CEFii
www.cefii.fr

Tutoriel jQuery - Carrousel

0
Introduction

Introduction

L'objectif de ce tutoriel est d'aborder des notions complémentaires JavaScript / jQuery et de mettre en pratique la création d'éléments.

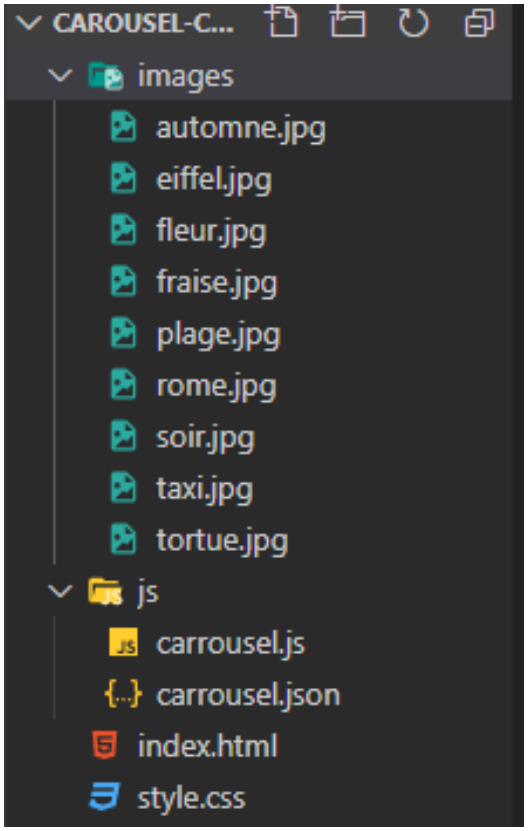
Pour cela, nous construirons le carrousel d'images suivant :

Le carrousel tourne sur lui-même automatiquement ou via un clic sur les flèches afin de visualiser les images les unes après les autres.
Au clic sur une image, celle-ci descend pour faire apparaître sa légende (texte).

1 Structure

Structure

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



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

2
JSON

fichier JSON

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

Dans ce fichier, nous renseignerons :

- une clé *prefs* permettant de paramétrer des options de préférences :
  - *dossier* : nom du dossier contenant les images
  - *auto* : activation de la rotation automatique
  - *time* : temps entre chaque rotation
  - *pausedOnHover* : activation de la mise en pause de la rotation automatique
- une clé *imgs* ayant pour valeur un tableau permettant de définir un objet pour chaque slide.
  Pour chacun de ces slides, on pourra définir les clés :
  - *file* : nom du fichier image
  - *text* : texte de la légende

2
JSON

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

carrousel.json

```json
{
    "prefs":{
        "dossier": "images",
        "auto": true,
        "time": 2000,
        "pausedOnHover": true
    },
    "imgs":[
        { "file" : "automne.jpg" , "text" : "C'est l'automne !" },
        { "file" : "eiffel.jpg" , "text" : "Tour Eiffel" },
        { "file" : "fleur.jpg" , "text" : "C'est beau la nature" },
        { "file" : "fraise.jpg" , "text" : "Mmmmmm" },
        { "file" : "plage.jpg" , "text" : "Les vacances" },
        { "file" : "rome.jpg" , "text" : "Le pays de la pizza" },
        { "file" : "soir.jpg" , "text" : "Joli coucher de soleil" },
        { "file" : "taxi.jpg" , "text" : "Hep taxi !" },
        { "file" : "tortue.jpg" , "text" : "Le tort tue ?" }
    ]
}
```

3
HTML

fichier HTML

Construisez le fichier *index.html* ainsi :

index.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>Carousel TD</title>
    <link rel="stylesheet" href="style.css">
    <script
        src="https://code.jquery.com/jquery-3.4.1.min.js"
        integrity="sha256-CSXorXvZcTkaix6Yvo6HppcZGetbYMGWSFlBw8HfCJo="
        crossorigin="anonymous"></script>
    <script src="js/carrousel.js"></script>
</head>
<body>
    <div id="carrouselContainer"></div>
</body>
</html>
```

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

4
script

fichier JS

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

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

carrousel.js

```
1  var datas = {};
2
3  $.getJSON('js/carrousel.json', init);
4
5  function init(obj){
6      datas = obj;
7      console.log(datas);
8  }
```

Résultat dans la console après exécution du fichier HTML :

```
▼ {prefs: {…}, imgs: Array(9)}
  ▶ prefs: {dossier: "images", auto: true, time: 2000, pausedOnHover: true}
  ▶ imgs: (9) [{…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}, {…}]
  ▶ __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#carrouselContainer :
conteneur principal

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

div#carrousel :
conteneur 3D du carrousel

div#circle :
conteneur direct des éléments

div.item :
contenant une balise *img* et un *span* pour le texte

Tutoriel jQuery - Carrousel

4
script

La structure HTML à obtenir sera donc :

```
<div id="carrouselContainer"> == $0
  <div id="carrousel">
    <div id="circle" style="transform: translateZ(-351.68px) rotateY(-92120deg);">
      <div class="item" data-index="0" style="transform: rotateY(0deg) translateZ(351.68px);">
        <span>C'est l'automne !</span>
        <img src="images/automne.jpg">
      </div>
      <div class="item" data-index="1" style="transform: rotateY(40deg) translateZ(351.68px);">…</div>
      <div class="item" data-index="2" style="transform: rotateY(80deg) translateZ(351.68px);">…</div>
      <div class="item" data-index="3" style="transform: rotateY(120deg) translateZ(351.68px);">…
      </div>
      <div class="item" data-index="4" style="transform: rotateY(160deg) translateZ(351.68px);">…
      </div>
      <div class="item" data-index="5" style="transform: rotateY(200deg) translateZ(351.68px);">…
      </div>
      <div class="item" data-index="6" style="transform: rotateY(240deg) translateZ(351.68px);">…
      </div>
      <div class="item" data-index="7" style="transform: rotateY(280deg) translateZ(351.68px);">…
      </div>
      <div class="item" data-index="8" style="transform: rotateY(320deg) translateZ(351.68px);">…
      </div>
    </div>
  </div>
  <div class="fleches droite"></div>
  <div class="fleches gauche"></div>
</div>
```

4 script

Afin de construire ces éléments HTML, nous créons une fonction *createCarrousel()* :

```
function createCarrousel(){
    var carrouselContainer = $('#carrouselContainer');
    var chemin = datas.prefs.dossier;
    var carrousel = $('<div>');
    carrousel.attr('id','carrousel');
    var circle = $('<div>');
    circle.attr('id','circle');
    carrousel.append(circle);
    carrouselContainer.append(carrousel);
}
```

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#carrouselContainer* dans laquelle nous ajoutons un nouvel élément *div* ayant pour id « carrousel » ayant lui-même un nouvel élément *div#circle*.

Exécutez cette fonction au sein de la fonction *init()* :

```
function init(obj){
    datas = obj;
    createCarrousel();
}
```

Pensez à vérifier dans l'inspecteur la présence de ces nouveaux éléments.

4 script

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

- un attribut personnalisé *data-index* pour lequel nous associons l'index ;
- une balise *img* à laquelle nous associons un attribut *src ;*
- une balise *span* pour le texte de la légende.

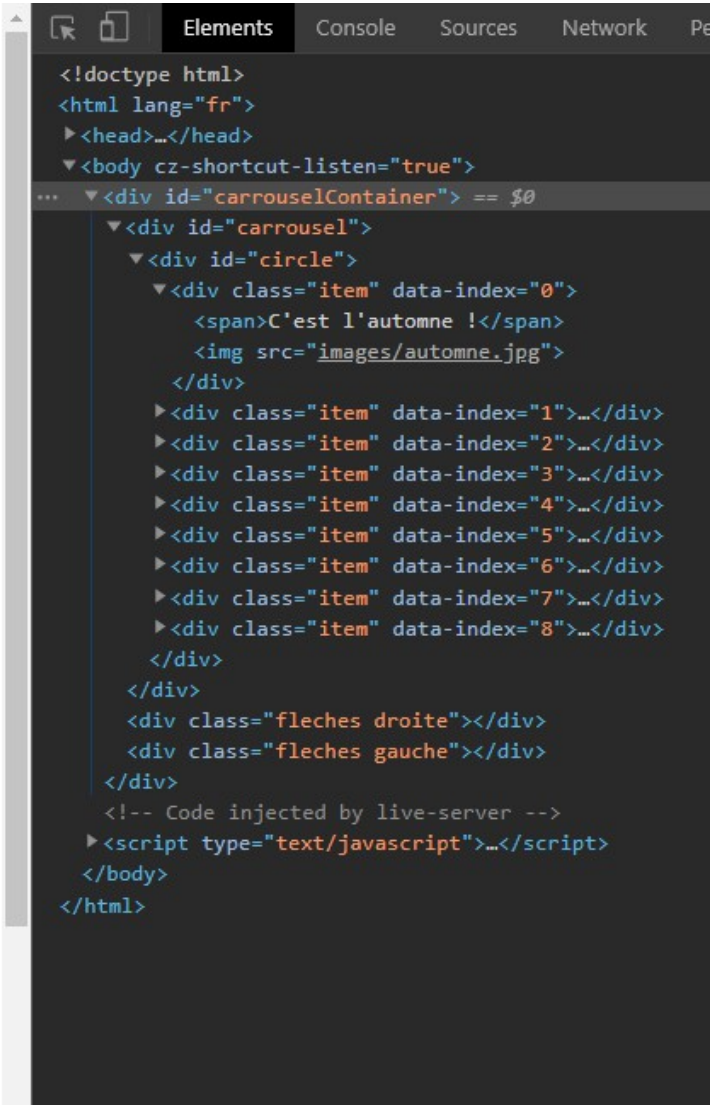
Nous ajoutons donc la boucle *each()* au sein de la fonction *createCarrousel()* définie de la manière suivante :

```
$.each(datas.imgs, function(index,val){
    var item = $('<div>');
     item.addClass('item');
     item.attr("data-index",index);
     var img = $('<img>');
     img.attr('src',chemin + "/" + val.file);
     var span = $('<span>');
     span.text(val.text);
     item.append(span);
     item.append(img);
     circle.append(item);
});
```

Tutoriel jQuery - Carrousel

4
script

Vérifiez ensuite dans le navigateur que les éléments sont bien insérés :



4
script

Voici le CSS final que nous utiliserons ; pour un confort de lecture, les préfixes n'ont pas été ajoutés :

style.css

```css
#carrouselContainer{
    max-width: 640px;
    height: 300px;
    margin: 0 auto;
    position: relative;
    perspective: 600px;
}

#carrousel{
    width: 40%;
    height: 100%;
    position: relative;
    margin: 0 auto;
    transform-style: preserve-3d;
    transform: rotateX(-15deg) translateY(100px);
}

#circle {
    width: 100%;
    height: 100%;
    position: absolute;
    transform-style: preserve-3d;
    transition: transform 0.5s;
}
```

Nous appliquons des dimensions au conteneur ainsi qu'une *position:relative* car les éléments enfants seront en *absolute*.
La *perspective* permet de mieux visualiser l'effet 3D.

Nous appliquons des dimensions au conteneur 3D des images que nous passons donc dans l'espace 3D. La transformation permettra d'incliner le carrousel afin de visualiser les images placées derrière.

Nous plaçons aussi la *div#circle* dans l'espace 3D et nous appliquons une *transition* pour la rotation future du carrousel.

4
script

```
26  #circle .item{
27      position: absolute;
28      width: 100%;
29      height: 100%;
30      cursor: pointer;
31  }
32
33  #circle span{
34      width: 100%;
35      top: 0;
36      font-size: 20px;
37      line-height: 40px;
38      height: 40px;
39      background-color: #900;
40      display: block;
41      position: absolute;
42      color: white;
43      text-align: center;
44  }
45
46  #circle .item img{
47      position: absolute;
48      width: 100%;
49      display: block;
50      top:0;
51      transition:top .4s;
52  }
```

Chaque *div.item* est positionnée en *absolute* afin d'être superposée à d'autres éléments.

Des styles « classiques » sont appliqués aux *span*.

On superpose les images aux *span* et on les place en haut de la *div* parente.
La *transition* sera appliquée lorsqu'on cliquera sur l'image pour faire apparaître le texte placé derrière.

4
script

```
54  #carrouselContainer .fleches{
55      position: absolute;
56      top: 50%;
57      transform: translate(0,-50%);
58      width: 0;
59      height: 0;
60      cursor: pointer;
61      border-top:20px solid transparent;
62      border-bottom:20px solid transparent;
63  }
64
65  #carrouselContainer .droite {
66      right: 5px;
67      border-left:20px solid rgba(0,0,0,0.6);
68  }
69
70  #carrouselContainer .gauche {
71      left: 5px;
72      border-right:20px solid rgba(0,0,0,0.6);
73  }
```

Nous construisons et positionnons enfin les futures flèches.

4
script

Nous obtenons :

En effet : pour le moment toutes les *div.item* sont superposées !

4
script

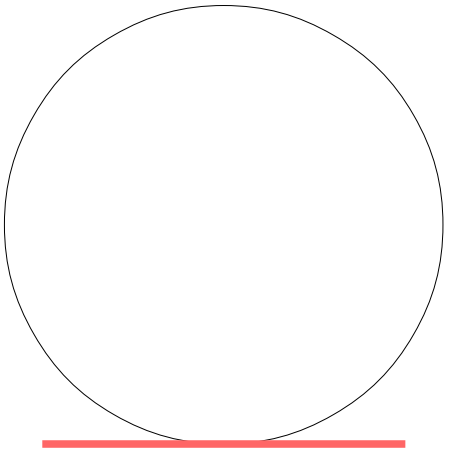
Avant d'effectuer le positionnement de chaque image, ajoutons les flèches.
Pour cela, nous créons une nouvelle fonction *placeCommandes()* :

```
function placeCommandes(){
    var carrouselContainer = $('#carrouselContainer');
    $.each([['droite',1],['gauche',-1]],function(index,val){
        var fleche = $('<div>');
        fleche.addClass('fleches');
        fleche.addClass(val[0]);
        carrouselContainer.append(fleche);
    });
}
```

Cette fonction est donc très similaire à celle utilisée dans le tutoriel précédent : nous bouclons sur un tableau contenant 2 tableaux ayant pour éléments un nom de classe à ajouter et un nombre nous permettant d'effectuer par la suite l'exécution d'une fonction pour la rotation et de lui passer un sens de rotation.
Exécutez ensuite cette fonction depuis *init()* .
Résultat:

4 script

Nous devons maintenant positionner chaque *div.item* tout autour du « cercle » car pour le moment nous avons en fait, **vu du dessus** :

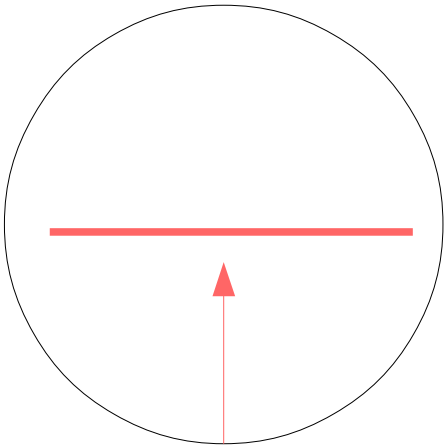


* Le cercle n'est pas « réel » car la *div#circle* reste un rectangle.

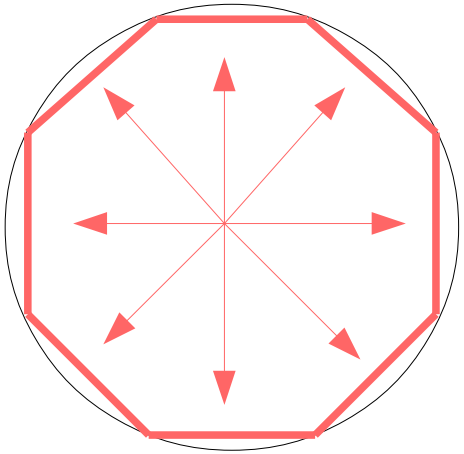
On positionnera donc les *div.item* tout autour de ce cercle fictif car elles sont pour le moment toutes superposées en avant du cercle (ici en rouge).

4
script

Pour le positionnement de nos *div.item* nous devons partir du centre ; pour cela nous appliquerons un *translate* à la *div.circle* (qui contient donc tous les items) :



Ensuite chaque *div.item* devra être tournée puis renvoyée à l'extérieur du « cercle » :



4 script

Plusieurs paramètres sont variables ici :

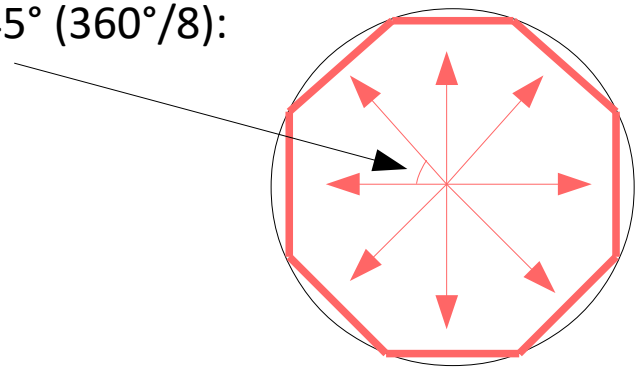
- la largeur des images qui correspondra en fait à celle de la *div#carrousel* .
  On pourra donc récupérer cette valeur facilement :

  var largeurImage = $("#carrousel").width();

- l'angle de rotation des images qui sera en fonction du nombre d'images (par exemple si on ne renseigne que 4 images dans le JSON, on obtiendra un carré et non pas un octogone).
  Cet angle correspond donc à 360° divisé par le nombre d'images (nombre d'éléments du tableau *datas.imgs*) :

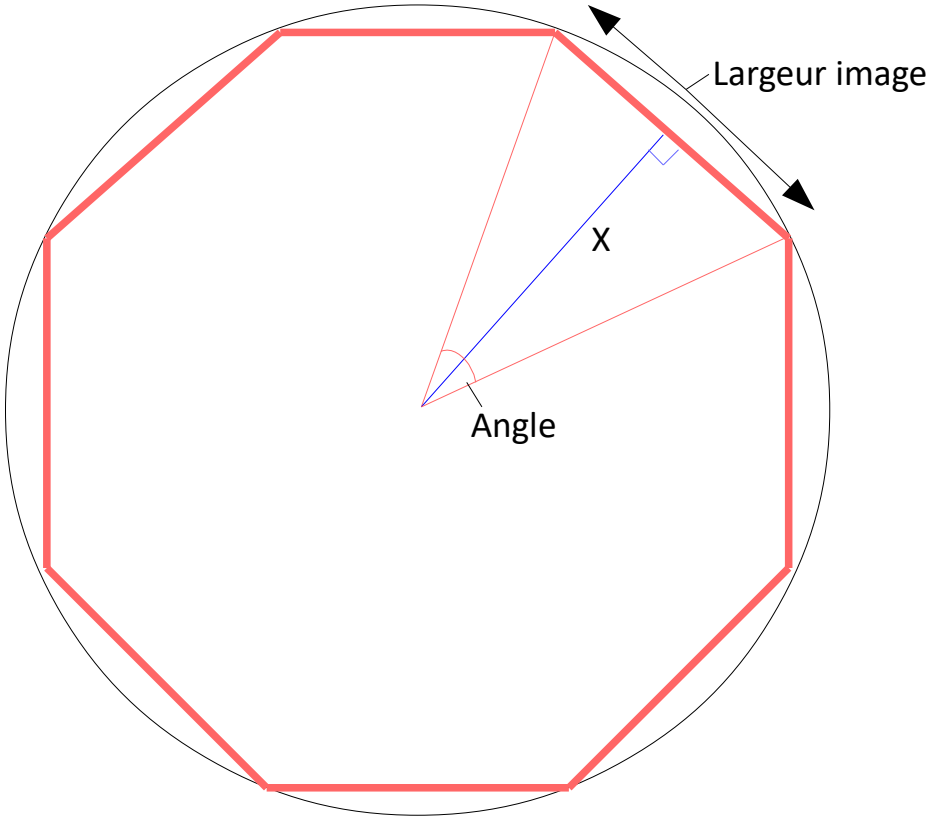
  var nbreImages = datas.imgs.length;
  angleImg = 360 / nbreImages;

  Par exemple, si renseigne 8 images dans le JSON, l'angle de chaque image sera de 45° (360°/8):



4
script

- la distance depuis le centre (le rayon du cercle) qui dépend de ces 2 valeurs précédentes. Pour celle-ci nous allons donc effectuer un calcul plus complexe. Pour rappel nous avons notre inconnue X et les éléments connus suivants :



4 script

Nous allons pouvoir effectuer ici un calcul de trigonométrie puisque nous sommes dans le cadre d'un triangle rectangle.
Nous avons :

- l'angle : angle de rotation divisé par 2 ;
- le côté opposé : largeur de l'image divisé par 2.

Nous recherchons donc le côté adjacent de l'angle, ce qui nous permet d'utiliser la tangente de l'angle.
Rappel :

$\tan(\alpha)$ = côté opposé / **côté adjacent**

Appliquons ce calcul à notre script :

$\tan(\alpha/2)$ = (largeurImage / 2) / **distanceImage**
<=> **distanceImage** = (largeurImage / 2) / $\tan(\alpha/2)$

En JavaScript, il est possible de calculer la tangente d'un angle à partir de l'objet *Math* mais cet angle doit être exprimé en radians et non en degrés :
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Objets_globaux/Math/tan

4 script

En radians, 2π correspondent à 360°.
Recalculons alors notre angle α :

α = 2π / nombreImages

donc α / 2 = π / nombreImages

Notre calcul devient donc :

distanceImage = (largeurImage / 2) / tan(π / nombreImages)

En JavaScript, on peut accéder à la valeur de π ainsi : *Math.PI* .

Notre calcul final en JS est donc le suivant :

distanceImage = (largeurImage/2) / Math.tan(Math.PI/nombreImages)

Nous arrondirons ce résultat grâce à *toFixed()* :

distanceImage = ((largeurImage/2)/Math.tan(Math.PI/nombreImages)).toFixed(2);

4 script

Nous pouvons enfin créer une nouvelle fonction *placeImages().*
Dans celle-ci, la rotation doit être exprimée en degrés.
De plus, les variables *angleImg* et *distanceImage* seront utilisées dans la fonction qui effectuera la rotation (animation) du carrousel. Ces 2 variables sont donc déclarées en global en début de script :

carrousel.js

```
var datas = {};
var distanceImage;
var angleImg;

$.getJSON('js/carrousel.json', init);
```

```
function placeImages(){
    var nbreImages = datas.imgs.length;
    angleImg = 360 / nbreImages;
    var largeurImage = $("#carrousel").width();
    distanceImage = ((largeurImage/2)/Math.tan(Math.PI/nbreImages)).toFixed(2);
    $("#circle")
        .css("transform","translateZ(-"+distanceImage+"px)");
    $("#circle .item").each(function(index){
       $(this)
           .css("transform","rotateY("+index*angleImg+"deg) translateZ("+distanceImage+"px)");
    });
}
```

L'angle de rotation de chaque item correspond à l'angle de rotation entre chaque item multiplié par l'index de celui-ci.

4
script

Ajoutez l'exécution de cette nouvelle fonction à la fin de la définition de la fonction *createCarrousel()* :

```
function createCarrousel(){
    var carrouselContainer = $('#carrouselContainer');
    var chemin = datas.prefs.dossier;
    var carrousel = $('<div>');
    carrousel.attr('id','carrousel');
    var circle = $('<div>');
    circle.attr('id','circle');
    carrousel.append(circle);
    carrouselContainer.append(carrousel);
    $.each(datas.imgs, function(index,val){···
    });
    placeImages();
}
```

On obtient :

4
script

Nos éléments étant positionnés, nous pouvons maintenant effectuer les animations.
Tout d'abord, lors du clic sur une image, nous souhaitons descendre celle-ci afin de faire apparaître la légende positionnée derrière (le *span*). Un 2nd clic viendra remonter l'image.
Nous allons donc pouvoir modifier la propriété CSS *top* de l'image lors du clic.
Une condition nous permettra de vérifier la valeur du *top* (différent de 0) afin de la remonter dans ce cas.
Notre nouvelle fonction *clickImg()* est donc la suivante :

```
function clickImg(){
    var top = parseInt($('img', this).css("top"));
    if(top !=0 ){
        $('img', this).css('top','0');
    } else{
        $('img','.item').css('top','0');
        $('img', this).css('top','40px');
    }
}
```

4 script

Nous plaçons l'exécution de cette fonction lors du clic sur une image, donc au sein de la boucle *each()* de la fonction *createCarrousel()* :

```
$.each(datas.imgs, function(index,val){
    var item = $('<div>');
     item.addClass('item');
     item.attr("data-index",index);
     var img = $('<img>');
     img.attr('src',chemin + "/" + val.file);
     var span = $('<span>');
     span.text(val.text);
     item.append(span);
     item.append(img);
     item.click(clickImg);
     circle.append(item);
});
```

Résultat lors du clic :

4 script

Ensuite nous mettons en place la rotation du carrousel en définissant une nouvelle fonction *rotation()* . Cette fonction prendra en paramètre un argument permettant de déterminer le sens de rotation.
L'angle de rotation général correspond donc à la variable *angleImg* calculée au sein de la fonction *placeImages()* ; cette variable étant en global, nous pouvons récupérer ici sa valeur.

A chaque rotation, nous devrons ajouter la valeur de cet angle à celle de l'angle de rotation déjà appliqué grâce à l'opérateur += .
Nous créons pour cela une nouvelle variable globale en début de script initialisée à 0 :

carrousel.js

```
1   var datas = {};
2   var distanceImage;
3   var angleImg;
4   var angleEnCours = 0;
```

Et nous créons notre fonction :
→ nous devons réappliquer le *translateZ* défini en CSS afin de ne pas l'annuler.

```
function rotation(arg){
    angleEnCours += (angleImg*arg);
    $("#circle").css('transform','translateZ(-'+distanceImage+'px) rotateY('+angleEnCours+'deg)');
}
```

4 script

Exécutons cette fonction lors du clic sur les flèches :

```
function placeCommandes(){
    var carrouselContainer = $('#carrouselContainer');
    $.each([['droite',1],['gauche',-1]],function(index, val){
        var fleche = $('<div>');
        fleche.addClass('fleches');
        fleche.addClass(val[0]);
        fleche.click(function(){
            rotation(val[1]);
        });
        carrouselContainer.append(fleche);
    });
}
```

Et de manière automatique au chargement de la page (si l'option *auto* est à *true*).
Nous ajoutons aussi la pause au survol si cette option est elle aussi à *true* :

```
function init(obj){
    datas = obj;
    createCarrousel();
    placeCommandes();
    if(datas.prefs.auto){
        delai = setInterval(rotation, datas.prefs.time, -1);
        if(datas.prefs.pausedOnHover){
            $("#circle").hover(function(){
                clearInterval(delai);
            },function(){
                delai = setInterval(rotation,datas.prefs.time, -1);
            });
        }
    }
}
```

4
script

Notre carrousel est maintenant fonctionnel mais si on réduit la fenêtre du navigateur, on obtient :

En effet, le rayon du cercle (distance des images) est calculé par rapport à largeur de la *div#carrousel*, or lors du rétrécissement de la fenêtre, celle-ci a aussi été modifiée.
Nous ajoutons alors une nouvelle exécution de la fonction *placeImages()* lors de la modification de la largeur de la fenêtre :

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

Conclusion

Notre carrousel est maintenant fonctionnel. N'hésitez pas à l'améliorer selon vos souhaits !

A travers ces différents tutoriels, nous avons pu aborder la notion de JSON, la création d'éléments en JS (jQuery), les calculs etc.

Nous avons pu constater aussi que les schémas étaient très importants pour l'élaboration de scripts, n'oubliez pas ces 2 outils qui sont le papier et le crayon.
Bon développement !