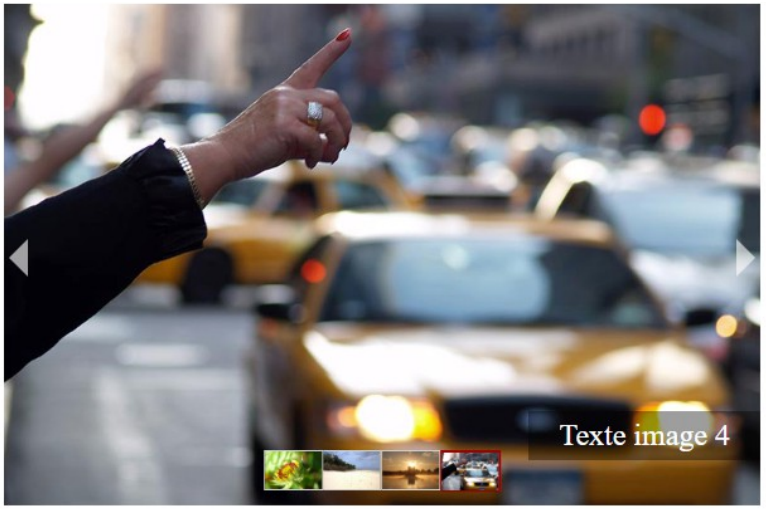
CEFii
www.cefii.fr

Tutoriel jQuery - Slideshow

0
Introduction

Introduction

L'objectif de ce tutoriel est d'aborder des notions complémentaires JavaScript / jQuery sur le même principe que le tutoriel précédent.
Pour cela, nous construirons le slideshow suivant :

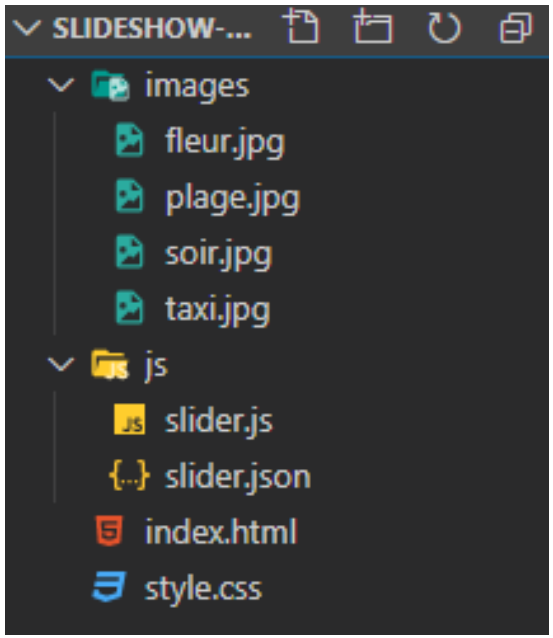


Le slideshow défilera de manière automatique ou via un clic sur une flèche.
Au survol, le slideshow arrêtera de défiler.
On pourra aussi sélectionner directement une image lors du clic sur une des miniatures.
On aura enfin la possibilité d'afficher ces miniatures sous forme de puce.
Chaque image aura éventuellement une légende associée et un lien cliquable.

1 Structure

Structure

Créez un nouveau dossier et les différents fichiers afin d'obtenir la structure suivante :



- un dossier « images » comprenant 4 images de votre choix
- un dossier « js » comprenant :
  - un fichier *slider.js* correspondant au script de notre slideshow
  - un fichier *slider.json* nous permettant de paramétrer le slideshow
- un fichier *style.css* pour la mise en forme
- un fichier *index.html*

2
JSON

fichier JSON

Nous allons à nouveau utiliser le format JSON (*JavaScript Object Notation*) afin de pouvoir paramétrer facilement notre slideshow et donc ne pas avoir à modifier le code du script.

Dans ce fichier, nous renseignerons 2 éléments principaux:

- une clé *prefs* permettant de paramétrer des options de préférences :
  - *duree :* durée de l'animation du slideshow
  - *size :* dimensions *w* (width) et *h* (height) du slideshow
  - *tIMG :* temps d'animation de chaque slide
  - *tTXT :* temps d'apparition du texte
  - *dossier :* nom du dossier des images
  - *commandes :* affichage ou non des miniatures
  - *vignettes :* affichage de miniatures sous forme de vignettes ou puces
  - *pause :* activation de la pause au survol du slidehow

- une clé *imgs* ayant pour valeur un tableau permettant de définir un objet pour chaque slide.

2
JSON

Pour chacun de ces slides, on pourra définir les clés :

- *file* : nom du fichier image
- *text* : légende de l'image
- *btn* : objet permettant de définir un bouton cliquable défini par :
  - *label* : texte du bouton
  - *lien* : URL vers lequel le bouton envoie
- *style* : classe qui sera appliquée au paragraphe de la légende (« dark » ou « light ») .

Nous obtenons (à modifier en fonction du nom de vos fichiers images) :

slider.json

```
{
    "prefs": {
        "duree": 2800,
        "size": {"w": "800", "h": "532"},
        "tIMG": 800,
        "tTXT": 800,
        "dossier": "images",
        "commandes": false,
        "vignettes": true,
        "pause": true
    },
    "imgs": [
        {"file": "fleur.jpg", "text": "Texte image 1", "btn": {"label": "En savoir plus", "lien": "https://www.cefii.fr"}, "style": "dark"},
        {"file": "plage.jpg", "text": "Texte image 2", "btn": {"label": "", "lien": ""}, "style": "light"},
        {"file": "soir.jpg", "text": "Texte image 3", "btn": {"label": "", "lien": ""}, "style": "dark"},
        {"file": "taxi.jpg", "text": "Texte image 4", "btn": {"label": "", "lien": ""}, "style": "dark"}
    ]
}
```

3
HTML

fichier HTML

Le fichier *index.html* sera le suivant:

index.html

```html
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initiale-scale=1, user-scalable=yes" >
    <title>Slideshow</title>
    <script
        src="https://code.jquery.com/jquery-3.4.1.min.js"
        integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
        crossorigin="anonymous"></script>
    <script src="js/slider.js"></script>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div id="slideshow"></div>
</body>
</html>
```

On retrouve donc différents appels aux fichiers et la création d'une unique *div#slideshow* .

4
script

fichier JS

Au travers de ce fichier, nous allons charger les données, construire toute la structure HTML du slideshow et mettre en place les différents effets.
N'oubliez pas d'abuser du *console.log()* lors de vos développements JS afin de bien visualiser les données récupérées, des messages d'erreur éventuels etc.

La 1ère étape consiste donc à charger les données du fichier JSON.
Nous procédons de la même manière que pour le tutoriel précédent :

slider.js

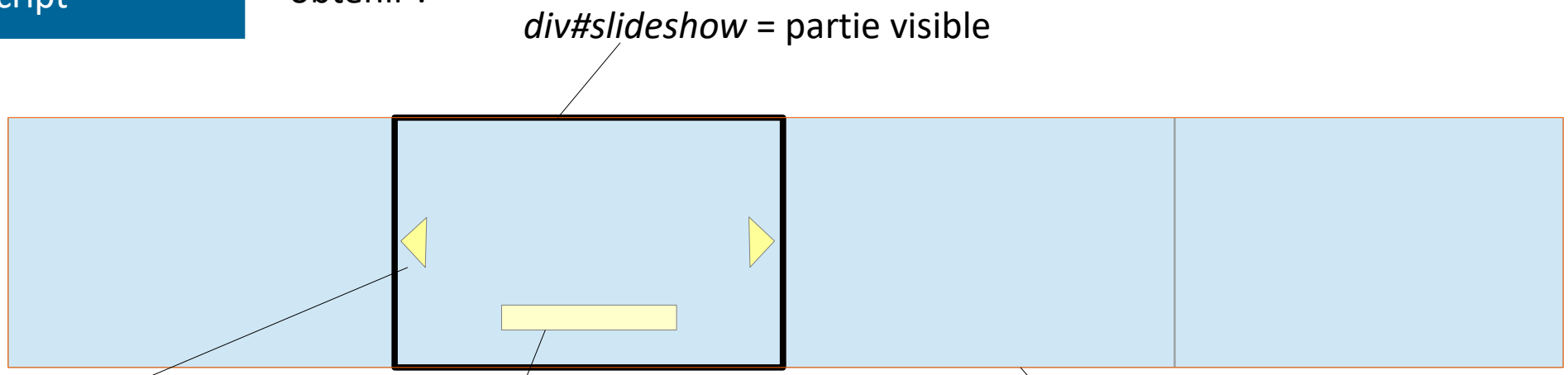
```
1  var datas = {};
2  $.getJSON('js/slider.json',init);
3  function init(obj){
4      datas = obj;
5      console.log(datas);
6  }
```

Résultat dans la console après exécution du fichier HTML :

```
▼ {prefs: {…}, imgs: Array(4)} i
  ▶ prefs: {duree: 2800, size: {…}, tIMG: 800, tTXT: 800, dossier: "images", …}
  ▶ imgs: (4) [{…}, {…}, {…}, {…}]
  ▶ __proto__: Object
```

4 script

Les données étant chargées, nous allons maintenant construire nos différents éléments HTML. Voici tout d'abord la structure finale que nous souhaitons obtenir :



div.fleches
Pour chaque flèche de classe « gauche » ou « droite » .

div#puces
Contenant les miniatures des slides sous forme de balise *img.vignette* ou de *div.puce*.

div#slides = partie contenant autant de *div.item* que d'images.
Chaque *div.item* contiendra une balise *img* et éventuellement un paragraphe pour la légende ainsi qu'un bouton cliquable.

4
script

La structure HTML à obtenir sera donc :

```
<div id="slideshow" style="max-width: 800px;">
  <div id="slides" style="width: 3200px; margin-left: 0px;">
    <div class="item" data-ref="0" style="max-width: 800px;">
      <img src="images/fleur.jpg" alt="Texte image 1">
      <p class="dark" style>Texte image 1</p>
      <button style>En savoir plus</button>
    </div>
    <div class="item" data-ref="1" style="max-width: 800px;">…</div>
    <div class="item" data-ref="2" style="max-width: 800px;">…</div>
    <div class="item" data-ref="3" style="max-width: 800px;">…</div>
  </div>
  <div class="fleches droite"></div>
  <div class="fleches gauche"></div>
  <div id="puces">
    <img class="vignette actif" src="images/fleur.jpg" data-ref="0">
    <img class="vignette" src="images/plage.jpg" data-ref="1">
    <img class="vignette" src="images/soir.jpg" data-ref="2">
    <img class="vignette" src="images/taxi.jpg" data-ref="3">
  </div>
</div>
```

4 script

Nous définissons pour cela une fonction *createSlider()* dans notre fichier *slider.js* :

```
function createSlider() {
    var chemin = datas.prefs.dossier;
    var slideshow = $("#slideshow");
    var slides = $('<div>');
    slides.attr('id','slides');
    slideshow.append(slides);
}
```

Nous avons donc créé tout d'abord une variable permettant de récupérer le nom du dossier des images renseigné dans *prefs* du JSON, ce qui nous servira par la suite.
Ensuite nous sélectionnons la *div#slideshow* dans laquelle nous ajoutons un nouvel élément *div* ayant pour id « slides » .

Exécutez cette fonction au sein de la fonction *init()* . On obtient dans l'inspecteur :

```
<div id="slideshow">
    <div id="slides"></div>
</div>
```

4 script

Au sein de la *div#slides* créée, nous souhaitons ensuite ajouter pour chaque image du tableau *imgs* de *datas* une *div.item* comprenant :

- un attribut personnalisé *data-ref* pour lequel nous associons l'index de l'élément en cours ;
- une balise *img* et ses attributs *src* et *alt* ;
- un paragraphe pour la légende correspondant à la clé *text* de l'objet JSON ;
- ce paragraphe aura pour classe la valeur de la clé *style* de l'objet JSON et sera ensuite masqué pour pouvoir apparaître ensuite après l'animation du slide ;
- un bouton éventuel ayant pour texte la valeur du *label* de l'objet et menant vers l'URL du lien lors du clic ;
- ce bouton sera aussi masqué pour apparaître après l'animation du slide.

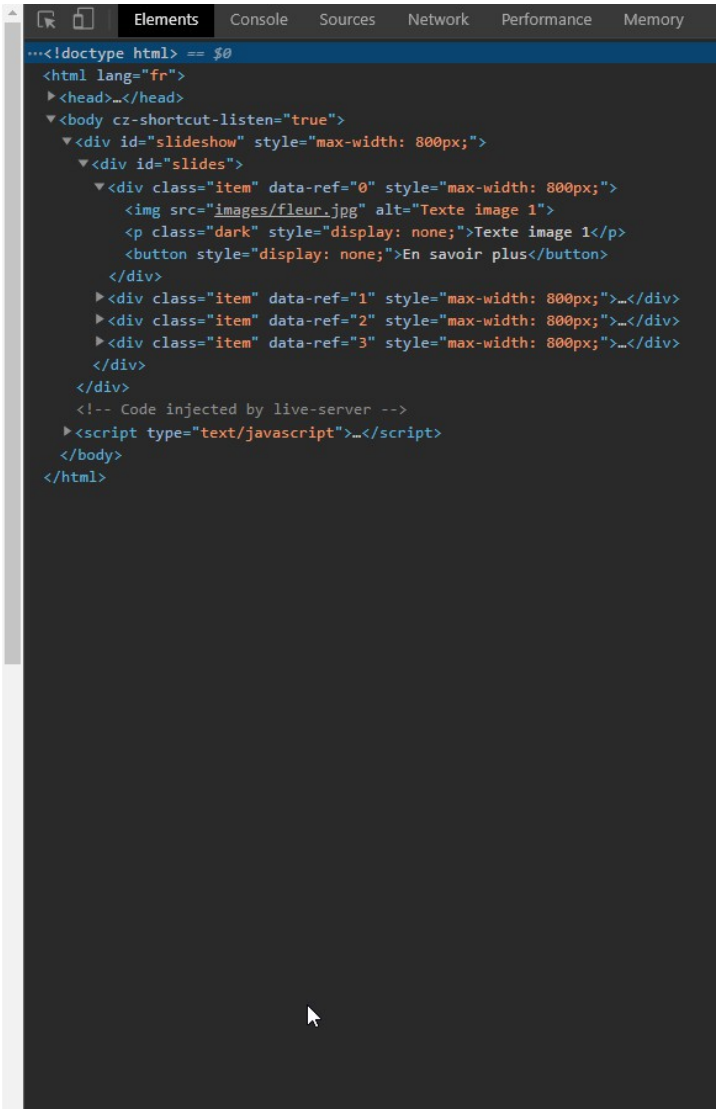
4 script

Nous ajoutons donc la boucle *each()* au sein de la fonction *createSlider()* définie de la manière suivante :

```
$.each(datas.imgs, function(index,val) {
    var item = $('<div>');
    item.addClass('item');
    item.attr("data-ref",index);
    slides.append(item);
    var image = $('<img>');
    image.attr('src', chemin + "/" + val.file);
    image.attr('alt', val.text);
    item.append(image);
    var parag = $('<p>');
    parag.text(val.text);
    parag.addClass(val.style);
    parag.hide();
    item.append(parag);
    if(val.btn.label != "") {
        var bouton = $('<button>');
        bouton.text(val.btn.label);
        bouton.click(function() {
            window.open(val.btn.lien, '_blank');
        });
        bouton.hide();
        item.append(bouton);
    }
});
```

4
script

N'oubliez pas de vérifier dans le navigateur que les éléments sont bien insérés :



4
script

Ajoutons ensuite la mise en forme suivante afin de positionner et styliser les éléments créés précédemment :

style.css

```css
#slideshow {
    margin: 0 auto;
    position: relative;
    overflow: hidden;
}
#slideshow img{
    max-width:100%;
    height: auto;
}
#slides {
    display: flex;
}
#slides .item{
    position: relative;
}
#slides button{
    position: absolute;
    top:0.5em;
    right: 0.5em;
    font-size: 1em;
    border-radius:0.25em;
    border:none;
    background-color: lightgrey;
    padding: 0.25em 1em;
    cursor: pointer;
}
```

Le *overflow:hidden* est important pour masquer les autres slides.

```css
#slides .item p{
    position: absolute;
    bottom:10%;
    right: 0;
    margin: 0;
    font-size: 2em;
    padding: 0.25em 1em;
    color: white;
    background-color: rgba(0,0,0,0.4);
}
#slides .item p.dark{
    color: white;
    background-color: rgba(0,0,0,0.4);
}
#slides .item p.light{
    color: #000;
    background-color: rgba(255,255,255,0.4);
}
```

4
script

Nous devons toutefois ajouter des styles dynamiquement depuis le script pour attribuer :

- la largeur maxi du slideshow définie dans le JSON ;
- une largeur à la *div#slides* en fonction du nombre d'images afin que les images soient bien alignées ;
- une largeur aux *div.item* en fonction de la largeur de la fenêtre et donc de celle de la *div#slideshow*.

Nous créons alors une nouvelle fonction :

```
function largeur(){
    $("#slideshow").css("max-width",datas.prefs.size.w+"px");
    var largeur = $("#slideshow").width();
    var lgSlides = datas.prefs.size.w*datas.imgs.length;
    $("#slides").css("width", lgSlides+"px");
    var item = $("#slideshow .item");
    item.css("max-width",largeur);
}
```

4
script

Cette fonction doit être exécutée :

- après avoir créé les éléments ;
- lors du redimensionnement de la fenêtre (responsive).

Nous ajoutons donc :

- en fin de fonction *createSlider()* :

```
largeur();
```

- en dehors de toute fonction :

```
$(window).resize(function(){
    largeur();
});
```

Testez dans le navigateur : le slideshow s'adapte à la fenêtre.

4
script

Nous souhaitons maintenant ajouter les 2 flèches qui nous permettront de faire ensuite défiler le slideshow manuellement.
Ces 2 flèches seront 2 *div* vides de classe « fleches » ayant chacune une classe supplémentaire « gauche » ou « droite » .
Ces classes nous permettent de créer totalement ces flèches en CSS :

```
46  .fleches{
47      position: absolute;
48      top: 50%;
49      -webkit-transform: translate(0,-50%);
50      transform:t ranslate(0,-50%);
51      width: 0;
52      height: 0;
53      cursor: pointer;
54      border-top: 20px solid transparent;
55      border-bottom: 20px solid transparent;
56  }
57  .droite{
58      right: 5px;
59      border-left: 20px solid rgba(255,255,255,.6);
60  }
61  .gauche{
62      left: 5px;
63      border-right: 20px solid rgba(255,255,255,.6);
64  }
```

4
script

Pour ajouter ces 2 nouveaux éléments HTML, nous créons une nouvelle fonction *placeCommandes()* qui sera exécutée depuis la fonction *init()* :

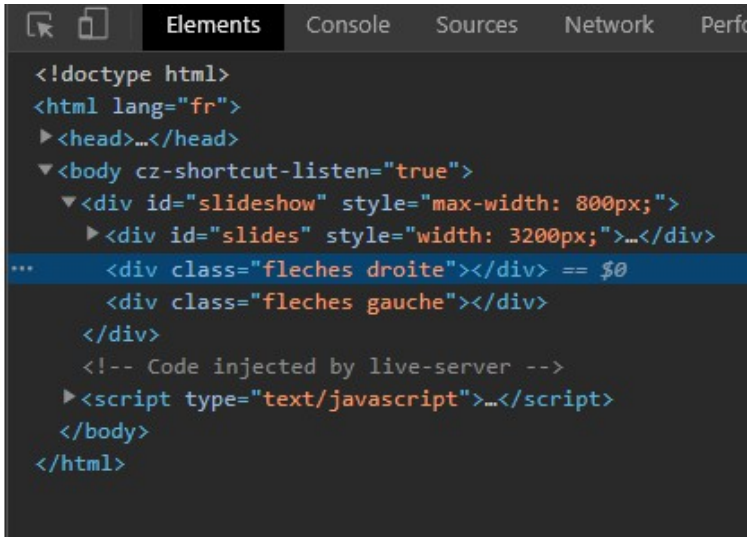
```
function init(obj) {
    datas = obj;
    createSlider();
    placeCommandes();
}
```

```
function placeCommandes(){
    var slideshow = $("#slideshow");
    var fleches = ['droite', 'gauche'];
    $.each(fleches, function(index, val){
        var fleche = $("<div>");
        fleche.addClass("fleches");
        fleche.addClass(val);
        slideshow.append(fleche);
    });
}
```

Dans cette fonction, nous avons donc initialisé une variable récupérant la *div#slideshow* et créé une variable tableau pour définir nos 2 flèches, sur laquelle nous bouclons pour créer les *div* correspondantes (code DRY).

4
script

Nos flèches sont bien présentes :



4 script

Ensuite nous souhaitons ajouter les miniatures des images ou les puces, si l'option « commandes » des *prefs* du JSON sont paramétrées à *true* :

```
{
    "prefs": {
        "duree": 2800,
        "size": {"w": "800", "h": "532"},
        "tIMG": 800,
        "tTXT": 800,
        "dossier": "images",
        "commandes": true,
        "vignettes": true,
        "pause": true
    },
```

Si c'est le cas, nous créerons alors une *div#puces*.
Au sein de celle-ci, nous devrons pour chaque image du tableau *imgs* de *datas* créer une miniature *img.vignette* si l'option *vignettes* est paramétrée à *true*, ou créer une *div.puce* qui aura l'aspect d'une puce dans le cas contraire.
Dans tous les cas, nous ajouterons un attribut *data-ref* pour lequel nous attribuerons en valeur l'index de l'élément parcouru, ce qui nous permettra par la suite de faire apparaître le bon slide lors du clic.

4
script

Nous ajoutons alors à la fonction *placeCommandes()* le code suivant :

```
if(datas.prefs.commandes){
    var puces = $("<div>");
    puces.attr('id','puces');
    slideshow.append(puces);
    var chemin = datas.prefs.dossier;
    $.each(datas.imgs,function(index, val){
        var puce;
        if(datas.prefs.vignettes){
            puce = $('<img>');
            puce.addClass('vignette');
            puce.attr('src',chemin + '/' + val.file);
        } else {
            puce = $('<div>');
            puce.addClass('puce');
        }
        puce.attr('data-ref',index);
        puces.append(puce);
    });
}
```

4
script

Pour l'affichage de ces éléments, nous ajoutons le CSS :

```css
#puces{
    overflow: hidden;
    position: absolute;
    display: -webkit-box;
    display: flex;
    bottom: 1.25em;
    left: 50%;
    -webkit-transform:translate(-50%);
    transform:translate(-50%);
}
#puces .vignette{
    width: 60px;
    height: 40px;
    margin: 0;
    display: block;
    cursor: pointer;
    border: 1px solid #ccc;
}
#puces .puce{
    width: 14px;
    height: 14px;
    border-radius:50%;
    background-color: rgba(255,255,255,.6);
    margin: 5px;
    position: relative;
    cursor: pointer;
}
```

4
script

N'oubliez pas de tester :

- En passant *commandes* à ***false*** dans le JSON :

- En passant *commandes* à ***true*** dans le JSON et *vignettes* à ***true*** :

En passant *commandes* à ***true*** dans le JSON et *vignettes* à ***false*** :

4
script

Enfin nous devons mettre en place l'effet de slide, c'est à dire l'animation du slideshow.
Cette animation pourra être déclenchée :

- de façon automatique au chargement de la page ;
- lors du clic sur une miniature ou une puce ;
- lors du clic sur une flèche.

Cet effet sera alors réalisé en animant la propriété CSS *margin-left* de la *div#slides.*
Attention : on affiche en fait toujours le premier item de cette *div*, cela signifie qu'après avoir réalisé l'effet (ou avant selon le sens de déplacement) nous devons alors prendre le 1er ou dernier élément pour le placer en dernier ou au début.

Illustrons cela par un schéma (voir page suivante).

De plus, lors du clic sur une miniature (puce ou image), il se peut que nous devions avancer ou reculer de plusieurs images !

4
script

Déplacement vers la droite :

1/ Clic sur la flèche de droite

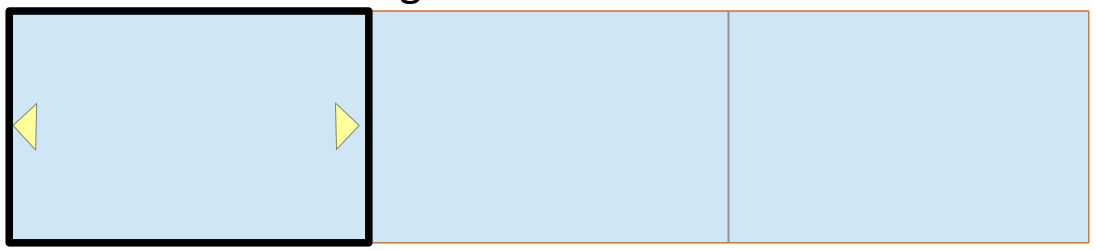
2/ Déplacement

3/ La 1ère image passe en dernier

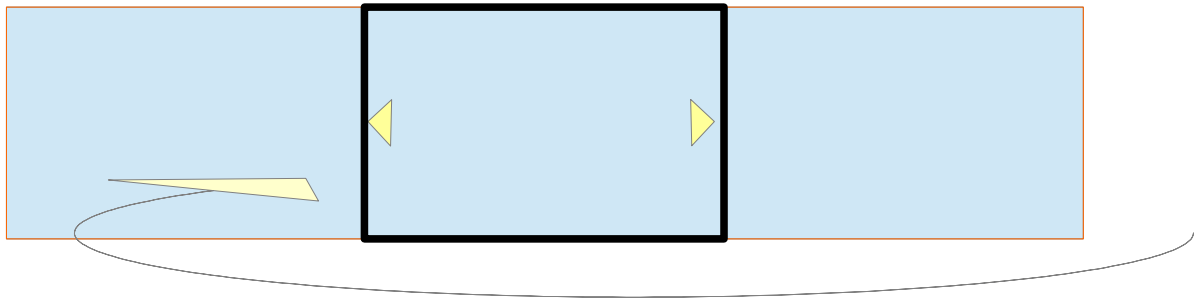
4 script

Déplacement vers la gauche:

1/ Clic sur la flèche de gauche



2/ La dernière image passe en premier



3/ Déplacement

4
script

Nous créons alors une nouvelle fonction *animSlide()* afin de mettre en place cela. Elle prendra en paramètre un argument qui sera un nombre correspondant au nombre de slides à avancer ou reculer (négatif) :

```
function animSlide(arg){
    var largeur = $("#slideshow").width();
    if(arg>0){
        $("#slides").animate(
            {"margin-left":-(arg*largeur)},
            datas.prefs.tIMG,
            function(){
                var items = $('#slides .item');
                var itemsAjoute = items.slice(0,arg);
                $("#slides").append(itemsAjoute);
                $("#slides").css("margin-left","0");
            }
        );
    } else if(arg<0){
        var derniersItems = $('#slides .item').slice(arg);
        $("#slides").prepend(derniersItems);
        $("#slides").css("margin-left",(arg*largeur)+"px");
        $("#slides").animate({"margin-left":0},datas.prefs.tIMG);
    }
}
```

4
script

Explications :

- *var largeur = $("#slideshow").width();*
  On récupère la largeur du slideshow puisque nous devrons déplacer la *div#slides* en fonction de celle-ci. De plus, cette largeur sera différente suivant les écrans.

- *if(arg>0){*
  On teste si le paramètre est positif (pour un déplacement vers la gauche).

- *$("#slides").animate(*
  *{"margin-left":-(arg*largeur)},*
  datas.prefs.tIMG,
  On anime alors le *margin-left* de la *div#slides* vers une valeur négative correspondant à la largeur du slideshow multiplié par le nombre de slides.
  On applique en durée d'animation la valeur *tIMG* définie dans le JSON.

- *function(){*
  var items = $('#slides .item');
  var itemsAjoute = items.slice(0, arg);
  $("#slides").append(itemsAjoute);
  $("#slides").css("margin-left","0");
  }

4
script

On exécute ensuite une fonction anonyme (passée en dernier paramètre de *animate()*).
Dans celle-ci on récupère toutes les *div* de classe « item », puis on récupère au sein de celles-ci les X éléments pour lesquels on a effectué le déplacement grâce à la méthode *slice()*.
Par exemple, si l'image affichée correspond à la 2ème image du slide et que l'on clique sur la 4ème puce, l'argument sera donc égal à 2 ; on déplace alors la *div#slides* de 2 fois la largeur du *slideshow*. Une fois ce déplacement effectué, on doit ensuite récupérer les 2 premiers items pour les placer en dernier.

Ces items étant récupérés on utilise la méthode *append()* pour les positionner en derniers enfants de la *div#slides*.
Note : si on effectue un *append()* d'éléments existants, ceux-ci sont alors supprimés de leur place initiale dans le DOM.

Enfin, ces éléments étant passés en dernier, on réinitialise le *margin-left* à 0.

Pour le cas de l'argument positif (un déplacement vers la droite), on effectue une animation similaire mais les derniers éléments doivent être placés en premier avant d'effectuer le déplacement.

4 script

Ajoutons maintenant les différentes exécutions de cette fonction.
1/ Lors du clic sur une flèche.
Nous devrons passer la valeur 1 ou -1 en paramètre de *animSlide()* puisque nous déplacerons alors d'une seule image le slideshow.
Dans la fonction *placeCommandes()*, nous modifions alors le tableau *fleches* ainsi :

```
var fleches = [['droite', 1],['gauche', -1]];
```

Et nous modifions la boucle :

```
$.each(fleches, function(index, val){
    var fleche = $("<div>");
    fleche.addClass("fleches");
    fleche.addClass(val[0]);
    slideshow.append(fleche);
    fleche.click(function(){
        animSlide(val[1]);
    });
});
```

Au clic sur l'une des flèches, notre slideshow se « déplace » d'une image vers la gauche ou la droite en fonction de la flèche cliquée.

4
script

2/ Lors du clic sur une miniature.
Ajoutez alors au sein de la boucle ***$.each(datas.imgs, …) ;*** le code suivant :

```
puce.click(function(){
    var indexEnCours = $("#slides .item").first().data('ref');
    var indexClique = $(this).data('ref');
    var decalage = indexClique - indexEnCours ;
    animSlide(decalage);
});
```

On récupère donc la valeur du *data-ref* de l'image affichée puis celui de la miniature (image ou puce) cliquée.
La différence entre les deux nous donne le nombre de slides à déplacer et donc la valeur à passer en paramètre de la fonction *animSlide()*.

Le clic sur une miniature est maintenant fonctionnel.

4
script

3/ Au chargement de la page.
On ajoute pour cela une exécution en boucle de cette fonction au sein de la fonction *init()*.
On ajoute aussi la pause de cette animation au survol du slideshow, si l'option *pause* du JSON est définie à *true*.
Note : le 3ème paramètre de la fonction de *setInterval()* correspond au paramètre que nous passons à la fonction *animSlide()* :

```
var delai = setInterval(animSlide, datas.prefs.duree, 1);
if(datas.prefs.pause){
    $("#slideshow").hover(function(){
        clearInterval(delai);
    },function(){
        delai = setInterval(animSlide, datas.prefs.duree, 1);
    });
}
```

4 script

Nous souhaitons maintenant afficher la légende et le bouton après l'animation du slide. Pour cela, nous créons une nouvelle fonction *afterAnim()* :

```
function afterAnim(){
    $("#slides .item").first().find('p, button').fadeIn(datas.prefs.tTXT);
}
```

Nous récupérons donc la 1ère *div.item* (celle visible) dans laquelle on sélectionne les balises *p* et *button* auxquelles on applique un *fadeIn()*. On passe en paramètre de cette dernière la durée définie dans l'option *tTXT* du fichier JSON.

Cette fonction doit maintenant être exécutée :

- au chargement de la page ;
- après chaque animation.

Nous modifions donc la function *init()* :

```
function init(obj){
    datas = obj;
    createSlider();
    placeCommandes();
    afterAnim();
    var delai = setInt
```

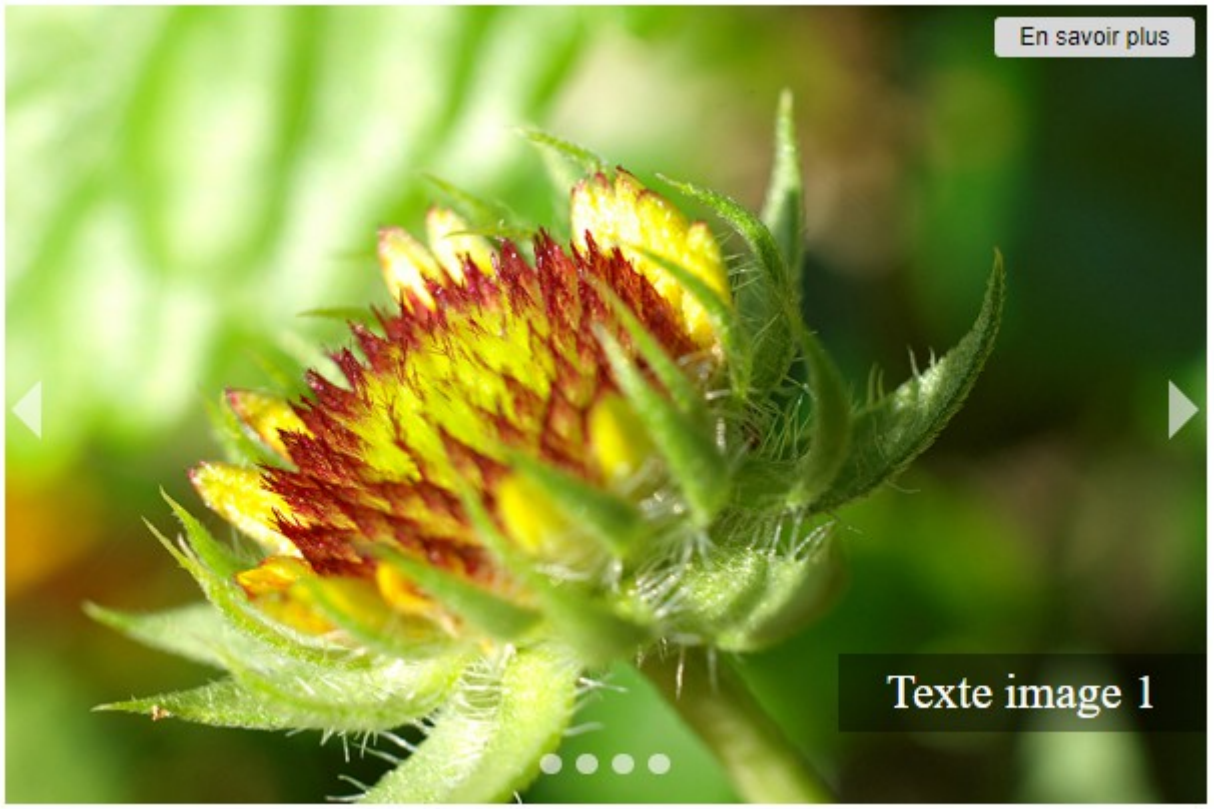
4
script

Ainsi que la fonction *animSlide()* ; dans celle-ci nous devons penser à masquer dans un 1er temps ce paragraphe et ce bouton afin de les faire réapparaître ensuite :

```
function animSlide(arg){
    var largeur = $("#slideshow").width();
    $("#slides .item").find('p,button').hide();
    if(arg>0){
        $("#slides").animate(
            {"margin-left":-(arg*largeur)},
            datas.prefs.tIMG,
            function(){
                var items = $('#slides .item');
                var itemsAjoute = items.slice(0,arg);
                $("#slides").append(itemsAjoute);
                $("#slides").css("margin-left","0");
                afterAnim();
            }
        );
    } else if(arg<0){
        var derniersItems = $('#slides .item').slice(arg);
        $("#slides").prepend(derniersItems);
        $("#slides").css("margin-left",(arg*largeur)+"px");
        $("#slides").animate(
            {"margin-left":0},
            datas.prefs.tIMG,
            afterAnim
        );
    }
}
```

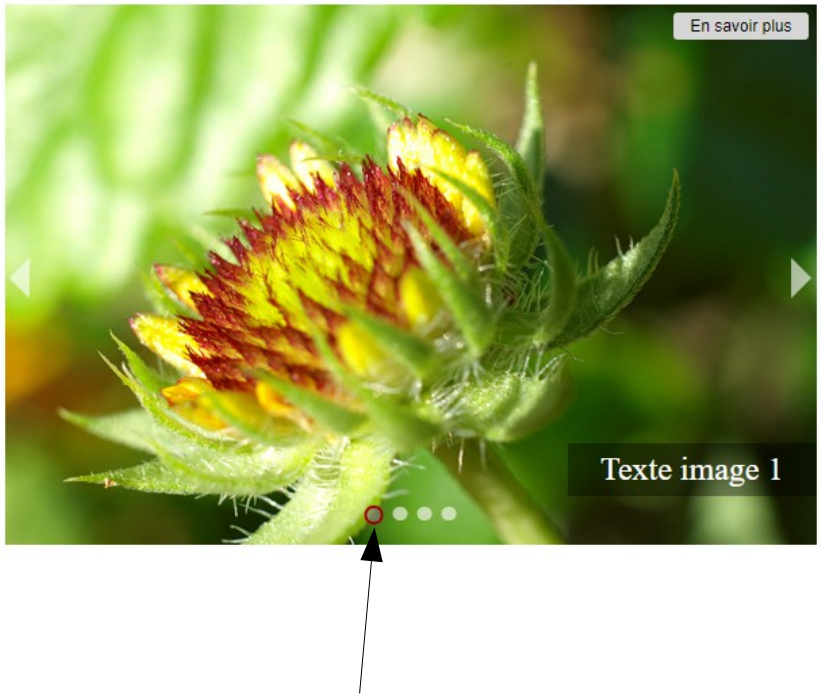
4
script

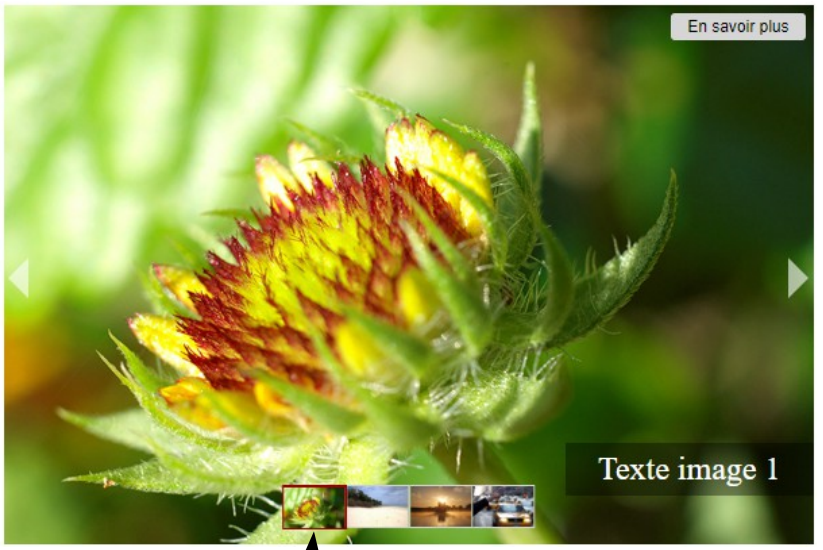
Testez ; le texte et/ou le bouton apparaissent bien après chaque animation :



4 script

Pour terminer ce slideshow, nous souhaitons marquer la miniature (image ou puce) correspondant à l'image affichée :





4 script

Nous allons gérer ce point par l'ajout / suppression d'une classe « active » sur la miniature correspondante.
Définissons tout d'abord le CSS de cette classe :

```
#slideshow .active{
    border: 2px solid #900;
    top:-1px;
    background-color: rgba(125,125,125,.6);
}
```

Nous profitons aussi de terminer le CSS en ajoutant une media query pour les petits écrans ; sur ces derniers on réduit la police et on masque les puces :

```
@media screen and (max-width:480px){
    #slideshow{
        font-size:0.625em;
    }
    #puces {
        display:none;
    }
}
```

4 script

L'ajout / suppression de la classe « active » se fera après l'animation du slide, donc au sein de la fonction *afterAnim()* que nous modifions ainsi :

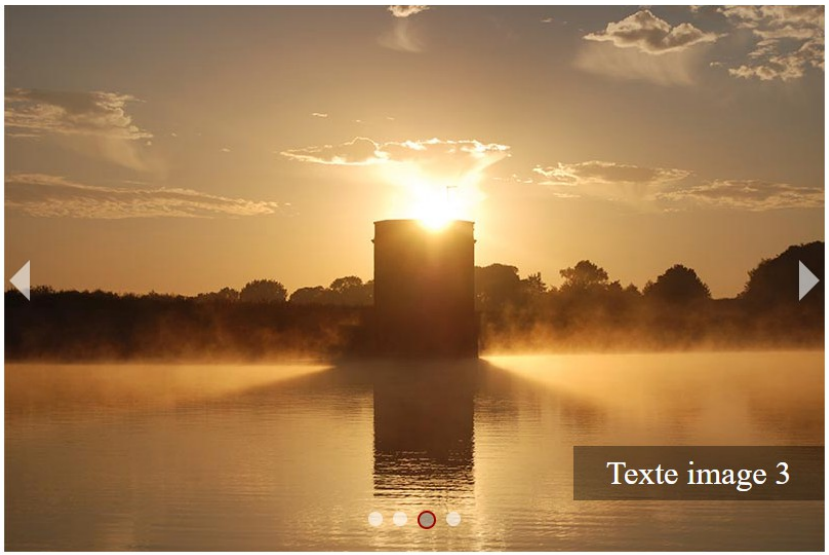
```
function afterAnim(){
    $("#puces .active").removeClass("active");
    var imageEnCours = $("#slides .item").first().attr('data-ref');
    $("#puces [data-ref='"+imageEnCours+"']").addClass("active");
    $("#slides .item").first().find('p, button').fadeIn(datas.prefs.tTXT);
}
```

Explications :

- *$("#puces .active").removeClass("active");*
  On retire la classe 'active' à tout élément qui la possède.
- *var imageEnCours = $("#slides .item").first().attr('data-ref');*
  On récupère la valeur de l'attribut *data-ref* de l'image affichée.
- *$("#puces [data-ref='"+imageEnCours+"']").addClass("active");*
  Puis on sélectionne l'élément enfant de *#puces* qui a pour valeur de *data-ref* celui de l'image affichée ; on lui applique alors la classe 'active' .
  Rappel : en CSS, par exemple le sélecteur *[data-ref='3']* nous permet de sélectionner tout élément ayant un attribut *data-ref* égal à '3'.

Conclusion

Notre slideshow est maintenant fonctionnel.



N'hésitez pas à l'améliorer selon vos souhaits !