, ça devrait suffire je suppose, je vais vous laisser réfléchir un peu quand même .

Aller, bonne chance à tous !

## Correction

Je vous invite vivement à ne consulter cette correction que si vous avez réellement essayé. Recopier bêtement le code n'aurait aucun intérêt pédagogique et ce tuto ne vous aurait servi à rien . Sur ce, voici donc ma solution. Ce n'est pas la seule façon de coder ce plug-in et ce n'est probablement pas la meilleure. Mais au moins elle fonctionne, et elle applique tout ce qu'on a vu .

**Code : JavaScript**

```
(function($)
{
 $.fn.imageSlide=function(options)
 {
            //On définit nos paramètres par défaut
            var defauts=
            {
  'interval': 5000,  //Intervalle entre chaque image, en
millisecondes
  'width': '300px',  //Largeur de la galerie
  'height': '150px',  //Hauteur de la galerie
  'scaleWidth': true,  //Doit-on adapter la largeur de l'image ?
  'scaleHeight': true,         //Doit-on adapter la hauteur de
l'image ?
  'makeLinks': false,  //Doit-on créer des liens ?
  'callback': null  //Fonction appelée à chaque nouvelle image
            };

            //On fusionne nos deux objets ! =D
            var parametres=$.extend(defauts, options);

      //Si l'intervalle est trop court, on le fixe à 1 seconde
      //pour éviter que le défilement soit trop rapide
      parametres.interval=Math.max(1000,parametres.interval);

            return this.each(function()
      {
    //On stocke notre élément dans une variable par commodité
    var element=$(this);

    //On compte le nombre d'images de notre galerie
    var totalImages=element.find('img').length;

    //Le compteur pour nous permettre de parcourir les images
    var compteur=0;

    //On modifie le style de notre galerie
    element.css(
    {
     'border': '1px solid #000',
     'width': parametres.width,
     'height': parametres.height,
     'overflow': 'hidden'
    }).find('img').each(function(id) //Puis on parcourt enfin chaque
image !
    {
     //Si on doit adapter les dimensions des images, on le fait
     if(parametres.scaleWidth)
     {
      $(this).css('width','100%')
     }
     if(parametres.scaleHeight)
     {
      $(this).css('height','100%')
     }
```

```
            //Si on fait des liens, on change le curseur et on
            //enlève la bordure bleue toute moche
            if(parametres.makeLinks)
            {
             $(this).css(
             {
              'cursor': 'pointer',
              'border': '0px'
             });
             //Et on entoure l'image de son lien, récupéré via le alt !
             $(this).wrap('<a href="'+$(this).attr('alt')+'"></a>');
            }

            //On ne fait apparaitre que la première image
            if(id>0)
            {
             $(this).hide();
            }
           });

           //Et on définit enfin notre défilement
           setInterval(function()
           {
            //On récupère l'image actuellement visible et on la fait
        disparaitre
            element.find('img:eq('+compteur+')').fadeOut(function()
            {
             //On incrémente le compteur si on n'est pas sur la dernière
        image
             //Sinon, on le remet à 0
             if(compteur!=totalImages-1)
             {
              compteur++;
             }
             else
             {
              compteur=0;
             }
             //Et on fait donc apparaitre l'image suivante
             element.find('img:eq('+compteur+')').fadeIn(function()
             {
              //Et si on a une fonction définie, on l'appelle !
              //Et on lui passe notre image en paramètre
              if(parametres.callback)
              {
               parametres.callback($(this));
              }
             });
            });
           },parametres.interval);
          });
         };
        })(jQuery);
```

J'ai commenté le code pour que vous compreniez bien ce que je fais. Vous voyez que le tout est en fait très simple : je me contente d'appliquer ce qu'on l'a vu tout au long du tuto pour la mise en place du plug-in. Tout le reste est du code **jQuery** relativement basique  .

Si on reprend notre code HTML précédent, voici une des nombreuses façons d'appeler notre plug-in :

**Code : JavaScript**

```
$('#galerie').imageSlide(
{
 'width': '500px',
 'height': '300px',
```

```
        'makeLinks': true,
        'callback': function(image)
        {
         $('#description').text(image.attr('title'));
        }
       });
```

Tester !

Le rendu final est plutôt sympatoche, non ?
Alors, est-ce difficile de créer son propre plug-in **jQuery** ?

Avec un peu de travail et beaucoup de patience, nous pouvons faire des choses vraiment très intéressantes, à l'image de jQuery UI, plug-in officiel permettant de réaliser de véritables interfaces fluides pour nos utilisateurs.

Avant de vous laisser partir, je me permets de vous donner un ultime conseil : si vous avez pour ambition de distribuer votre plug-in aux autres développeurs, pensez à bien le documenter ! Que fait-il, comment fonctionne-t-il, comment utiliser ses paramètres etc... La qualité et le succès d'un plug-in dépendent en partie de la clarté de sa documentation, pensez-y .

Aller, soyez imaginatifs, et développez-nous de beaux plug-ins !