

5XCLI

# Script coté client



# 1

## Introduction



# Présentation



Bienvenue à la formation «Script cotée client» (5XCLI dans l'horaire) qui se déroulera durant le premier semestre.

Le module est de 120 périodes, soit ± 30 jours de formations, voici les dates et heures :

Jour	Heures	Date
Mardi	18h00-21h30	
Vendredi	18h00-21h30	

## 1. Dossier pédagogique

### 1.1. Finalités générales

Conformément à l'article 7 du décret de la Communauté française du 16 avril 1991 organisant l'Enseignement de promotion sociale, cette unité de formation doit :

- concourir à l'épanouissement individuel en promouvant une meilleure insertion professionnelle, sociale et culturelle ;
- répondre aux besoins et demandes en formation émanant des entreprises, des administrations, de l'enseignement et d'une manière générale des milieux socio-économiques et culturels.

### 1.2. Finalités particulières

Cette unité de formation vise à permettre à l'étudiant :

- d'insérer des scripts clients dans des pages web auxquelles ils ajouteront des possibilités d'interaction ou d'animation ;
- de mettre en œuvre des notions de programmation orientée objet dans des scripts clients ;
- d'accroître la richesse de ses réflexions techniques et ses compétences en communication, en organisation, en observation.

## 2. Programme

### 2.1. L'étudiant sera capable :

Face au matériel et au logiciel adéquats et face à une structure informatique opérationnelle connectée à Internet, disposant des logiciels appropriés et de la documentation nécessaire, en utilisant le vocabulaire technique et l'orthographe adéquate, et en respectant les normes et standards en vigueur,

- d'identifier, dans une page web, les éléments impliquant l'usage d'un script client ;
- d'analyser un script client en termes de:
  - définition des variables et des objets,
  - structures conditionnelles et itératives,
  - fonctions et de procédures,
  - structures interactives (gestion des évènements,...),
  - ... ;
- d'exploiter un script client dans une page web ;
- de modifier et de créer un script et de l'intégrer dans une page web ;
- de décrire et de caractériser objets, propriétés et méthodes ;
- de déterminer les événements auxquels les éléments de la page doivent réagir ;
- de traduire sous forme de commentaires, de schémas, de dessins... les éléments nécessaires à la résolution d'un problème posé (structures procédurales, interactives, animations, objets...) ;
- de mettre en œuvre la résolution d'un problème posé au moyen du langage client choisi ;
- d'exploiter le côté orienté objet du langage choisi :
  - les classes prédéfinies et leurs composants (Windows, document, cookie...),
  - la définition de classes et leur instanciation,
  - ... ;
- d'utiliser, dans le langage choisi, les variables, les structures conditionnelles, les structures itératives, les tableaux, l'affichage dans une page web... ;
- d'exploiter la notion d'expression régulière (validation de formulaires...) ;
- d'exploiter des données structurées en XML (Extensible Markup Language), en JSON (JavaScript Object Notation)... contenue dans un fichier externe ;
- de décrire et de mettre en œuvre des technologies entrant dans le développement d'applications web dynamiques et animées tel que AJAX (Asynchronous Javascript and XML)... ;
- de choisir et d'exploiter une bibliothèque tierce telle que jQuery, MooTools... en vue du développement de scripts spécifiques pour RIA (interfaces riches), transmissions asynchrones... ;
- d'identifier des erreurs de programmation au moyen d'outils ou de techniques de débogage et d'y apporter une solution pertinente ;
- d'utiliser à bon escient la documentation disponible.

## 2.2. CAPACITÉS TERMINALES

### Pour atteindre le seuil de réussite, l'étudiant sera capable :

face au matériel et au logiciel adéquats et face à une structure informatique opérationnelle connectée à Internet, disposant des logiciels appropriés et de la documentation nécessaire, en utilisant le vocabulaire technique et l'orthographe adéquate, et en respectant les normes et standards en vigueur,

et au départ d'un cahier des charges contenant un projet de pages web interactives et animées

- de créer et d'exploiter des scripts clients basés sur des classes prédefinies ;
- de créer et d'exploiter ses propres classes ;
- de créer et d'exploiter des scripts basés sur une bibliothèque tierce

### Pour la détermination du degré de maîtrise, il sera tenu compte des critères suivants:

- des techniques de programmation utilisées,
- la pertinence des commentaires dans le code,
- la lisibilité du code,
- le degré d'autonomie atteint.

## 3. Évaluation

La formation sera découpée en une série de chapitres, à chaque chapitre une évaluation sera réalisée, en fin d'UF une évaluation certificative sera faite

L'attribution des points se fera suivant les critères suivants :

Critères	Points
Chapitre	40
Certificative	60

## 4. Contenu du cursus

Après une rapide révision des bases du JavaScript, nous allons apprendre à connaître les différents objets du navigateur, créer ses propres class et utiliser différentes librairies existantes.

## 5. Conventions

Dans le cadre de la rédaction de ce syllabus, j'ai défini une série de règles de mise en page que voici:

### 5.1. Le code dans le texte.

Le texte du langage courant (le français une fois) sera écrit normalement, le langage de programmation sera lui mis en **courrier** et dans une autre police de caractère.

Ex : Le transfert du contenu de la variable `prixTvaComprise` sera le résultat du calcul ...

## *5.2. Programme complet ou extraits.*

Les codes complets sont alignés à gauche, même s'il est entrecoupé de commentaires explicatifs. Les extraits de code seront eux décalés vers la droite.

Exemple de programme complet :

```
<script>
// affiche un message
document.write("Salut");
?>
```

Exemple d'extrait de code :

```
alert("Ajoute de l'utilisateur dans la bd");
// gestions des erreurs
```

## *5.3. Les commentaires dans le code*

Les commentaires dans ce syllabus seront écrits en italique, ce qui ne sera pas possible dans le code que vous composerais.

Ex.:

```
/* Affichage du mot Bonjour à l'écran */
document.write("Bonjour"); // action d'affichage
```

## *5.4. Ligne de code trop longue.*

Les lignes de code qui dépasse des limites de la page seront alignées à droite à partir de la 2<sup>ème</sup> ligne.

Cette typographie signifie donc que vous devez taper le tout sur une seule ligne.

Ex.:

```
document.write("Liste des ".groupes." qui ont réalisé un chiffre de vente pour la région européenne du nord de plus de ".
$euro."€ pour l'année ".annee);
```

## **6. Pourquoi ce document n'est-il pas corrigé ? Parce que ! L'ordre des lettres...**

Sleon une édtue de l'Uvinertisé de Cmabrigde, l'odrre des Itteers dans les mtos n'a pas d'ipmrotncae, la suele coshe ipmrotnate est que la pmeirère et la drenière siot à la bnnoe pclae. Le rsete puet êrte dnas un dsérorde ttoal et vuos puoevz tujoruos Irie snas porlblème. C'est prace que le creaveu hmauin ne lit pas chuaqe ltetra elle-mmême, mias le mot cmome un tuot. La peruve..... Arlos ne veenz plus m'enrever aevc les corerticons otrah-hgropquies de mes syllabus ! Hihi.

# 2

## Rappel



### 1. Un peu de culture générale :

JavaScript a été inventé par *Netscape Communications* et baptisé *LiveScript 1.0*. Lorsque *Java* est devenu populaire, le nom a été transformé en *JavaScript*. Il a fait sa première apparition dans le navigateur *Netscape* version 2.0. Considérant l'impact et l'utilité de JavaScript, *Microsoft* (encore lui) a produit sa propre version sous le nom de *Jscript 1.0*, que l'on trouve à partir du navigateur *Internet Explorer 3.0*.

Nescape a rendu publiques les définitions de *JavaScript* et, par conséquent, a trouvé un terrain d'entente avec Microsoft et d'autres éditeurs pour créer un standard neutre. L'ECMA (European Computer Manufacturers Association), un organisme de standardisation dont le siège est en Suisse, a édité une version standard en 1997, sous le nom d'ECMAScript.

Après moult évolutions, les deux acteurs principaux du marché des navigateurs clament haut et fort que leurs produits sont très proches du standard ECMAScript. À vous de voir ???

## 2. Les outils

Il vous faut un simple éditeur pour visualiser et éditer vos sources JavaScript. Celui que vous avez utilisé pour l'HTML fera l'affaire. DreamWeaver en mode source notepad++ ou bbedit, gedit, je vous conseil brakets ou adobe edge, bref ! n'importe quel outil qui traite du code.

Il vous faut un navigateur pour JavaScript côté client. Vous devez vous assurer la comptabilité de vos scripts avec les différents navigateurs (moins de problèmes qu'il y a quelques années).

*Exemple*

```
<html>
<body>
<script>
    document.write('Bonjour à toi !\n') ;
</script>
</body>
<html>
```

## 3. Interpréteur

### 3.1. Code dans un document HTML

L'insertion de scripts JavaScript dans un document HTML peut être considérée sous deux aspects : l'emplacement de ces scripts dans le code HTML et les actions réalisées par ces scripts. Nous allons voir qu'il n'est possible d'insérer des scripts JavaScript qu'en certains endroits bien précis du document HTML

Puisque JavaScript ne peut être placé qu'à certains endroits d'un document, les possibilités d'action sur le document ou sur le navigateur qui l'affiche sont limitées. Voici ce que l'on peut attendre de JavaScript dans un document HTML :

- Agir sur la mise en page du document, pendant que les balises HTML et leur contenu sont en phase de chargement par le navigateur ;
- Agir sur le nombre de fenêtres ou d'objets de type fenêtre affichés par le navigateur ;
- Déetecter les actions de l'utilisateur pour créer une véritable interactivité ;
- Réaliser des automatismes simples ainsi que des tâches en retour ;
- Enrichir les formulaires de fonctionnalités que n'offre pas HTML ;

- Poursuivre l'action de l'utilisateur dans applets, des plug-ins et d'autres produits externes incorporés dans le document.

Un navigateur est une application hôte qui héberge un interpréteur JavaScript et dispose d'un interpréteur imbriqué dans son propre code. De la même façon, une page web peut comporter des scripts JavaScript insérés dans son code HTML.

### *3.2. La balise <script>*

Script est l'élément qui permet de créer un lien entre les langages JavaScript et HTML. N'importe quelle quantité de code JavaScript peut-être placé entre les balises `<script>` et `</script>` de cet élément.

#### *Exemple 1 (le plus cool)*

```
<html>
<head>
  <script>
    var exemple=1 ;
  </script>
</head>
</html>
```

#### *Exemple 2*

```
<script > exemple=2 ;</script>
```

#### *Exemple 3*

```
<script language= »JavaScript1.1 »> exemple=3 ;</script>
```

#### *Exemple 4*

```
<script language= »JavaScript1.2 »> exemple=4 ;</script>
```

#### *Exemple 5*

```
<script src= "mesfonctions.Js"></script>
```

#### *Exemple 6*

```
<script type= "text/JavaScript"> exemple=6 ;</script>
```

Les exemples 1 à 4 montrent l'utilisation la plus courante, que l'on peut qualifier de JavaScript intégré. Qui sont exécutées immédiatement lorsque le navigateur les détecte au cours de sa lecture.

L'exemple 5 illustre la façon dont un script JavaScript peut être récupéré à partir d'un autre fichier. Ce procédé est particulièrement utile si de nombreuses fonctions JavaScript doivent être employées dans plusieurs documents HTML.

### *3.3. Script intégré dans un document HTML*

Cet exemple illustre la manière d'intégrer du code JavaScript en recourant à l'utilisation d'une fonction. Toutes les indentations du texte n'ont d'autre propos que de le rendre plus lisible.

```
<html>
<head>
```

```

<script>
    function meteo() {
        if ( !Math.random ) { // n'existe pas dans N 2.0
            document.write('<PRE> --- Je ne sais pas ---</PRE>');
        } else if ( Math.floor((Math.random()*2)) == 0 ) {
            document.write('<STRONG> --- Plein soleil ---</STRONG>\n');
        } else {
            document.write(' <EM> Drache nationale </EM>\n');
        }
    }
</script>
</head>
<body>
    <p>Quel temps pour aujourd'hui ?</p>
    <script>
        meteo();
    </script>
    <p>fin de soirée</p>
</body>
</html>

```

Dans l'en-tête de cet exemple, matérialisé par les balises `<head>` et `</head>`, vous voyez un script intégré qui n'a d'autre utilité que de créer une fonction. Nous voyons ensuite, dans le corps du document (`<body>...</body>`), un script intégré qui appelle la fonction créée dans l'en-tête. Cette dernière provoque l'exécution de la commande `document.write()`, dont le résultat est une chaîne de caractères qui s'ajoute au texte HTML.

Le document HTML contiendra soit ceci :

```

<p> Quel temps pour aujourd'hui ?</p>
<strong>--- Plein soleil ---</strong>
<p>fin de soirée</p>

```

Soit cela :

```

<p> Quel temps pour aujourd'hui ?</p>
<em> Drache nationale </em>
<p>fin de soirée</p>

```

Avec le navigateur comme Nescape Navigator 2.0 (juste pour le fun!)

```

<p> Quel temps pour aujourd'hui ?</p>
<pre> --- Je ne sais pas --- <pre>
<p>fin de soirée</p>

```

### 3.4. Où placer les `<script>` ?

Un document HTML comporte généralement soit un élément `head` et un élément `body`, soit un élément `head` et un élément `frameset`. Il y a trois endroits où l'on peut raisonnablement placer des balises `<script>` : dans l'en-tête `<head>...</head>`, dans le corps du document `<body>...</body>` où bien après `</head>`, mais avant `<body>` ou `<frameset>`.

L'en-tête ne contient théoriquement aucun contenu affichable tel quel à l'écran, mais seulement des métainformations sur le document. Toujours en théorie, le code de l'en-tête est entièrement lu par le navigateur avant l'affichage.

fichage des éléments du corps de document. En conséquence, c'est un bon emplacement pour y mettre des instructions JavaScript d'initialisation ou de paramétrage invisibles pour les utilisateurs ; en particulier, les déclarations de fonctions (ex. : ta\_soif()), de variables. En revanche, on ne devrait pas y voir apparaître d'instructions du type document.write(), qui agit directement sur l'affichage à l'écran.

### 3.5. Pièges à éviter

Le stockage de code JavaScript dans un document est en soi d'une grande simplicité. L'incorporation de JavaScript dans un document HTML ayant un autre genre de contenu et de format peut cependant conduire à des problèmes.

Essayons tout cela ...

```
<html> <body>
<script>
var warning = 'ne jamais mettre </script> dans une chaîne de caractères ';
</script>
</body> </html>
```

Une solution au problème :

```
var astuce = '<SCR'+ 'IPT'
var warning = 'ne jamais mettre '+astuce+'dans la chaîne de caractères' :
```

Autre problème :

```
<html> <body>
<script>
document.write('un<deux') ;
</script>
</body> </html>
```

Solution :

```
document.write('un&lt ;deux') ;
```

## 4. Tour d'horizon

### 4.1. Les variables

Vous pouvez stocker des informations dans des scripts JavaScript dans des entités appelées variables.

Les variables ne sont qu'un espace réservé à l'intérieur de la mémoire de l'ordinateur. Elles sont référencées par un nom qui doit obéir à certaines règles.

- La première lettre ne peut être un chiffre.
- L'on ne mette d'espace dans un nom de variable (prenez le caractère « \_ » si vous utilisez un nom composé).
- Évitez l'utilisation de caractère spécial ex. \$%£ etc ...
- Évitez des noms qui se différencient seulement par une majuscule minuscule ex. Fred & fred.
- L'on ne peut utiliser des noms de variable qui existe déjà comme mots clés du JavaScript

La déclaration d'une variable se fait avec le mot clé "var" (il n'est pas obligatoire, mais très conseillé pour une meilleure lisibilité du code).

### Exemple

```
var nom ;  
var nouveau_prix = 10,69 ;  
adresse = "23 chaussée de wavre"; >> a déconseiller  
var premier = 1 ; second = 2 ;
```

## 5. Les types de données

### 5.1. Booléen

La variable de type booléen n'a que deux valeurs possibles : True ou False.

### 5.2. Nombre

Le type nombre est utilisé pour des nombres à virgule flottante, qu'ils aient un ou plusieurs chiffres après la virgule, de signe positif ou négatif. Il existe quatre notations différentes :

- Exponentielle              var résultat = 2.55e+2 ;
- Hexadécimale              var résultat = 0xFF ;
- Octale                      var résultat = O377 ;
- Décimale                    var résultat = 256 ;

### 5.3. String

Le type chaîne est utilisé pour les chaînes de caractères. Quelques caractères spéciaux peuvent être stockés dans une variable de type string ; il commence tout par «\» comme le symbole «\n» qui ajoute un retour à la ligne.

#### 5.3.I.Tableau des symboles

Séquence	Produit
\b	Backspace
\t	Tabulation
\n	Passer à la ligne
\f	Form feed
\r	Enter
\“	Double quote
\’	Simple quote
\\\	backslash

\r	Enter
\“	Double quote
\’	Simple quote

Les strings doivent être intégré entre-deux " "

## 6. Les tableaux

Les tableaux vous permettent de stocker plusieurs données différentes dans une seule variable. Généralement , les données ainsi stockées ont un rapport entre elles. L'utilisation d'un tableau vous évite de déclarer plusieurs variables avec des noms similaires.

*Exemple :*

```
var nom = new Array(26) ;
nom[1] = "Paul" ;
nom[10] = "Géraldine" ;

var jours = new Array("Dimanche", "Lundi", "Mardi" ...) ;
var a = 4 ;
document.write(jours[a]) ;
document.write(nom[10]);
```

**Attention** : lorsque l'on déclare un tableau de taille 26, la première cellule n'est pas la 1, mais la 0 ...

La taille d'un tableau peut être augmentée en ajoutant un élément avec un indice supérieur. L'indice n'est pas obligatoirement un chiffre, l'on peut utiliser du texte.

Un tableau peut avoir plusieurs dimensions ; voici un exemple en deux dimensions :

```
var articles = new Array(2) ;
articles[0] = new Array(10) ;
articles[1] = new Array(10) ;
articles[0,0] = "Pomme" ;
articles[0,1] = "Poire" ;
articles[0,2] = "Lemon" ;
articles[1,0] = "10,52" ;
articles[1,1] = "12,35" ;
articles[1,2] = "25,23" ;
```

## 7. Commentaires

Les commentaires sont des annotations ajoutés par le programmeur (vous peut-être)

```
// commentaires
var = bon      // Variable qui ne servira à rien
```

## 8. Expressions et test conditionnel

Nous avons besoin de manipuler les données ou d'effectuer des tests sur leurs valeurs. De telles tâches peuvent être effectuées grâce aux expressions et tests.

Les expressions combinent plusieurs valeurs en une seule, tandis que les tests comparent plusieurs valeurs renvoient une valeur booléenne comme résultat.

### 8.1. Opérateur arithmétique

Il s'agit des opérateurs habituels utilisés en mathématiques

- Addition +
- Soustraction -
- Division /
- Multiplication \*
- Reste division %

```
Euro= 10*40.3399 ; total=5+10 ; reste=2%3 ; solde=125-58 ;
```

### 8.2. Opérateur relationnel

Ils sont utilisés pour des comparaisons. Ils sont tous binaires

- Inférieur juste <
- Inférieur ou égale <=
- supérieur juste >
- Supérieur ou égale >=
- Égalité ==
- Différence !=

```
Euro > Fb ; 15 >= 18 ; (a + b) <= (c * d)
```

### 8.3. Opérateurs logiques

Ils sont utilisés avec les opérateurs relationnels. Ils permettent de combiner les résultats de plusieurs tests conditionnels en un seul.

- Et logique &&
- Ou logique ||
- Non logique !

```
A && b ; a || b ; (a>b) && (c<b)
```

### 8.4. Opérateurs unaires spéciaux

Raccourcisse l'écriture

```
a = a + 1      a++  
a = a - 1      a--
```

Quand le signe ++ est placé avant l'opérande sur lequel il agit, l'opérande est incrémenté immédiatement avant d'intervenir dans l'opération qui suivrait.

*Exemples*

```
a = 5 ; c = a++ +2 ; donne c = 7 pas 8
```

```
a = 10 ; c = ++b+ 2 ; donne c = 13 pas 12
```

### 8.5. Opérateurs d'affectation

```
a = 2 ; c = 10  
d = (a + (b+c))      b = a + b + c  
d = a = b = c = 4    a, b, c et d prennent la valeur 4  
a += 3 ;             équivaut à a = a + 3
```

### 8.6. Opérateur d'affectation conditionnelle (oufti ??)

```
solde = 125 ; max = 120 ;  
remise = (solde > max) ? 15 : 10 ;      résultat = 15
```

*Explications :* Si la valeur de solde (125) est supérieur à max(120) alors on prend la première valeur après le ? Soit 15 sinon on prend 10. (il s'agit d'un raccourci de la fonction 'if ... Else ...')

### 8.7. Concaténation

Permet de former une nouvelle chaîne avec deux chaînes

```
a = "Bonne";  
b = "frites";  
c = a + " " + b;      Résultat c = "Bonne frites"
```

## 9. If... else ...

On demande de choisir entre deux blocs d'instructions et de n'en exécuter qu'un.

```
if (age > 35)  
    Retour = "presque vieux";  
else  
    Retour = "Gamin";
```

Ici on n'exécute qu'une seule instruction, si l'on doit en réaliser une série elles seront encadrées par { .... }

## 10. Switch .. case

permet avec une variable unique de séparer les instructions à réalisé suivant la valeur de cette variable

```
var date = new Date();  
var jour=date.getDay();  
switch(jour) {  
    case 0:  
        document.write("Dimanche !");  
        break;  
    case 6:  
        document.write("Samedi !");  
        break;  
    default:  
        document.write("Pleine semaine : Au boulot");  
        break;  
}
```

## 11. While...

L'instruction permet de répéter une instruction ou un bloc jusqu'à ce que la condition du while est remplie

```

var fin = 10 ;
var indice = 1;
while (indice <= fin) {
    document.write(indice) ;
    indice++
}

```

## 12. For ...

Même chose que while mais avec un compteur

```
for (initialisation compteur ; condition ; incrémentation compteur)
```

```

var jours = new Array("Dimanche","Lundi","Mardi","Mercredi","Jeudi","Vendredi","Samedi");
for (compteur=0; compteur<8; compteur++)
    document.write(jours[compteur]);

```

## 13.Exercices :

**1.**

Après une recherche sur l'objet date, réalisé un script qui affiche le jour comme ceci :

**Nous sommes le Jeudi 4 septembre 2014**

**2.**

Réalisé un tableau avec une colonne avec vos prénom, et les autres tous les jours du mois de septembre (30 jours)

Nom	Septembre										
	L - 1	M - 2	M - 3	J - 4	V - 5	S - 6	D - 7	L - 8	M - 9	M - 10	...
Jérémy											
Laetitia											
Brice											
...											

# 3

## Js dans le navigateur



### 1. Quand et pourquoi utiliser le JavaScript

Le JavaScript est un langage très dynamique, on peut faire presque tout avec. Vous pouvez faire de bonne chose comme de très mauvaise chose, il est très facile d'être perdu dans votre code JavaScript. Juste parce que le JavaScript peut modifier l'HTML et contrôler la CSS.

### 2. Améliorer l'expérience de l'utilisateur

Ce que JavaScript fait (ou comment vous l'utilisez, du moins) est d'améliorer l'expérience d'utilisateur par la couche de comportement. Nous ajoutons du peps à ce qui existe déjà, ajoutant des gestions d'événement au site ou à l'application, la rendre plus responsive avec l'aide de l'Ajax, créer un stockage de donnée en sur la machine cliente pour une plus grande performance, etc.

### 3. Utiliser le JavaScript responsive

Utiliser le JavaScript pour un site responsive implique que vous connaissez la différence entre l'utilisation du JavaScript ou de la CSS.

Le JavaScript doit être entièrement chargé, est lent et à parfois des problèmes de sécurité que vous devez prendre en compte. Quand vous préparez la couche de gestion d'événement de votre projet, vous devez vous poser la question de savoir si vous devez absolument utiliser du JavaScript pour l'opération que vous tentez de faire. Dans beaucoup de cas une utilisation combinée de JavaScript et CSS permet de créer le même effet simplement en changeant la class d'un élément.

Visible / caché est une action courante sur le web, et est souvent approché avec du JavaScript. Mais une simple interaction comme celle-ci peut être réalisée plus efficacement avec une combinaison de CSS et JavaScript. Dans le listing suivant vous pouvez voir une page classique HTML que nous allons utiliser pour expliquer le concept de masque du texte par l'action d'un click sur un bouton. Vous pouvez remarquer qu'il n'y a pas de JavaScript dans la page.

```
<!doctype html>
<html lang="fr">
<head>
    <title>Click pour cacher</title>
    <meta charset="utf-8">
    <link rel="stylesheet" href="css/hide.css">
</head>
<body>
    <button type="button" id="hide">cache le texte</button>
    <div id="target">
        <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Morbi dolor metus, sagittis et aliquam et, ornare et libero. Etiam eu nisi felis, ac posuere metus. Vivamus molestie bibendum imperdiet. Etiam et faucibus metus.</p>
    </div><!--/#target-->
    <script src="js/hide.js"></script>
</body>
</html>
<!-- Fin de la page ex301.html -->
```

Maintenant voici la page CSS

```
.hide {
    display: none;
}
```

Ajouter une classe `hide` dans la feuille CSS nous permet de l'utiliser quand on le veut et sur n'importe quel objet de la page. Cet exemple est très primitif, mais vous pouvez ajouter n'importe quel class dans la page CSS que vous auriez besoin de faire référence avec le JavaScript. Cela permet également de ne pas mélanger la CSS et le JavaScript.

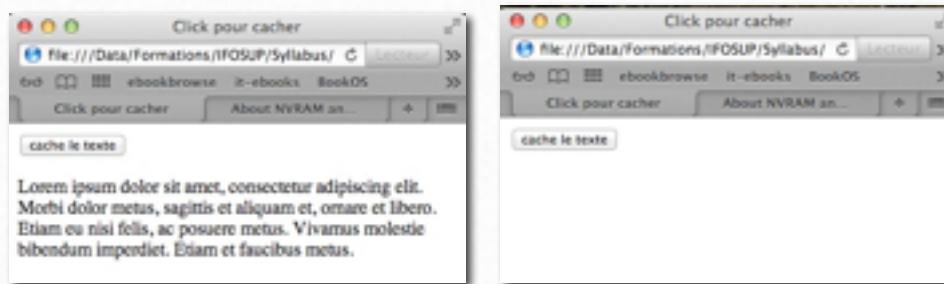
Voici le code JavaScript pour votre page

```
/* sauvegarde les 2 noeuds dans des variables */
var button = document.getElementById("hide"),
    target = document.getElementById("target");

/* La fonction qui ajoute la class hide a l'élément */
function hide() {
    target.setAttribute("class", "hide");
}

/* ajoute-la class dans le cas du click sur le bouton */
button.addEventListener("click", hide, false);

// Fin du fichier hide.js
```



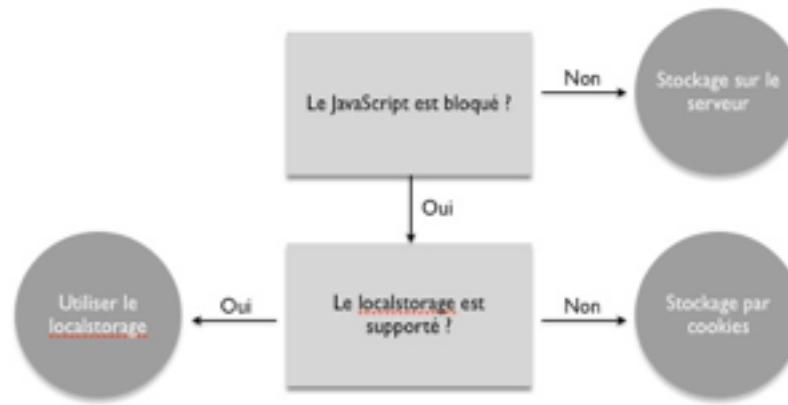
Tout ce que l'on fait ici est de rajouter une classe à notre div target et on laisse la CSS faire son travail. Si vous tester cette page, en cliquant sur le bouton vous ferrez disparaître le texte. Cette méthode en utilisant la CSS vous permettra de rendre le site plus performant, car le traitement CSS est plus rapide que le JavaScript et la CSS sera déjà dans le cache de votre ordinateur avant de l'utiliser.

#### 4. Générer des Fallbacks

Un fallback est du code que vous créez si votre code principal ne peut fonctionner. Le premier type de fallback est par exemple l'utilisation d'Ajax ou de mise en forme de tableau si l'utilisateur a bloqué le JavaScript, le site fonctionne'il ?

Le second niveau de création de fallback est lié à l'utilisation de méthode JavaScript qui ne serait pas compatible avec les différents navigateurs, mais que vous pourriez faire autrement.

Prenez par exemple que nous voulons utiliser une nouvelle technologie de l'HTML5 , le stockage dans une base de données en local (sur la machine du client), le JavaScript peu déjà stoker des infos depuis toujours avec les cookies, dans notre exemple nous aurons 3 niveaux de fallbacks, la figure suivante vous explique quelle méthode de stockage sera pris et dans quel cas :



Exemple de fallback

```

/* You know JavaScript is enabled at this point because none of this would apply
otherwise */

/* Test si le localStorage est supporté */

if(typeof window.localStorage !== "undefined"){

    // Utilisons localStorage
} else {
    // utilisons les cookies
}

```

## 5. Des outils pour utiliser le JavaScript

Le JavaScript n'est pas comme un langage côté serveur, quand vous faites une erreur de syntaxe; celle-ci sera très visible, car un message sera affiché en plein milieu de votre site. De plus le débogage des erreurs est parfois très difficile.

Par chance , une série d'outils sont disponibles pour un débogage de notre JavaScript.

### 5.1. Les outils intégrés dans le langage

La console JavaScript est l'outil qui peut sauver votre vie de designer ou développeur. La console est intégrée dans le navigateur, mais je considère celle-ci comme intégré dans le langage.

La console affiche automatiquement une série d'information, comme, le suivi des appels Ajax et code d'erreur avec le numéro de la ligne (facile ...). Vous pouvez voir l'URL appelée par la requête Ajax et les données transmises. Cela peut être très utile pour votre débogage.

La console peut être aussi répondre a des commandes du JavaScript, Il est parfois intéressant d'afficher des données ou points de passage ou d'erreur dans votre code. Voici un exemple :

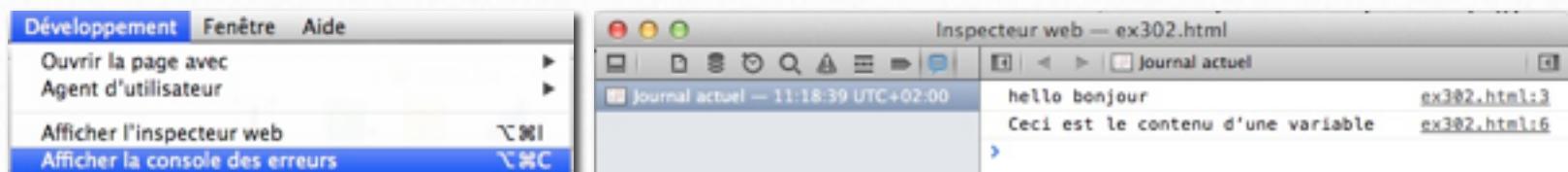
```

<script>
    /* Envoi d'un message simple à la console */
    console.log("hello bonjour");
    /* Envoi du contenu d'une variable à la console */
    var msg = "Ceci est le contenu d'une variable";
    console.log(msg);
</script>
<!-- fin du fichier ex302.html -->

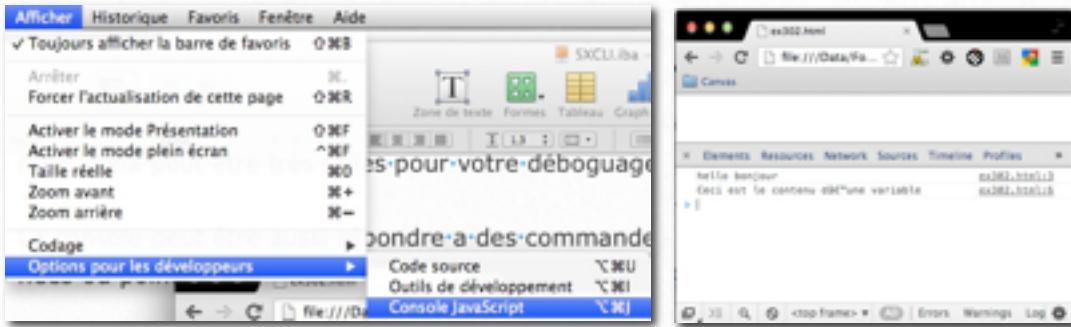
```

Chaque navigateur a une console, cela est plus une difficulté de la trouver dans les menus. Si vous avez un menu Developer, vous la trouverez obligatoirement.

*Exemple pour Safari version 6.0.5*



*Pour Chrome version 28*



## 5.2. Les outils dans le navigateur

Par outils dans le navigateur je pense aux applications web créer pour vous aidées à écrire en JavaScript. Ces outils peuvent par exemple supprimer tous les espaces inutiles de votre code pour le rendre plus léger, comme YUI Compressor (<http://refresh-sf.com/yui/>); ce compresseur va augmenter la performance de votre site en réduisant la taille de votre fichier JS et même CSS. Attention de toujours garder une version non compressée de votre code en cas de modification, car vous pouvez retrouver tout votre JS en une seul ligne !

Des outils de compression comme YUI est super, mais il excite un autre outil en ligne appelé JSLint (<http://www.jslint.com>) qui vas analyser votre code et afficher les erreurs de syntaxe et de mise en forme, JSLint va vous aidée à écrire un code plus propre et plus fonctionnel, les erreurs ne sont pas toutes des erreurs qui vont planté votre navigateur.

# 4

## Terminologie JavaScript



Dans ce chapitre nous allons apprendre la terminologie JavaScript que vous aurez besoin pour la suite du cursus et avancée dans l'apprentissage de ce langage.

Nous verrons certain nouveau mot *cochon* et nous analyserons des exemples de code, cela est aussi important de savoir rentrer dans du code que vous n'avez pas réalisé, de savoir identifiez ce que vous chercher.

## 1. Les basics

Nous avons déjà revu une partie des bases dans le chapitre 2, voici une autre partie importante à connaître avant la suite.

Le JavaScript est très flexible (certains le trouvent pas structuré) certains termes peuvent dire autre chose dans certaine circonstance (comme fonction et méthode).

### 1.1. Document Object Model (DOM)

À son niveau le plus bas, le DOM est une découpe du document HTML que vous accédez. Il fonctionne de la même manière qu'un livre ou un article, tout objet avec une structure. Il y a un niveau au-dessus et des éléments liés en dessous et des éléments groupés comme des pages dans un chapitre et des chapitres dans un livre.

Les éléments sont appelés nœud et tous les nœuds dans le DOM ont une relation avec les nœuds autour d'eux, comme une vraie famille

- Parent
- Child (enfant)
- Sibling (frère «même parent»)

#### I. Parent

Le nœud parent est tous les éléments comportant des nœuds, vous ne pas comprendre? Par exemple l'élément <body> est parent des tous les éléments qu'il comprend (balise div, élément de tableau, etc. ), tous ces éléments seront les enfants (child). Le <body> peut aussi contenir d'autres parents , mais ils seront toujours ses enfants (child).

```
<!-- <ul> est le l'élément parent de tous la liste des <li> -->
<ul>
    <li>Cléo</li>
    <li>Maxime</li>
    <li>Diego</li>
    <li>Michael</li>
    <li>Jérémie</li>
    <li>Franck</li>
</ul>
```

#### II. Child (enfant)

Dans le DOM, les enfants se positionnent à l'intérieur d'un nœud parent

```
<!-- tous les éléments <li> du nœud parent <ul> -->
<ul>
    <li>Cléo</li>
    <li>Maxime</li>
    <li>Diego</li>
    <li>Michael</li>
    <li>Jérémie</li>
    <li>Franck</li>
</ul>
```

### III. Sibling

Sibling, comme dans la vie normale, est au même niveau. Si un parent a plusieurs enfants, ils sont référencés comme sibling. Dans le code suivant tout les `<li>` sont frère (sibling), car tous dans un même nœud parent `<ul>`

```
<!-- tous les <li> sont sibling des autres <li>, car ils sont au même niveau -->
<ul>
    <li>Cléo</li>
    <li>Maxime</li>
    <li>Diego</li>
    <li>Michael</li>
    <li>Jérémy</li>
    <li>Franck</li>
</ul>
```

## 2. Javascript Object Notation (JSON)

Nous avons précédemment vu le stockage des données dans un tableau, il existe un autre format, le JSON, il est souvent utilisé pour des envois de données vers des services extérieurs au JavaScript. Il a été créé à la base pour être une alternative au XML.

Il a vite pris de la place par rapport au XML, car , plus léger, et peux facilement être utilisé avec l'Ajax entre différents domaines, de plus il est natif dans le JavaScript, indépendants par rapport aux plateformes et est utiliser par toutes les technologies cotées serveur.

Voici le même exemple en JSON

```
{
    "family" : [
        "Cléo",
        "Maxime",
        "Diego",
        "Michael",
        "Jérémy",
        "Franck"
    ]
}
```

## 3. Objet

Tout est objet en JavaScript et il est facile de créer de nouveau, chaque objet aura des propriétés et peut être aussi des méthodes, prenons l'exemple de ma grand-mère (un objet aussi «figurer!»), ces propriétés son son nom, son prénom, son surnom, etc., elle pourrait avoir des méthodes style : marche, parle, etc.

Voici une représentation de ma grand-mère en JavaScript :

```
/* Stocke des informations sur ma grand-mère dans un objet */
var grandMere = {
    "nom": "Renard",
    "prenom": "Nicole",
    "surnom": "mémé"
};
```

Tout peut être stocké dans un objet, comme le nom, ou encore une gestion d'événement ou des fonctions (méthode) complet comme `receptionEmail()`. C'est la première étape de créer de ma programmation-objet en JavaScript.

## 4. Fonctions

Une fonction est une block de code qui est exécuté suite un événement ou une demande, l'évènement peut être le chargement d'une page ou un évènement de l'utilisateur comme un click. Vous pouvez également envoyer des données (paramètres) à votre fonction, ce qui permet d'étendre les possibilités de la fonction. De même une fonction peut renvoyer une valeur qui peut être utilisée dans votre code.

```
<script>
function recuperLesNoms() {
    /* Stocke les nom dans un tableau */
    var developer = [
        "Cléo",
        "Maxime",
        "Diego",
        "Michael",
        "Jérémie",
        "Franck"
    ];
    var nbDeveloper = developer.length;
    var i;
    /* Test si la liste n'est pas vide */
    if(nbDeveloper > 0) {
        for(i = 0; i < nbDeveloper; i++) {
            var personne = developer[i];
            /* Si la personne est Cléo, on affiche un message */
            if(personne === "Cléo") {
                document.write("Bonjour " +personne+"<br>");
            } else {
                document.write(personne+"<br>");
            }
        }
    }
}
/* Appel de la fonction recuperLesNoms();
recuperLesNoms();
</script>
<!-- fin du fichier ex401.html -->
```

Bonjour Cléo  
Maxime  
Diego  
Michael  
Jérémie  
Franck

## 5. Fonction anonyme

Une fonction anonyme est déclarée quand elle est utilisée, elles n'ont pas de nom assigné. A la place décrire une fonction détaillée comme celle vue le point précédent, vous pouvez utiliser une fonction anonyme qui sera directement exécutée quand la page est chargée à la place d'avoir la référence d'une fonction quelque part dans le document.

Les fonctions anonymes sont plus rapides d'une fonction définie, car il n'y a rien à référencer; elle démarre au besoin. Ces fonctions ne s'utilisent qu'une seule fois. Voici la conversion de la fonction recuperLesNoms en fonction anonyme :

```
<script>
( function() {
    /* Stocke les nom dans un tableau */
    var developer = [
        "Cléo",
        "Maxime",
        "Diego",
        "Michael",
        "Jérémie",
        "Franck"
    ];
    var nbDeveloper = developer.length;
    var i;
    /* Test si la liste n'est pas vide */
    if(nbDeveloper > 0) {
        for(i = 0; i < nbDeveloper; i++) {
            var personne = developer[i];
            /* Si la personne est Cléo, on affiche un message */
            if(personne === "Cléo") {
                document.write("Bonjour " +personne+"<br>");
            } else {
                document.write(personne+"<br>");
            }
        }
    }
})()
</script>
<!-- fin du fichier ex402.html -->
```

Le résultat est le même



Bonjour Cléo  
Maxime  
Diego  
Michael  
Jérémie  
Franck

## 6. Fonction callback

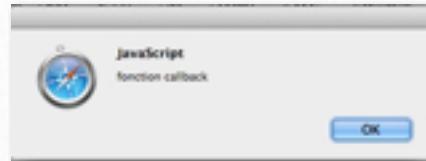
Quand une fonction est passée en paramètre à une autre fonction, on l'appelle une fonction callback. La fonction est quelque chose que vous définissez, ou elle peut aussi être une fonction native du JavaScript.

Les fonctions sont très utiles pour séparer votre logique et garder votre base de code la plus réutilisable possible. Les fonctions anonymes peuvent également être des fonctions callback. Plus difficile à expliquer, mais regardons le code suivant :

```

<script>
window.addEventListener("load", function () {
    alert("fonction callback ");
}
, false );
</script>
<!-- fin du fichier ex403.html

```



Quelques explications : La fonction anonyme callback est attachée à la méthode `addEventListener` native du JavaScript et se lance à l'événement `load`.

## 7. Méthode

Premièrement une méthode est dans tous les cas une fonction. La différence entre l'étiquette que l'on donne à une «fonction» est purement organisationnelle. Plus tôt dans le syllabus, j'ai parlé de l'objet en JavaScript, et j'ai mentionné que l'on pouvait y sauver plein de choses dedans, également des fonctions. Quand cela arrive et qu'une fonction est enregistrée dans un objet elle s'appelle méthode (capitchi!).

Comme une méthode native du JavaScript `alert()`, vous pouvez également générer des méthodes, et cela dépendra totalement au choix que vous avez fait pour organiser votre code (parfois le bor...). Dans un chapitre futur, je vais essayer de rentrer plus dans le détail d'une bonne organisation de son code. Pour le moment il est important que lorsque vous avez une fonction native ou d'une librairie externe, elle s'appelle méthode. Voici un exemple de création d'une méthode :

```

<script>
/* définit notre objet qui contient des méthodes */
var getInformation = {
    /* Première méthode (fonction dans un objet) nommée "nom" */
    "nom": function () {
        alert("Charge les noms");
    },
    /* second méthode est appellé "rechCleo" */
    "rechCleo": function () {
        alert("Recherche de Cléo")
    }
};

/* charge les noms au chargement «load» */
window.addEventListener("load", getInformation.nom, false);
/* recherche Cléo au click */
document.addEventListener("click", getInformation.rechCleo, false);
</script>
<!-- fin du fichier ex404.html

```

Si vous testez ce script, vous aurez la première méthode `nom` qui démarre au chargement de la page et la seconde méthode `rechCleo` qui sera appelée uniquement si vous cliquez dans le document.

L'appel d'une méthode sera toujours précédé de l'objet dans lequel elle a été construite sauf pour les méthodes natives au JavaScript.

## 8. évènements (events)

La gestion des évènements est le seul moyen d'avoir un retour de l'utilisateur. Si vous voulez exécuter n'importe quel code JavaScript dans une page, il doit avoir un déclencheur quelconque. Chargement de la page, clique sur un lien et envoyer un formulaire sont tous considéré comme des events, et, vous pouvez y attaché une fonction. Il existe une grande liste d'évènement que vous pouvez lié a une fonction en JavaScript, en voici une série ;

• click	• dblclick	• mousedown	• mouseout
• mouseover	• mouseup	• keydown	• keypress
• keyup	• blur	• focus	• submit
• load	• touchstart*	• touchmove*	• touchcancel*
• orientationchange*	• gesturestart*	• gestureend*	• gesturechange*

\* Nouveaux évènements liés aux écrans tactiles (smartphone, tablette, écran d'ordinateur)

## 9. Ajax

L'Ajax est un concept de rafraîchissement d'une partie du document HTML sans recharger l'entièreté de la page. Très peu utiliser, mais je vais vous montrer les méthodes propres d'utilisation de cette technologie dans le cadre de ce cursus.

# 5

## DOM



Dans ce chapitre nous allons mettre en oeuvre les concepts vus précédemment, nous allons voir comment nous allons de différente façon manipuler les noeuds (les éléments) d'une page, de monter ou descendre la structure du document. Ceci est la base de toutes interactions avec le JavaScript.

## 1. Qu'est-ce que le DOM ?

Quand vous entendez les gens parler de JavaScript, ils mentionnent souvent le «DOM». Ceci est un aspect le plus commun et des plus importants du JavaScript. Vous allez l'utilisez tout le temps quand vous modifiez une page ou que vous insérez des données ou modifier l'interface. Le DOM est le meilleur copain du développeur JavaScript quand il en a une maîtrise parfaite.

Le DOM n'est pas du HTML, ni du JavaScript, mais il en est très interconnecté. Ce sont les combinaisons des éléments (object) que vous utilisez, l'ordre que vous choisissez pour les afficher (model), et les possibilités d'ajout, de modification ou suppression d'une partie de la structure complète (document) tandis que vous naviguer avec les différentes méthodes que le JavaScript fournit.

### 1.1. L'arborescence du DOM

Chaque document a une sorte de structure derrière. Le DOM commence avec le navigateur, quand vous visitez un site, le navigateur affiche l'HTML dans un document, passe à travers les éléments et les affiche à l'utilisateur dans le format correct. Ce processus crée le modèle initial du document, et les stockent comme cela in ne doit pas refaire tout le travail pour chaque éléments. Considérons la page HTML suivante :

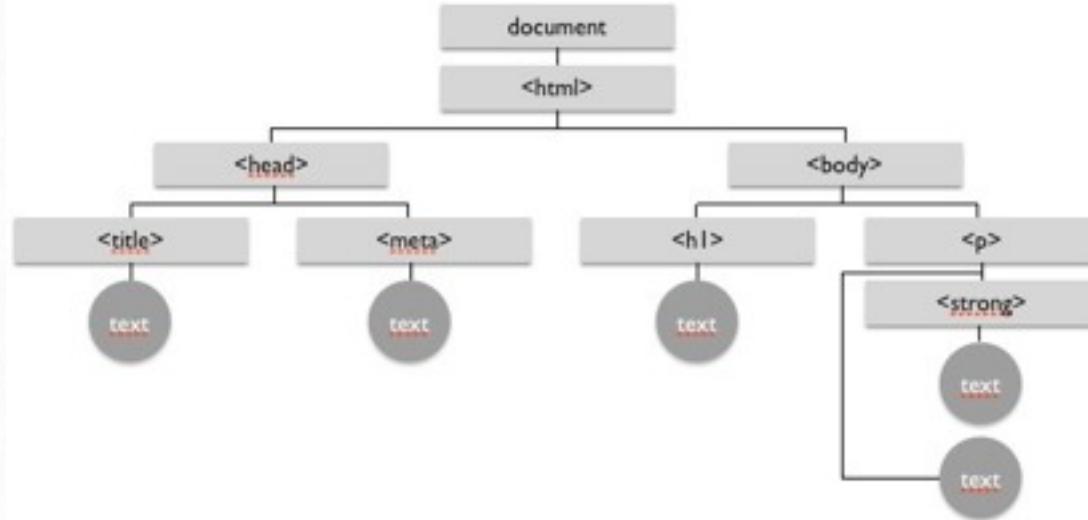
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset= "utf-8 ">
    <title>exemple basic du DOM</title>
  </head>
  <body>
    <h1>Bonsoir !</h1>
    <p>Ceci n'est <strong>qu'un document basic en HTML</strong>, Il sert actuellement d'exemple détaillé du Document Object Model.</p>
  </body>
</html>
```

A premier abord, cela ressemble a une page HTML extrêmement basic, et cela est vrai, mais ces aussi une très bonne représentation des tous les éléments que nous retrouvons dans le DOM.

Le DOM est fait d'éléments appelés nœud (on a déjà oublié,) qui peuvent avoir des parents, des enfants (children) et de frère (sibling) et cela détermine sa position dans l'arborescence. Les enfants sont en dessous des parents, et les sibling sont aux mêmes niveaux que les autres sibling (frères ou sœur), voici une représentation graphique de cet arbre.

Sur la figure à la page suivante vous pouvez voir la relation parent-enfant en partant du dessus et en descendant dans l'arborescence

comme vous l'avez sûrement (j'espère) observé , il y a différent type de nœuds dans le DOM. Les nœuds représentant des éléments HTML s'appellent *nœud d'élément*, les nœuds représentant du texte s'appellent *nœuds de texte* (facile !) et les derniers sont les nœuds d'attribut qui, vous l'avez deviné, représente des attributs. Tous ces nœuds sont des enfants (children) du parent qui est le *nœud document* qui représente le haut de la figure.



Il est important de connaître la différence de type de nœuds, car nous les accédons différemment suivant le parcours que nous réalisons dans le DOM en JavaScript.

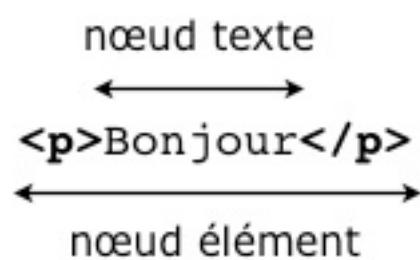
### 1.2. Nœud élément

Les nœuds élément occupent la majorité de votre document et sont la base de votre déplacement (nous allons comprendre plus loin). Ces nœuds vont définir la structure et contiennent la plus grande partie des données sur laquelle vous voulez interagir.

### 1.3. Nœud texte

Les nœuds texte sont similaires aux nœuds élément par leurs positions dans le DOM et les mêmes méthodes JavaScript qui permettent d'y accéder. Si il y a une différence entre ces deux types de nœuds sinon on aurait pu apprendre d'autre technologie.

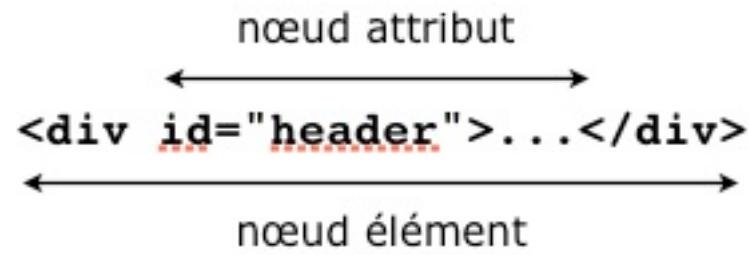
La grosse différence est la façon dont ils sont représentés, les nœuds élément sont entourés de chevrons «<...>», les nœuds texte sont entre deux nœuds élément.



Une importante différence entre les deux, le nœud texte ne peut pas avoir d'enfant (children), vous pouvez le voir dans la partie droite de la figure avec les éléments <strong>.

### 1.4. Nœud attribut

Les nœuds attribut, comme le nœud document, passe souvent inaperçus quand vous créez votre DOM, car il apparaît comme faisant partie d'un élément, pourtant ils sont un autre type de nœud, et sont d'une importance primordiale.



Comme les nœuds texte, les nœuds attributs ne peuvent avoir d'enfants (pauvre de nous) mais ne peuvent être un enfant (oups cela ce complique!). Exact , le nœud attribut n'est pas considéré comme un enfant de l'élément dans lequel il se trouve. Il se retrouve dans la structure du DOM dans un élément parce qu'ils font toujours attaché à un nœud élément, mais ne sont pas des enfants, ils sont traités et accessible par des méthodes différentes.

# 6

## Travailler avec les nœuds



Maintenant que vous avez bien compris (sinon retour chapitre 5) ce qu'est le DOM et la relation parent-children, nous pouvons voir comment y avoir accès. Tous les nœuds vus précédemment sont accessibles, certains facilement, d'autres plus difficilement. Je vous rassure, tout ce qui se trouve dans le DOM est récupérable et modifiable en JavaScript.

## 1. Par ID

Le standard HTML stipule que la valeur de l'Id doit être unique dans le document. En d'autres mots, il ne peut y avoir qu'une fois un Id dans une page. Grâce à cela, utiliser un élément avec un id sera facilement accessible et donc préférable.

```
<div id="header">
    <h1><a href="/" id="base">titre de votre site</a></h1>
</div>
```

Voici un code HTML simple avec deux Id que nous allons accéder :

```
document.getElementById("header");
document.getElementById("base");
```

Comme vu précédemment, vous devez d'abord accéder au document avant d'atteindre le nœud que vous désirez. Quand vous aurez accès au nœud rechercher , vous pouvez le stocker, le modifier ou même récupérer des informations complémentaires sur l'élément avec une pléthore de méthode construite dans le JavaScript que nous allons survoler dans ce syllabus.

Le support de la méthode `getElementById()` est reconnu dans tous les navigateurs, c'est une des méthodes des plus ancienne et des plus usitée.

L'utilisation d'Id dans votre HTML est très pratique pour le JavaScript, mais également pour la CSS.

## 2. Par balise (tag name)

Parce que l'Id est unique dans chaque document, `getElementById()` ne permet que de sélectionner un seul élément. Parfois vous avez besoin d'en sélectionner plus d'un, ou même un groupe. Dans ce cas nous allons utiliser la méthode `getElementsByName()`. Faites attention à ceci, car vous pouvez gagner du temps de débogage.

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>getElementsByName Exemple</title>
</head>
<body>
    <h1>Utilisation de getElementsByName</h1>
    <p>Content paragraph</p>
    <p>Content paragraph</p>
    <p>Content paragraph</p>
</body>
</html>

```

Utilisation de la méthode :

```
document.getElementsByTagName("p");
```

Cette méthode va nous renvoyer un objet DOM appelé `NodeList` (Liste de nœud). Une `nodeList` est une collection de nœuds qui garde l'ordre de la source. Dans d'autres mots, une liste d'éléments, et ces éléments apparaissent dans le même ordre que dans la page. Cet ordre est important.

Voici une des méthodes utilisables avec une NodeList: la méthode `length` qui renvoie ... le nombre de nœuds dans la NodeList

```
document.getElementsByTagName("p").length;  
// devrait valoir 3  
  
// peut servir dans un test  
if(document.getElementsByTagName("p").length > 0) {  
  
}
```

Pour le moment nous avons ciblé un groupe d'élément et testé ça taille, après cela, nous allons sélectionné un des éléments pour faire quelque chose. Il y a deux possibilités :

- Utilisé la méthode sur un élément
- Utilisé une gestion des tableaux ( le NodeList est un tableau)

Ces deux méthodes nous fourniront la même valeur, cela est donc un choix personnel qui vous fera choisir, voici les deux exemples d'utilisation :

```
// le premier paragraphe en utilisant une méthode sur l'élément  
document.getElementsByTagName("p").item(0);  
  
// le premier paragraphe par le tableau  
document.getElementsByTagName("p")[0];  
  
// le second paragraphe en utilisant une méthode sur l'élément  
document.getElementsByTagName("p").item(1);  
  
// le second paragraphe par le tableau  
document.getElementsByTagName("p")[1];
```

### 3. Par class

Jusqu'à maintenant nous avons un mode de sélection des nœuds comme le CSS. En CSS vous pouvez cibler par l'Id ou Balise, vous pouvez aussi ciblé par class. Le JavaScript le permet aussi.

```
<!doctype html>  
<html>  
<head>  
    <meta charset="utf-8 ">  
    <title>getElementsByTagname exemple</title>  
</head>  
<body>  
    <h1>Utilisation de getElementsByTagName</h1>  
    <p class="dropcap huge">Content paragraph</p>  
    <p id="debut" class="congres">Contenu <span class="belge">paragraphe</span></p>  
    <p class="congres">contenu paragraphe</p>  
</body>  
</html>
```

Voici les méthodes de sélection :

```

// sélectionne tous les éléments avec la class = congres
document.getElementsByClassName("congres");

// sélectionne tous les éléments avec les deux class congres et Belge)
document.getElementsByClassName("congres belges");

// sélectionne tous les éléments avec la class belge et se trouvant dans
// l'élément, dont l'Id = debut
document.getElementById("debut").getElementsByClassName("belge");

```

Nous nous retrouvons avec plus de flexibilité, car nous pouvons combiner pour mieux cibler .

Juste un petit truc ... Comme cela a été implémenté récemment voici le tableau des navigateurs compatibles :

Navigateur	Version min
IE	9.0
Firefox	3.0
Chrome	4.0
Opera	9.5
Safari	3.1

#### 4. Utiliser les sélecteurs CSS en JavaScript

Avec l'arrivée de l'HTML5 et une nouvelle version de l'API du JavaScript, deux nouvelles méthodes sont arrivées : `querySelector()` et `querySelectorAll()`.

```

<body>
  <div id="header >
    <h1>Utilisation de querySelector</h1>
  </div>
  <p class="congres belge">Paragraphe 1</p>
  <p class="congres">Content <span class="belge">Paragraphe 2</span></p>
  <p class="congres">Paragraphe 3</p>
</body>

```

Voici un code HTML simple avec une combinaison de class.

Et l'utilisation de ces nouvelles méthodes :

```

// récupère le nœud avec l'Id = header
document.querySelector("#header");

// récupère le premier nœud avec la class congres
document.querySelector(".congres");

// récupère tous les nœuds avec la class congres / soit un NodeList
document.querySelectorAll(".congres");

```

```
// récupère tous les nœud avec la class congres OU Belge
document.querySelectorAll(".congres, .belge");

// récupère tous les paragraphes p possédant une class
document.querySelectorAll("p[class]");
```

`querySelector()` est une méthode extrêmement flexible. Il peut prendre de multiple sélecteur Id, class, attribut et même les pseudo class du CSS. Le seul inconvénient est son manque de message d'erreur en cas de problème. Voici le tableau du support de cette méthode dans les navigateurs :

Navigateur	Version min
IE	8.0
Firefox	3.5
Chrome	1
Opera	10
Safari	3.2

## 5. Travailler avec les nœuds attributs

Maintenant que vous êtes des champions dans le criblage des nœuds , nous pouvons commencer a travailler avec ces nœuds et les nœuds attributs. Pour rappel un attribut est dans un nœud élément, mais n'en est pas un enfant de celui-ci, de là nous devons utiliser des nouvelles méthodes spéciales pour récupérer, modifier, ou supprimer cette information. Les voici :

- `getAttribute()`
- `setAttribute()`
- `removeAttribute()`
- `hasAttribute()`

Dans toutes ces méthodes nous ciblons l'attribut et en récupérons sa valeur comme dans l'exemple d'une `class="visible"`, vous devriez ciblé la "class" et récupérer la valeur "visible".

Voici le code HTML sur lequel nous allons travailler :

```

```

### 5.1. Récupérer un attribut

Avant de récupérer notre attribut, nous devons nous plonger dans le DOM vers le nœud (avec les méthodes vues précédemment) et seulement à ce moment-là nous pourrons avoir accès à ces attributs.

```
// vérifie si le nœud avec l'Id monImage bien une class attachée
if(document.getElementById("monImage").hasAttribute("class")) {
    // récupère la valeur de la class si elle existe
    document.getElementById("monImage").getAttribute("class");
}
```

## 5.2. Changer la valeur d'un attribut

L'établissement d'une valeur d'un attribut est utile dans beaucoup de cas, pour sauvegarder temporairement des données, changé une class ou remplacer un attribut par quelque chose de nouveau. Ces méthodes ont un test intégré, grâce à cela nous ne devons pas réaliser de test tant que nous voulons récupérer sa valeur.

```
// remplace valeur de la class en cours par la valeur "hidden"
document.getElementById("monImage").setAttribute("class", "hidden");

// ajoute un attribut à l'image
document.getElementById("monImage").setAttribute("title", "Nous avons retiré cette image");
```

La méthode reçoit 2 paramètres, le premier est le nom de l'attribut et le second est sa valeur. Dans le premier exemple comme la class existe, il en change sa valeur. Dans le second cas, l'attribut étant inexistant sur le nœud, la méthode la rajoute (le fameux test intégré).

Voici la page css :

```
* Ceci est l'état par défaut de l'image*
.visible {
    position: static;
}
/* ceci est appelé lors du changement d'état de la class */
.hidden {
    position: absolute;
    top: -9999px;
    left: -9999px;
}
```

**Important** : Il est important de ne pas utiliser `display: none` quand vous travaillez de cette façon en CSS, car les lecteurs d'écran ne peuvent voir le contenu avec la propriété réglée à `display: none`.

## 5.3. Supprimer un attribut

Supprimer est aussi simple que sélectionné un attribut. Il suffit de se positionner sur le nœud élément et d'utiliser la méthode `removeAttribute()` pour l'envoyer aux oubliettes. Il n'y a pas d'erreur JavaScript si vous voulez supprimer un attribut qui n'existe pas, mais la meilleure technique de programmation vous fera tester avec `hasAttribute()` avant de supprimer.

```
/ en premier on teste si l'attribut existe sur l'élément
if(document.getElementById("monImage").hasAttribute("class")) {
    // si l'attribut class existe on peut la supprimer
    document.getElementById("monImage").removeAttribute("class");
}
```

Ces méthodes sont compatibles avec tous les navigateurs.

## 6. Les nœuds texte

Maintenant que nous avons vu les nœuds élément et les attributs, il nous reste le dernier type de nœud : les nœuds texte. Les manipulations les plus courantes dans ce type de nœud est la modification de son contenu, il n'y a pas de navigation dans ce type de nœud, car ils ne peuvent avoir d'enfant.

Changer du contenu est très courant en JavaScript, pour changer le nom d'un bouton, charger la valeur avec

l'Ajax, vous allez de vous même insérer du code HTML ou du texte dans le DOM. Vous pouvez réaliser cela avec la méthode `innerHTML()`.

```
<body>
  <div id="zoneCible">
    <p>Ceci est notre texte.</p>
  </div>
</body>
```

et l'utilisation de la méthode :

```
// ciblé le contenu du texte
document.getElementById("zoneCible").innerHTML;

// changer le contenu du texte.
document.getElementById("zoneCible").innerHTML = "<p>Bonjour les dévellopper</p>";
```

# 7

## Se déplacer dans le DOM



En déplaçant à l'intérieur du DOM, nous commençons à bien utiliser l'arborescence du DOM et bénéficions de la connaissance des zones de notre page HTML, nous connaissons les relations entre tous ces éléments. Maintenant nous savons cibler n'importe quel nœud directement, mais l'écriture d'un HTML sémantique n'est pas encore pour tout le monde. Nous allons voir une série de méthodes qui nous permet de naviguer d'une autre façon dans notre DOM, voici ces méthodes :

- parentNode

- previousSibling
- nextSibling
- firstChild
- lastChild

Quand vous avez ciblé un *enfant* d'un nœud élément, vous pouvez récupérer son *parent* et ils ont tout un *parent*. Regarder le code suivant, nous allons récupérer le parent d'un élément sélectionné et lui ajouté une class active.

```
<ul id="nav">
  <li><a href="/" id="home">Accueil</a></li>
  <li><a href="/info" id="info">Info sur nous</a></li>
  <li><a href="/contact" id="contact">Contactez-nous</a></li>
</ul>
```

Avec ce code JavaScript nous pouvons le réaliser :

```
// nous ciblons l'identifiant «info» et ajoutons la class active à son parent
// ici un élément de la liste
document.getElementById("info").parentNode.setAttribute("class", "active");
```

Nous avons vu le déplacement parent-enfant (haut - bas), mais vous pouvez également vous déplacer coter à côté dans le DOM et cibler des nœuds juste à côté. Ils sont les appeler *siblings* (rappel), et il existe deux méthodes pour les cibler : *previousSibling* et *nextSibling*.

```
// cible le parent de info et son frère précédent et lui ajoute une class
document.getElementById("info").parentNode.previousSibling.setAttribute("class", "previous");

// idem mais sélectionne le suivant
document.getElementById("info").parentNode.nextSibling.setAttribute("class", "next");
```

Le code modifier donne ceci :

```
<ul id="nav">
  <li class="previous"><a href="/" id="home">Accueil</a></li>
  <li class="active"><a href="/info" id="info">Info sur nous</a></li>
  <li class="next"><a href="/contact" id="contact">Contactez-nous</a></li>
</ul>
```

## 1. Cibler le premier et dernier enfant

Le JavaScript nous offre une autre route alternative pour cibler des nœuds, l'accès direct au premier ou dernier enfant.

```
// navigue au premier nœud et ajoute une class
document.getElementById("nav").firstChild.setAttribute("class", "premier");

// navigue au dernier nœud et ajoute une class
document.getElementById("nav").lastChild.setAttribute("class", "dernier");
```

Ce qui devrait donner comme code HTML :

```
<ul id="nav">
```

```
<li class="premier"><a href="/" id="home">Accueil</a></li>
<li> <a href="/info" id="info">Info sur nous</a></li>
<li class="dernier"><a href="/contact" id="contact">contactez-nous</a></li>
</ul>
```

# 8

## Modifier la DOM



Jusqu'à maintenant, nous avons travaillé avec des nœuds qui existait, souvent quand vous créer une interface, nous avons besoin d'insérer dynamiquement des nœuds dans votre DOM, créez de nouveaux éléments, les remplir avec du contenu, et de les placer quelque part dans votre document. Par chance nous avons cette capacité avec le JavaScript. Voici les 4 nouvelles méthodes :

- createElement()
- createTextNode()

- `appendChild()`
- `removeChild()`

```
<body>
  <div id="zoneCible">
    <p id="ligne">Bonjour !</p>
  </div>
</body>
```

## 1. Ajout d'élément

La création et l'insertion d'un nouvel élément se font en trois étapes :

1. Création de l'élément
2. Le remplir avec du contenu (ou vide)
3. Ajoute sur le DOM

Premièrement, nous avons besoin d'utiliser `createElement()` pour créer l'élément, cette méthode a un seul paramètre, celui-ci est le nom de la balise à ajouter. Quand j'écris que l'on crée un nouvel élément en fait il n'existe pas encore tant que nous ne l'avons pas ajouté au DOM.

La seconde étape est d'ajouter du contenu à cette balise virtuel, vous pouvez le faire tout de suite ou le faire plus tard ,mais il serait souhaitable d' ajouter quelque chose , à quoi peut servir un élément vide ? Nous allons utiliser la méthode `createTextNode()` pour faire cette opération. Il y a un seul paramètre : une chaîne de caractère qui sera ajouté à l'élément.

La dernière étape et de sélectionner l'endroit où nous allons insérer le nouvel élément. Nous réalisons ceci avec la méthode `appendChild()`. Cette méthode reçoit un seul argument, qui est l'élément complet à ajouter. Et par magie, vous avez généré votre nouvel élément dans le DOM.

```
// stocke le chemin de la cible dans une variable
var zoneCible = document.getElementById("zoneCible");

// Crédation d'une balise <p>
var p = document.createElement("p");

// Ajout un nœud text dans la balise <p>,
// nous utilisons la variable p
var nouvElement = p.createTextNode("Ceci est un paragraphe génère");

// Insertion du nouveau nœud dans le dom
zoneCible.appendChild(nouvElement);
```

Le code HTML modifier devrait ressembler à ceci :

```
<body>
  <div id="zoneCible">
    <p id="ligne">Bonjour !</p>
    <p> Ceci est un paragraphe génère </p>
  </div>
</body>
```

#### **4. Supprimer un élément.**

Comme vous pouvez ajouter facilement (oups!) un nouvel élément vous pouvez également en supprimer un facilement (re oups!) , en lieu et place de trois méthodes successives une seul suffira. Il faut des mois pour construire une maison et quelques heures pour la détruire.

```
// sauvegarde la cible pour une plus grande lisibilité du code
var zone_cible = document.getElementById("zoneCible");

// conserve également l'élément à supprimer de la zone cible
var aSupprimer = document.getElementById("asup");

// supprime du DOM
targetArea.removeChild(aSupprimer);
```

# 9

## Les données en Javascript



Stocker des données fait partie intégrale du JavaScript, comme n'importe quel langage de programmation. Dans ce chapitre nous allons voir et parfois revoir les différentes méthodes de stockage intégrées dans le JavaScript.

### 1. Variables

Les variables sont la version la plus basique de stockage de données en JavaScript. **Une variable est une étiquette à laquelle on lie une valeur**. Parfois la valeur change, mais l'étiquette ne change jamais, ce qui permet de l'utiliser dans tout le code.

Elle peut être de type string (chaîne), nombre ou booléen. La première chose à faire est de déclarer une variable ce qui est fait en déclarant un nom (étiquette) à la variable. Attention au nom de variable, le JavaScript est case sensitive (heu ! majuscules différentes des minuscules). Voici les différentes méthodes de déclaration de variable simple :

```
/* déclaration normale et conseillée*/
var monSandwichsPrefere

/* Certain utilise les soulignés comme séparateur */
var mon_Sandwichs_prefere
var MON_SANDWICHES_PREFERE // déconseillé

/* toutes ces variables sont différentes.*/
```

### 1.1. String (chaîne)

Le string est le premier type de donnée que vous pouvez stocker dans une variable. C'est une collection de caractères et peut être un simple mot, un paragraphe ou même de multiple paragraphe. Il est entouré de guillemets et assigné avec le signe «=»

```
var monSandwichsPrefere = "Américain piquant";
var maNouriturePrefere = "Suchy";
var maBierePrefere = "Chimay Bleu";
```

Les chaînes de caractères peuvent être entourées de simples ou doubles guillemets, c'est à vous de choisir. Il faut surtout savoir que si vous commencez avec des simples vous devez terminer avec des simples et idem pour les doubles.

*Voici des exemples de méthodes d'échappement :*

```
/* double quotes */
var monSandwichPrefere = "Américain piquant";

/* simple quotes */
var maNouriturePrefere = 'Suchy';

/* double quotes échappées */
var phrase = "J'ai dit, \"étudier est très important \", vous le savez ! .";

/* simple quotes échappées */
var test = 'ceci est une liste d\'exemples';
```

Vous avez bien compris le bazar ...

**Truc** : comme en PHP, comme l'HTML n'utilise que des doubles quote, il serait plus intéressant d'utiliser des simples en JavaScript.

### 1.2. Nombres

Les nombres sont exactement ce que vous pensez : ce sont des valeurs numériques qui sont stockées dans la variable. Le problème est que les nombres peuvent être aussi des caractères si entre guillemets! Le traitement sera différent.

```
/* stocker le nombre de sandwichs mangé */
var nbSandwichs = 3;

/* Prix des sandwichs */
var prixSandwichs = 2.5;

/* affiche le prix des sandwichs mangé */
alert(nbSandwichs * prixSandwichs);
```

### 1.3. Booléen

Les valeur booléenne sont simple , juste true (vrai) ou false (faux), il son utilisé principalement dans des procédures de test bon/pas bon.

Les valeurs ne sont pas entourées de guillemets.

```
var jaiFaim = true;
var jaiSoif : false;
```

En lieu et place du true/false vous pouvez utiliser 1/0, très pratique pour des tests.

## 2. Les Arrays (tableau)

La plupart des variables ne peuvent contenir qu'une seule valeur, dans beaucoup de cas il en faut plus. Nous allons utiliser des arrays. Le tableau est un ensemble de valeur stocker dans une seule variable, ces valeurs sont structurées simplement avec la possibilité de la retrouver avec un système d'index.

La déclaration se fait avec le mot clé new qui crée un nouveau type d'objet.

```
/* création avec le mot clé new */
var mesBoissonsFavorites = new Array();

/* création simplifiée */
var mesBoissonsFavorites = [];
```

Après la déclaration du tableau, vous avez besoin de le remplir. Si vous connaissez déjà les valeurs a stocker, vous pouvez les assignés au moment de la déclaration :

```
var jourSemaine = ["Lundi", "Mardi", "Mercredi", ...];
```

Comme les autres types de variables, vous pouvez stocker des chaînes, des nombres ou même des booléens.

### 2.1. Les tableaux simples

Comme dit précédemment, un tableau est un ensemble de valeur. L'index de ce tableau est le référent à la valeur.

```
var jourSemaine = ["Lundi", "Mardi", "Mercredi", "Jeudi", "Vendredi"];
/* l'index est créé automatiquement à la création du tableau */

// autre méthodes
```

```

var jourSemaine = [];
jourSemaine[0] = "Lundi";
jourSemaine[2] = "Mardi";

/* affiche le 2e élément du tableau (qui comme à 0) */
console.log(jourSemaine[1]); // mardi

```

**Attention :** l'index (indice) commence comme dans la plupart des langages de programmation à 0. Ne l'oubliez pas, car cela peut vous créer de sérieux problèmes de débogage.

## 2.2. Les tableaux associatifs

Si vous voulez donner plus de sens à vos données, vous pouvez utiliser les tableaux associatifs. La différence entre les tableaux simples et tableaux associatifs est que l'associatif a un index comme une chaîne de caractères. Actuellement, il est déconseillé d'utiliser ce type de tableau.

```

/* déclaration d'un tableau */
var lesRepasSemaine = [];

/* remplissage du tableau manuellement */
lesRepasSemaine["lundi"] = "Moules frites";
lesRepasSemaine["mardi"] = "pêche au thon";
lesRepasSemaine['mercredi'] = "assiette nordique ";

/* affichage du repas du mardi */
console.log(lesRepasSemaine["mardi"]);

```

## 2.3. Les tableaux multidimension

Les tableaux multidimension sont des tableaux avec des tableaux comme valeur.

```

/* Déclaration du déjeuner */
var dejeuner = ["tartine", "œuf au bacon", "salade de fruit"];

/* Déclaration des dîners */
var diner = ["Steak frite", "américain frite", "moules frites"];

/* Déclaration du souper */
var souper = ["Soupe", "Salade", "Yaourt"];

/* Combinaison des trois */
var menu = [dejeuner, diner, souper];

```

C'est une méthode pour stocker plus d'informations et de garder une structure. Voici la méthode pour récupérer le contenu :

```

console.log(menu[0][1]) // renvoi "œuf au bacon"
console.log(menu[1][0]) // renvoi "Steak frite"
console.log(menu[2][1]) // renvoi "Salade"

```

## 2.4. Ajouter des données dans un tableau

Maintenant que vous connaissez les différents types d'array et comment y stocker des valeurs, nous allons voir comment ajouter dynamiquement des valeurs dans un tableau.

```

/* Déclaration du déjeuner */
var dejeuner = ["tartine", "oeuf au bacon", "salade de fruit"];

/* Ajout d'un élément au tableau */
dejeuner.push("Pancake");

/* afficher tout le tableau */
alert(dejeuner);

/* Affichage du dernier ajouté */
alert(dejeuner[3]);

```

### 3. Les méthodes pour les tableaux

Le JavaScript nous fournit une autre série de méthode pour manipuler les tableaux :

- join()
- slice()
- shift()
- unshift()
- pop()
- concat()
- sort()

#### 3.1. Join()

La méthode `join()` est une manière de combiner tous les éléments d'un tableau et de l'extraire en une chaîne avec un séparateur choisi. Le séparateur peut être une chaîne de caractère.

```

var mesDiner = diner.join(", et ");
console.log("Mes dîners de la semaine : "+mesDiner);

```

résultat



#### 3.2. Slice()

La méthode `slice()` permet d'extraire certain élément d'un tableau existant, simplement en stipulant l'indice de départ et l'indice de fin, attention l'indice de fin n'est pas repris dans l'exportation.

```

/* Déclaration des dîners */
var diner = ["Steak frite", "américain frite", "moules frites"];

/* Extraction des deux premiers éléments */
var plusMeilleur = diner.slice(0,2);
alert(plusMeilleur.join(" & "));

```

Ce script devrait renvoyer : *Steak frite & américain frite*

### 3.3. *Shift()* & *unshift()*

La méthode *shift()* va supprimer le premier élément du tableau, ce qui veut dire que l'élément qui avait l'index 1 devient 0.

La méthode *unshift()* va-t-elle ajouter une donnée en première position du tableau.

```
/* Déclaration des dîners */
var diner = ["Steak frite", "américain frite", "moules frites"];

/* Suppression du Steak frite, étant le premier */
diner.shift();

/* Ajout d'un élément en première place */
diner.unshift("Waterzooi");
```

Notre tableau contiendra donc : Waterzooi , américain frite et moules frites.

### 3.4. *pop()*

La méthode *pop()* permet de récupérer le dernier élément du tableau et de le supprimer du tableau.

```
/* Déclaration des dîners */
var diner = ["Steak frite", "américain frite", "moules frites"];

/* Suppression et sauvegarde du dernier élément */
monDernierRepas = diner.pop();

/* affichage de la variable */
console.log(monDernierRepas);
```

**Note :** Les méthodes *shift()*, *unshift()* et *pop()* change physiquement la taille du tableau.

### 3.5. *concat()*

Concat et le diminutif de concaténer, cette méthode va copier et combiner de multiples tableaux en un seul tableau. Les tableaux originaux ne changent pas.

```
/* Déclaration du déjeuner */
var dejeuner = ["tartine", "oeuf au bacon", "salade de fruit"];

/* Déclaration des dîners */
var diner = ["Steak frite", "américain frite", "moules frites"];

/* Déclaration du souper */
var souper = ["Soupe", "Salade", "Yaourt"];

/* création de la nouvelle table avec les trois */
var mesRepas = dejeuner.concat(diner, souper);

/* affichage de contenu du tableau dans l'HTML */
document.write(mesRepas.join("<br>"));
```

Ici j'ai utilisé une autre méthode d'affichage, la méthode `document.write()` qui va directement écrire dans la page.

### 3.6. `sort()`

La méthode `sort()` est très utiles et intéressante, elle trie par défaut dans l'ordre alphabétique, attention que comme beaucoup d'algorithmes de tris, celui est imparfait, la chaîne "30" sera avant "4", il ne sait faire la différence entre un chiffre et une lettre. Par chance (ou malchance pour vous cher développer) il est possible d'ajouter en paramètre une fonction qui triera suivant votre besoin.

```
<body>
</body>
<script>
/* Déclaration du déjeuner */
var dejeuner = ["tartine", "oeuf au bacon", "salade de fruit"];

/* tri le tableau par défaut */
var dejeunerTrie = dejeuner.sort();

/* mémorise la cible */
var cible = document.getElementsByTagName("body")[0];

/* Affiche le résultat dans la cible *.
cible.innerHTML = dejeunerTrie.join("<br>");
</script>
```

Encore une nouvelle technique d'écriture dans la page, on mémorise l'élément `<body>` et on écrit dedans avec la méthode `innerHTML()`.

## 4. Dans un objet

Quand les données commencent à être complexes et difficiles à manipuler. Il est possible de se retourner vers les objets. Vous pouvez toujours sauvegarder des tableaux en objets, mais réaliser des tableaux multiples et complexes, vous pouvez créer des données plus facilement lisibles (a mon avis) avec des objets.

Voici un exemple avec notre repas :

```
/* sauvegarde de tous mes repas dans un objet */
var repas = {
    dejeuner : ["tartine", "oeuf au bacon", "salade de fruit"],
    diner : ["Steak frite", "américain frite", "moules frites"],
    souper : ["Soupe", "Salade", "Yaourt"],
};

/* affichage d'un élément */
document.write(repas.diner[2]); // moules frites
```

Un object, de fait, ne sont pas obligé de contenir des tableaux telle que l'exemple précédent, il peut contenir des chaînes de caractère, nombres, valeur booléenne, ou aussi des fonctions.

```
/* sauvegarde les ingrédients sandwich jambon/fromage */
var sandwich {
    pain : "baguette",
```

```

        viande : "Jambon",
        fromage : "Gouda jeune",
    };
    /* affiche le type de fromage */
    alert(sandwich.fromage); // Gouda jeune

```

## 5. Json

Le JSON est une nouvelle méthode de stockage en JavaScript, elle est très utilisée depuis de nombreuses années. Elle se définit comme JavaScript Object Notation et est un format très léger pour l'échange entre le serveur et le client. Elle n'est pas que facile à lire et écrire pour nous, mais aussi pour les machines (pas si bête la bête). Tous les langages modernes supportent le JSON. Voici un exemple :

```

var mesSandwichFavory = {
    "diner" : [
        {
            "nom" : "Fromage jambon",
            "pain" : "Blanc"
        },
        {
            "nom" : "Poulet curry",
            "pain" : "Entier"
        }
    ],
    "souper" : [
        {
            "nom" : "Américain",
            "pain" : "Gris"
        }
    ]
};
/* récupération du pain du premier dinер */
document.write(mesSandwichFavory.diner[0].pain); // Blanc

```

### 5.1. Les avantages du JSON

Il existe une flopée de formats de donnée en JavaScript, pour laquelle vous devriez choisir le JSON à la place d'un autre? D'abord par ce que le JSON est très lisible, ce qui est un grand avantage quand vous voulez communiquer autour de vous et vérifier ce qui se passe.

Le JSON est utilisé avec beaucoup de service externe pour le transfert de données, car est compatible multiplate-forme et domaine. Avant le JSON on avait le XML, mais celui-ci était difficilement utilisable en JavaScript sans passer par un proxy. Comme le JSON est plus léger, il peut être transférer plus facilement, très utile avec la technologie Ajax.

### 5.2. Utilisation d'une API

API veux dire **A**pplication **P**rogramming **I**nterface. Plein de service professionnel vous donne accès a une API pour développer et étendre votre possibilité de coder. *Flickr* a une API pour vous pour ajouter des images et données dans votre propre site et *Netflix* fait le même, mais pour les vidéos. Beaucoup d'autres sites fonctionnent de la même manière (*google map*, *YourTube*, etc.). Pour le moment il est important de comprendre comme une API fonctionne, et qu'il y a une méthode sûre pour accéder aux informations stockées dans ces autres sites.

Quand vous réalisez une requête à *Netflix* par son API, nous feriez (en JavaScript) cela comme ceci : «Donne-moi ma liste des films avec comme acteur *tom Hawk*» et l'API de *Netflix* devrait vous répondre avec une liste des films avec cet acteur, comme si l'information était stockée dans votre site. Le retour de l'information sera en JSON.

La plupart des API modernes utilisent cette technique, vous réalisez votre requête et de retour vous recevez des données au format JSON. Si vous utiliser du JavaScript, du Python, PHP, Ruby les données ne changeront pas (ce qui est très pratique). Les API sont sûrement les plus grandes utilisatrices du JSON pour le moment.

# 10

## Stockage avec l'HTML5



Jusqu'à maintenant de notre évolution en JavaScript , nous avons parlé de stockage de données du coté développeur ( le vôtre). Il est peut-être temps de gérer des données du côté utilisateur.

Dans le passé nous utilisions les cookies pour sauvegarder des informations de l'utilisateur, mais les cookies sont de gros salop... Elles ne sont pas sécurisées , permette d'être utilisé par des malwares, vous affiche de la pub ciblée suite au cookie installé (google !). De plus les utilisateurs surfent de plus en plus en bloquant les cookies.

Le W3C connaît le problème et dans les nouveaux standards du HTML5 nous permet le WebStorage avec deux nouvelles technologies : les objets `localStorage` et le `sessionStorage`, ce qui va nous permettre de mieux interagir avec les données stockées de l'utilisateur.

## 1. LocalStorage et sessionStorage

Comme mentionné, stocké des informations localement dans le navigateur de l'utilisateur date du début du web et cookies. Parfois pour simplement dire que l'on est déjà passé par ici parfois pour des données actuelles (nom, adresse, etc..) ajouter par certains événements? C'est un outil très utile pour personnaliser l'expérience de l'utilisateur.

En HTML5, nous avons accès au stockage Web avec deux objets: `localStorage` et le `sessionStorage`. Ces objets ont trois méthodes intégrées : `setItem`, `getItem` et `removeItem`. Regardons l'utilisation de celle-ci.

### 1.1. `setItem()`

La méthode `setItem()` reçoit deux paramètres . Le premier est équivalent à l'étiquette d'une variable; ou le nom de ce que vous voulez stocker. Le second paramètre est la valeur, qui doit être une chaîne. L'appel de `setItem()` avec local/sessionStorage va stocker la donnée localement avec l'utilisateur.

```
/* enregistre dans localStorage */
localStorage.setItem("monDiner", "Moules frites");
```

### 1.2. `getItem()`

Après avoir sauvegardé les informations, il est naturel que dans certains cas, il nous faille les récupérer. Pour cela nous avons la méthode `getItem()` qui n'a qu'un seul paramètre, le nom de la variable à récupérer.

```
/* récupère dans localStorage */
var recup = localStorage.getItem("monDiner");

/* affiche le résultat */
alert(recup); // Moules frites
```

### 1.3. `removeItem()`

Ma mère m'a toujours dit de nettoyer après mon passage (respect du suivant) et je garde cela en mémoire même en programmant. Le W3C a pensé aussi à cette méthode, le `removeItem()` va supprimer la donnée de la machine du client (respect!). Comme la méthode précédente, celle-ci n'a besoin que d'un seul paramètre, le nom de la variable à supprimer

```
/* supprime du localStorage */
localStorage.removeItem("monDiner"); // ma mère va être contente
```

Les deux objets local et sessionStorage utilisent la même syntaxe et méthode. La seule différence entre les deux objets est ce que `localStorage` garde les informations sur la machine client tant que nous ne les avons supprimés (le développer ou l'utilisateur) en passant par les préférences du navigateur. Il n'est pas simple pour un utilisateur de faire cette opération. Le `sessionStorage`, lui est plus simple, car toutes les infos disparaissent à la fermeture de la session (ou le navigateur est fermé).

## 2. Stocker plus de données en JSON

Le web storage ne permet que de sauvegarder des chaînes. Cela peut être problématique quand vous voulez sauvegarder des différents types de données. Vous pouvez passer outre de ce problème en utilisant un objet JSON (encore lui!) et d'utiliser la méthode `JSON.stringify` pour convertir l'objet complet en une chaîne qui pourra être stocké dans le navigateur.

```
/* transforme l'objet JSON diner en une chaîne */
var stringDiner = JSON.stringify(mesSandwichFavoris);

/* sauvegarde dans le navigateur */
localStorage.setItem("sandwich",stringDiner);
```

Maintenant que cet objet JSON est une chaîne de caractère, et que nous ne pouvons plus extraire facilement une des informations, il existe une autre méthode pour convertir la chaîne en objet JSON, la méthode `JSON.parse`.

```
/* récupère les données du navigateur */
var infoStocké = localStorage.getItem("sandwich");

/*Conversion en objet JSON */
var infoConvertie = JSON.parse(infoStocké);

/* Affichage pour prouver le processus */
console.log(infoConvertie.diner[0].nom); Fromage jambon
```

### 3. Compatibilité

Comme le webStorage est récent en HTML5, il ne sera utilisable qu'avec les versions les plus récentes des navigateurs.

Navigateur	Version min
IE	8.0
Firefox	3.5
Chrome	4.0
Opera	10.5
Safari	4.0

Comme vous pouvez le voir dans le tableau ci-dessus, cette technologie n'est pas compatible avec tous les navigateurs. Ceci avec l'envie d'avoir un site fluide et convivial pour l'utilisateur nous renvoie sur la possibilité de détections de compatibilité.

L'idée principale est de détecter la présence de cette technologie pour pouvoir l'utiliser, si celle-ci est inexistante il faudra utiliser une alternative avec les anciennes méthodes ou pourquoi pas avec la partie serveur.

```
/* teste le support pour localStorage */
if (typeof(localStorage) == "undefined" ) {
    // le localStorage n'est pas reconnu , cookies ?
} else {
    // ajouter votre code utilisant le localStorage
}
```



# 11

## Les événements



Dans ce chapitre, nous allons apprendre comment interagir avec l'utilisateur en créant et exécutant des événements. Pour le moment vous (en tant que développeur) vous avez le plein contrôle de toutes les conditions quand une fonction doit être lancé. Vous avez appris à réaliser des fonctions, faire des boucles, stocker des données et naviguer dans le DOM (oufty ! beaucoup de choses). Tous les contrôles d'interactions sont entre vos doigts. La plupart des fonctionnalités background fonctionnent comme cela, mais ce n'est que la moitié de la bataille. Vous ne devez pas toujours avoir le contrôle de ces actions. Vous avez plus souvent besoin que ce soit l'utilisateur qui décide quelles actions doivent être prises. On appelle cela la gestion des événements.

Un évènement est toujours lié à un élément du DOM pour être exécuté, voici le modèle de fonctionnalité :

- Cibler le nœud du DOM
- Attacher un évènement à ce nœud
- Exécuter une fonction définie au déclenchement de l'évènement.

## 1. Attacher un évènement (event)

Maintenant que nous pensons évènements, nous allons voir comment l'attacher à un élément du DOM.

Il existe plusieurs façons de lier un évènement à un nœud du DOM. Certaines sont mauvaises (utilisation du JavaScript dans une balise HTML), d'autres sont moins mauvaises et comme par hasard, d'autres sont plus meilleures. Nous allons parler de 2 méthodes

- event handler (dérouleur d'évènement)
- event listeners (auditeurs d'évènement)

### 1.1. Event handler

L'event handler est une des méthodes les «pas trop mauvaise» pour attacher un évènement à un élément du DOM. Cette méthode a juste des retours, qui sont à traiter en utilisant d'autres méthodes.

Elles sont très faciles à mettre en œuvre, car très lisibles (pour l'humain que nous sommes)

```
<button type="button" id="btn">cliquez-moi</button>
<script>
/* assemblé le tout dans une fonction anonyme */
(fonction () {
    // sauvegarde le nœud bouton dans une variable
    var btn = document.getElementById("btn");

    // attache un event handler «le click»
    btn.onclick() {
        // affiche un message au click
        alert("vous avez cliquer sur le bouton");
    }
})();
</script>
```

Quand vous utilisez les event handler, vous le préfacerez toujours du mot «on». Voilà le pourquoi de l'utilisation du onclick en lieu et place du click, ceci est bon pour tous les événements : onsubmit, onchange, onfocus, onblur, onmouseover, onmouseout, ontouchstart, ongesturestart etc.

La plupart du temps les events handler sont bons, ils ont une bonne compatibilité dans les différents navigateurs et font partie du standard. Mais vous ne pouvez pas mettre plusieurs événements sur un seul nœud du DOM. Cela est parfois important par exemple pour le contrôle d'un formulaire et l'envoi en Ajax du formulaire, le tout sur le bouton submit.

### 1.2. Event listeners

Les event listener ont une utilisation similaire au event handler, vous avez toujours besoin d'un nœud du DOM auquel vous allez l'attacher, vous avez toujours besoin de définir le type d'évènement à traiter, et vous devrez appeler une fonction. La grosse différence c'est qu'il n'y a plus de limite du nombre d'évènement attaché à un nœud.

```
// un event listener avec une fonction anonyme
element.addeventListener("event", fonction (){
    // chose à faire à l'évènement
}, false);
```

Un event listener est fait de 4 parties :

- L'élément du DOM (`element`)
- Le type d'évènement (`event`)
- L'action à réaliser, dans notre cas une fonction anonyme
- Un booléen utilisé pour initialiser ce que l'on va appeler «capture»

La «capture» qui est à `false` dans 99% des cas permet de ne pas propager l'évènement à tout le DOM. Si vous le changez à `true`, tous les éléments du DOM seront au courant.

```
<button type "button" id="btn">un bouton</button>
<script>
(function() {
    // ajoute un event listener click sur le bouton
    var btn = document.getElementById("btn");
    btn.addeventListener("click", fonction (){
        alert("tu a cliquez sur le gros bouton");
    }, false);
})();
</script>
```

### 1.3. Compatibilité

Juste *Internet Explorer 8* et version précédente n'est pas compatible avec cette méthode. Par chance il existe une méthode très ressemblante `attachEvent()` mais qui n'a pas le troisième paramètre. Il vous faudra donc tester le navigateur avant de pouvoir agir.

```
var btn = document.getElementById("btn");

// test de la compatibilité
if ( btn.addEventListener ) {
    // version compatible
    btn.addeventListener("click", fonction (){
        alert("tu as cliqué sur le gros bouton ");
    }, false);
} else {
    // version IE 8 et -
    btn.attachEvent("click", fonction (){
        alert("tu a cliquez sur le gros bouton");
    });
}
```

Voilà un système compatible avec tous les navigateurs.

Quand vous attachez des évènements aux éléments du DOM, il y a certaines informations à connaître.

Nous avons jusqu'à maintenant utiliser des fonctions anonymes, si vous utilisez une fonction extérieur, dans la liste des paramètres il ne faudra surtout ne pas mettre de parenthèse à la fin du non de la fonction.

```
var btn = document.getElementById("btn");

/* création de la fonction */
function message() {
    alert("tu a cliquez sur le gros boutton");
}

/* ajout de l'évènement */
btn.addEventListener("click", message, false);
```

Comme nous l'avons vu précédemment une fonction peut recevoir des paramètres, dans l'utilisation telle que l'exemple ci-dessus il est impossible, car nous n'avons pas de parenthèse. Le seul moyen est d'insérer la fonction paramétrée dans une fonction anonyme (cela se complique un peu!)

```
var btn = document.getElementById("btn");

/* création de la fonction */
function message(texte) {
    alert(texte);
}

/* ajout de l'évènement avec une fonction anonyme */
btn.addEventListener("click", function() {
    // appel de la fonction avec un paramètre
    message("tu as cliqué sur le gros bouton ");
}, false);
```

## 2. Détailler un évènement

Suivant les actions sur le site, il est parfois intéressant de supprimer une gestion d'évènement sur un élément du DOM. Comme (et vous ne serez pas surpris) IE8 et - a aussi sa propre méthode pour supprimer l'attachement d'un évènement à un nœud.

```
var btn = document.getElementById("btn");

// test de la compatibilité
if ( btn.removeEventListener ) {
    // version compatible
    btn.removeEventListener("click",message, false);
} else {
    // version IE 8 et -
    btn.detachEvent("click", message);
}
```

## 3. Événement clavier et souris

Les évènements les plus courants, autres que le load, que vous allez utiliser sont ceux liés à la souris ou au clavier. Voici une liste non exhaustive

- click
- focus
- blur
- change
- mouseover
- mouseout
- submit

Il existe un grand nombre de types d'évènement en JavaScript. Certains que vous n'utilisez jamais, comme le double click. La liste précédente sont les plus utilisés et que vous utiliserez sûrement le plus.

Nous allons réaliser une application de gestion de contact. Voici le code HTML de base :

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <title>Mes contacts</title>
    <style>
        .active { background: #ddd; }
        .auDessus { background: #eee; }
        form > div { padding: 10px; }
    </style>
</head>
<body>
    <h1>Mes contacts</h1>
    <form action="" method="get" id="formRecherche">
        <div class="section">
            <label for="q">Recherche</label>
            <input type="search" id="q" name="q" required placeholder="a rechercher">
        </div>
        <div class="button-group">
            <button type="submit" id="btnRech">recherche</button>
            <button type="button" id="Tous">Tous</button>
        </div>
    </form>
    <div id="cible"></div>
    <script src="js/1101.js"></script>
</body>
</html>

```

Vous pouvez remarquer dans l'HTML que j'ai préparé l'interface en ajoutant une div vide qui sera remplis par le JavaScript.

Voici les données JSON de nos contact ( a taper dans le fichier js/1101.js):

```

var contacts = {
    "address" : [
        {
            "nom": "hillisha",
            "email": "hill@example.com",
        },
        {
            "nom": "Paul",
            "email": "cleveland@example.com",
        },
        {
            "nom": "vishaal",
            "email": "vish@example.com",
        },
        {
            "nom": "mike",
            "email": "grady@example.com",
        },
        {
            "nom": "jamie",
            "email": "dusted@example.com",
        }
    ]
};

```

### 3.1. Click

L'évènement click est le plus utilisé des différents types d'action de l'utilisateur.

Voici la gestion du premier bouton : (code à ajouter a la suite de vos données en JSON)

```

/* Définition d'un objet qui contiendra les méthodes */
var adr = {
    toutAfficher : function() {
        /* sauvegarde le chemin de la cible */
        var target = document.getElementById("cible");
        /* sauvegarde les adresses dans une variable */
        var book = contacts.address;
        /* enregistre le nb de contacts pour une bonne lisibilité */
        var count = book.length;
        /* près à lire chaque ligne! */
        var i;
        /* vide la zone cible au cas où */
        target.innerHTML = "";
        /* Test si il y a du monde */
        if(count > 0){
            /* boucle dans les contacts */
            for(i = 0; i < count; i++) {
                var obj = book[i];
                target.innerHTML += '<p>' + obj.nom + ', <a href="mailto:' + obj.email + '">' + obj.email + '</a></p>';
            } // Fin de la boucle
        } // fin du test
    } // fin de la méthode toutAfficher
} // fin object

```

```
// mémorise le chemin du bouton tous
var btn = document.getElementById("Tous");

// ajoute l'évènement au bouton
btn.addEventListener("click", adr.toutAfficher, false);
// fin du fichier js/1101.js
```



Voici le résultat suite au click sur le bouton tous, la méthode `afficheTous` est lancée et avec une boucle qui insère ligne par ligne les contacts avec la méthode `innerHTML`.

### 3.2. focus & blur

Avec l'évènement Click, l'utilisation du `focus` et `blur` sont les évènements que vous allez utiliser dans une application JavaScript. Le Web est rempli de liens et de formulaire et ces objets sont les plus pratiques pour utiliser les évènements `focus` et `blur`. Un bon exemple du `focus`/`blur` est l'activation d'un champ de recherche. Soit en cliquant dans le champ ou en utilisation la touche TAB pour naviguer et en activé le `focus`, et par l'inverse en quittant le champ (soit en cliquant dans un autre) va activé l'évènement `blur`.

Dans notre formulaire nous allons ajouter avec les évènements `focus` et `blur` un renforcement visuel. Le renforcement visuel est une méthode qui va renforcer l'attention sur la zone ciblée, dans notre cas nous allons mettre du gris autour du champ de recherche quand celui-ci est actif. Attention d'ajouter une virgule après la méthode `toutAfficher`

```
, // fin de la méthode toutAfficher

// définit la méthode pour activé la zone
ajouteActive : function() {
    // ajoute la class active à la div définie
    this.parentNode.setAttribute("class", "active");
}, // Fin de la méthode : Attention à la virgule

supprimeActive : function() {
    // Retire la class active à la div définie
    this.parentNode.removeAttribute("class");
}
```

et ceci après l'objet adr :

```
// Mémorise l'élément champs de recherche
var champDeRecherche = document.getElementById("q");

// active la gestion du focus sur l'élément
```

```

champDeRecherche.addEventListener("focus", adr.ajouteActive, false);

// active la gestion du blur sur l'élément
champDeRecherche.addEventListener("blur", adr.supprimeActive, false);

```

qui sont l'activation des gestions d'évènements sur le champ de recherche

### *3.3. L'accessibilité*

Il y a une caractéristique importante associée avec le focus et comment il est en relation avec l'accessibilité. Vous pouvez remarquer que le listener va également s'exécuter quand on clique dans la zone de l'étiquette . Il y a trois façons différentes d'activer la zone de recherche :

- Cliquer dedans.
- Cliquer sur l'étiquette devant ( si l'attribut for est identique à l'Id)
- Utiliser la navigation avec la touche Tab.

Garder ces trois accès vous assurera une accessibilité complète et une meilleure expérience pour l'utilisateur.

### *3.4. change*

L'évènement change est applicable sur des éléments du formulaire comme menus select, bouton radio, et check box.

Dans les radios et check box, un changement de valeur se fera lors du passage en checked d'un box/button. Ce-  
la va de même avec l'étiquette et la navigation avec Tab.

Enter/return dans un select menu, l'évènement change sera activé lorsqu'il a une nouvelle valeur ou option sélectionner par un clique ou par navigation avec le clavier.

Dans notre petit exercice, il n'y a pas ce type d'objet, mais il est très facile de le traiter.

### *3.5. mouseover & mouseout (survol)*

L'évènement mouseover est actif au survol du curseur au-dessus d'un élément, le mouseout est lui l'inverse, il est actif au départ du curseur de l'élément survolé. L'association des deux permet de gérer le survol d'un élément.

Nous allons ajouter une gestion de survol sur notre formulaire , il existe déjà une CSS dans notre page HTML :

```

},
ajouteHover : function(){
    // ajoute une class au formulaire
    formulaire.setAttribute("class", "auDessus");
}, // fin de ajouteHover

supprimeHover: function(){
    // supprime-la class au formulaire
    formulaire.removeAttribute("class");
} // fin de supprimeHover
} // fin object

```

n'oublie pas les virgules entre les déclarations des méthodes

et ceci à la fin du code

```
// sauvegarde le nœud formulaire
var formulaire = document.getElementById("formRecherche");

// active le survol du formulaire
formulaire.addEventListener("mouseover", adr.ajouteHover, false);

// supprime le survol du formulaire
formulaire.addEventListener("mouseout", adr.supprimeHover, false);
```

### 3.6. Submit

L'évènement submit est activé à la soumission du formulaire ou avec la navigation Tab et return/enter sur le bouton.

Dans notre petit exercice, l'évènement submit va réaliser une recherche du nom dans les données JSON. Nous allons utiliser la méthode `indexof()` qui permet de rechercher dans une chaîne de caractère, si vous faites une recherche de la chaîne «i», tous les contacts qui ont un «i» dans leur nom sera trouvé, cette méthode renvoi -1 si elle ne trouve rien.

```
, // fin de supprimeHover

// Définition de la méthode de recherche
recherche : function(event) {
    // Sauvegarde la valeur à chercher
    var valeurRecherche = champDeRecherche.value,
        i;
    // supprime la gestion par défaut
    event.preventDefault();
    /* Sauvegarde le chemin de la cible */
    var target = document.getElementById("cible");
    /* Sauve les contacts dans une variable */
    var book = contacts.address;
    /* Enregistre le nb de contacts pour une bonne lisibilité */
    var count = book.length;
    /* prêt-à boucler! */
    var i;
    // vide la cible au cas où
    target.innerHTML = "";
    // vérifie si il a des contacts
    if(count > 0 && valeurRecherche !== ""){
        // boucle dans les contacts
        for (i = 0; i < count; i = i + 1) {
            // test le contenu si on retrouve la valeur à chercher
            var obj = contacts.address[i],
                trouver = obj.nom.indexOf(valeurRecherche);
            // si différent de -1 on écrit dans la cible
            if (trouver !== -1) {
                target.innerHTML += '<p>' + obj.nom + ', <a href="mailto:' + obj.email + '">' + obj.email + '</a><p>';
            } // fin du trouvé
        } // fin de la boucle
    } // fin de la condition
} // fin de la fonction recherche
```

```

        } // fin de la boucle
    } // fin du test comptage
} // fin de la méthode recherche

} // fin object

```

**Attention** d'ajouter la virgule à la fin de la méthode précédente ce code.

```

// active le submit du formulaire
formulaire.addEventListener("submit", adr.recherche, false)

```

### 3.7. Changer les évènements par défaut

Changer le comportement par défaut de certains éléments du DOM est parfois intéressant en JavaScript. Par exemple, un lien avec la balise href va changer de page, ou comme dans notre exercice précédent changer le comportement de l'envoi du formulaire.

Pour pouvoir changer le comportement, vous devez l'inclure dans la nouvelle méthode qui va être utilisée. Ceci se réalise avec la méthode `preventDefault()`.

```

recherche : function(event) {
    // supprime la gestion par défaut
    event.preventDefault();
    // la suite de votre code
}

```

### 3.8. keydown, keypress et keyup

Ces trois méthodes font référence à la gestion des touches du clavier, je ne vous les explique pas (un peu d'Anglais). Ces évènements sont souvent utilisés pour l'autocomplissage d'un champ de formulaire (style google). Et nous allons modifier notre petite application pour la rendre encore plus conviviale.

L'ordre des évènements est celui-ci

- `keydown`
- `keypress`
- `keyup`

Ici la modification sera très simple : juste changer le type d'évènement a utilisé pour la recherche :

```

// gestion du keyup dans la soumission du formulaire
formulaire.addEventListener("keyup", adr.recherche, false)

```

Cool ! Il ne manque plus qu'un peu d'Ajax pour aller chercher les adresses dans une base de données ....

Voici le code complet (mis à jour)

```

var contacts = {
    "address" : [
        {
            "nom": "hillisha",
            "email": "hill@example.com",
        },
        {

```

```

        "nom": "Paul",
        "email": "cleveland@example.com",
    },
    {
        "nom": "vishaal",
        "email": "vish@example.com",
    },
    {
        "nom": "mike",
        "email": "grady@example.com",
    },
    {
        "nom": "jamie",
        "email": "dusted@example.com",
    }
}

];
};

// Sauvegarde des nœuds utiles *****/
// mémorise le chemin du bouton tous
var btn = document.getElementById("Tous");
// mémorise l'élément champs de recherche
var champDeRecherche = document.getElementById("q");
// sauvegarde le nœud formulaire
var formulaire = document.getElementById("formRecherche");

/* Définition d'un objet qui contiendra les méthodes */
var adr = {
    toutAfficher : function() {
        /* sauvegarde le chemin de la cible */
        var target = document.getElementById("cible");
        /* sauvegarde les adresses dans une variable */
        var book = contacts.address;
        /* enregistre le nb de contacts pour une bonne lisibilité */
        var count = book.length;
        /* près à lire chaque ligne! */
        var i;
        /* vide la zone cible au cas où */
        target.innerHTML = "";
        /* Test s'il y a du monde */
        if(count > 0){
            /* Boucle dans les contacts */
            for(i = 0; i < count; i++) {
                var obj = book[i];
                target.innerHTML += '<p>' + obj.nom + ', <a href="mailto:' + obj.email + '">' + obj.email + '</a></p>';
            } // Fin de la boucle
        } // fin du test
    }, // fin de la méthode toutAfficher

    ajouteActive : function() {
        // ajoute la class active a la div définie
        this.parentNode.setAttribute("class", "active");
    } // Fin de la méthode : Attention à la virgule
}

```

```

supprimeActive : function() {
    // retire la class active à la div définie
    this.parentNode.removeAttribute("class");
},
ajouteHover : function(){
    // ajoute une class au formulaire
    formulaire.setAttribute("class", "auDessus");
},
// fin de ajouteHover

supprimeHover: function(){
    // supprime-la class au formulaire
    formulaire.removeAttribute("class");
},
// fin de supprimeHover

recherche : function(event) {
    // sauvegarde la valeur à chercher
    var valeurRecherche = champDeRecherche.value,
        i;
    // supprime la gestion par défaut
    event.preventDefault();
    /* sauvegarde le chemin de la cible */
    var target = document.getElementById("cible");
    /* sauve les contacts dans une variable */
    var book = contacts.address;
    /* enregistre le nb de contacts pour une bonne lisibilité */
    var count = book.length;
    /* prêt-à boucler! */
    var i;
    // vide la cible au cas où
    target.innerHTML = "";
    // vérifie si il a des contacts
    if(count > 0 && valeurRecherche !== ""){
        // boucle dans les contacts
        for (i = 0; i < count; i = i + 1) {
            // test le contenu si on retrouve la valeur à chercher
            var obj = contacts.address[i],
                trouver = obj.nom.indexOf(valeurRecherche);
            // si différent de -1 on écrit dans la cible
            if (trouver !== -1) {
                target.innerHTML += '<p>' + obj.nom + ', <a href="mailto:' + obj.email + '">' + obj.email + '</a><p>';
            } // fin du trouvé
        } // fin de la boucle
    } // fin du test comptage
} // fin de la méthode recherche

} // fin object

***** Gestion des évènements *****/
// Ajoute l'évènement au bouton
btn.addEventListener("click", adr.toutAfficher, false);
// active la gestion du focus sur l'élément

```

```

champDeRecherche.addEventListener("focus", adr.ajouteActive, false);
    // active la gestion du blur sur l'élément
champDeRecherche.addEventListener("blur", adr.supprimeActive, false);
    // active le survol du formulaire
formulaire.addEventListener("mouseover", adr.ajouteHover, false);
    // supprime le survol du formulaire
formulaire.addEventListener("mouseout", adr.supprimeHover, false);
    // active le submit du formulaire
//formulaire.addEventListener("submit", adr.recherche, false)
    // gestion du keyup dans la soumission du formulaire
formulaire.addEventListener("keyup", adr.recherche, false)

```

## 4. Gestion tactile et d'orientation

La gestion d'évènement du touché et de l'orientation est un peu intimidant, car nouveaux pour vous et très récemment ajouté au JavaScript. Il s'utilise de la même façon que les autres et peut utiliser les mêmes méthodes.

Comme le projet précédent était complet, nous allons réaliser un nouvel exercice:

```

<!doctype html>
<html lang="fr">
<head>
    <title>gestion tactile</title>
    <meta charset="utf-8">
    <style>
        body { min-height:600px; background:#ddd; }
    </style>
</head>
<body>
    <h1>Démo de la gestion tactile</h1>
    <!-- JS à la fin pour une plus grande performance-->
    <script src="js/1102.js"></script>
</body>
</html>

```

### 4.1. touchstart et touchend

Ces deux événements se ressemblent beaucoup, mais sont opposés. Le touchstart va être activé au contact avec l'écran, tandis que le touchend se fera uniquement au départ du doigt ou stylet de l'écran.  
Nous allons créer un nouvel objet.

```

/* encore une fonction anonyme */
(function(){
    var body = document.getElementsByTagName("body")[0];
    // Déclaration de l'objet pour la gestion des événements
    var touchControls = {
        toucheEcran : function(){
            // envoi un message dans le body
            body.innerHTML += "Tu me touches RRrrrr!<br>";
        },
        lacheEcran : function(){
            // envoi un autre message au body
            body.innerHTML += "SVP ne le fais plus.<br><br>";
        }
    };
    body.addEventListener("touchstart", touchControls.toucheEcran);
    body.addEventListener("touchend", touchControls.lacheEcran);
});

```

```

        }
    } // fin object

    // ajoute la gestion des deux évènements.
    body.addEventListener("touchstart", touchControls. toucheEcran, false);
    body.addEventListener("touchend", touchControls. lacheEcran, false);
}();
// fin fichier js/1102.js

```

Tester ... heu il faut le faire sur une tablette ou smartphone

#### 4.2. touchmove

L'évènement touchmove est enclenché lorsque l'utilisateur glisse son doigt glisse sur l'écran. Il est toujours précédé d'un touchstart. Cela est souvent utilisé pour déplacer des objets sur l'écran ou le glisser.

Ajoutons la méthode et l'activation dans notre code :

```

var touchControls = {
    /* les anciennes méthodes */
},
montreMouvement : function(){
    // envoi d'un message à l'écran
    body.innerHTML += "déplacement !!<br>";
} // fin du montreMouvement
} // fin object
// ajout de la gestion du déplacement
body.addEventListener("touchmove", touchControls.montreMouvement, false);
}();
// fin fichier js/1102.js

```

#### 4.3. orientationchange

Cet évènement est le seul qui est dans cette catégorie, mais sans toucher (on ne rigole pas) . Cet évènement est relié à la présence d'un accéléromètre dans l'appareil.

Il y a des réglages pour le mode portrait, paysage, portrait à l'envers et paysage ont l'envers.

Ajoutons cette méthode qui va nettoyer l'écran.

```

},
changedOrientation : function(){
    // efface le contenu du body
    body.innerHTML = "";
} // fin méthode changedOrientation

```

Ajoutons encore la liaison de l'évènement au body :

```

// Ajout l'évènement au body
body.addEventListener("orientationchange", touchControls.changedOrientation, false);

```

#### 4.4. Compatibilité

Le support ne sera possible que sur des appareils qui ont des écrans tactiles et la gestion de l'orientation uniquement s'il existe un accéléromètre dans celui-ci.



# 12

## Communiquer avec l'Ajax



Dans ce chapitre nous allons rentrer dans le Nouveau Monde de l'Ajax. Nous allons adapter les codes précédents et ajouter un niveau de communication avec les serveurs pour y récupérer des données à utiliser en JavaScript dans le navigateur.

À partir de maintenant vous devrez travailler sur un serveur en ligne ou en local pour pouvoir faire fonctionner les requêtes et récupérer des données du serveur.

### 1. Un peu d'histoire

L'Ajax est un concept de rafraîchissement d'une partie du document HTML sans recharger l'entièreté de la page. En 1999 Microsoft (et oui ils ont parfois fait des trucs intéressants!) a intégré l'objet `XMLHttpRequest` au JavaScript. Pour envoyer des mails avec access 2000 en passant par Internet Explorer 5 et un autre objet qui est ActiveX. Plus tard une compagnie nommée Mozilla développa l'objet `XMLHttpRequest` qui ne demandait plus la technologie ActiveX de Microsoft. Tous les navigateurs ont ajouté cet objet, Microsoft le fera à partir de la version 7 de son navigateur, l'Ajax n'est donc pas un nouveau langage, mais simplement du JavaScript. Il a été nommé pour la première fois en 2005 par un journaliste Jesse James Garrett.

Ajax vient à la base de «Asynchronous JavaScript and XML». Il permet de

- être synchrone ou asynchrone
- peu utilisé du XML, JSON texte ou du HTML comme source de données.

Mais maintenant cela représente plutôt des technologies de communication client/serveur

## 2. Communication avec le serveur

La communication avec le serveur est le but de l'Ajax. Le but est d'envoyer et recevoir des informations entre le client et le serveur pour générer une expérience utilisateur plus convivial. Avant la technologie Ajax, tout le travail se générerait sur le serveur et remplissait une partie de page en passant par un iframe (pas terrible) ou une page complète.

L'Ajax fournit deux méthodes de communication

- Synchrone
- Asynchrone

### 2.1. Synchrone

On dit que le script est exécuté de façon synchrone : quand un appel externe au script principal est réalisé, le script en attend la réponse ou la fin de l'exécution.

### 2.2. Asynchrone

Le contraire de synchrone est asynchrone. Quand un appel est asynchrone, le script principal n'attend pas d'avoir reçu les données pour continuer.

## 3. XMLHttpRequest

La méthode `XMLHttpRequest` est le cœur de l'Ajax. Nous allons ajouter cette technologie dans notre moteur de recherche d'adresse que nous avons réalisée dans les chapitres précédents.

La première étape est de créer une instance de `XMLHttpRequest` et cela est très facile :

```
var xhr = new XMLHttpRequest();
```

### 3.1. La compatibilité

Souvenez-vous, certain ancien navigateur (comme dab de Microsoft) utilise un autre objet : XMLHttpRequest. Ce sont de vieux navigateur, mais toujours utilisé par certaines personne. Voici la technique a utiliser pour rendre la technologie compatible :

```
if ( window.XMLHttpRequest ) { // vérifie le support
    // si supporté
    xhr = new XMLHttpRequest();
} else if ( window.ActiveXObject ) { // test l'IE6
    // utilise les méthodes anciennes
    xhr = new ActiveXObject("Msxml2.XMLHTTP");
}
```

Le plus simple encore est de transformer ce test en une fonction qui renvoie l'objet créer que vous pourrez utiliser plus rapidement.

```
function getHTTPObject() {
    // initialise la variable
    var xhr;
    if ( window.XMLHttpRequest ) { // vérifie le support
        // si supporté
        xhr = new XMLHttpRequest();
    } else if ( window.ActiveXObject ) { // test l'IE6
        // utilise les méthodes anciennes
        xhr = new ActiveXObject("Msxml2.XMLHTTP");
    }
    // renvoi la variable pour être utilisée
    return xhr;
}
```

## 4. Créer un appel Ajax

Créer une instance de l'objet Ajax est séparer de la création de l'appel. Dans notre application, il y a quelques transformations à réaliser pour utiliser de l'Ajax.

Le plus gros changement sera la source des données, précédemment elle était stockée dans un objet JSON à l'intérieur du code JavaScript, nous allons sortir cet objet et créer un fichier texte nommé contacts.json . Cette extension n'est pas obligatoire, mais cela est une convention et l'extension veux bien dire ce qui il y dedans.

Il suffit d'once de faire un copier collé dans le nouveau fichier et de ne pas oublier de supprimer l'instruction var.

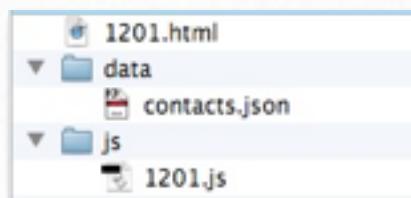
```
{
  "address" : [
    {
      "nom": "hillisha",
      "email": "hill@example.com",
    },
    {
      "nom": "Paul",
      "email": "cleveland@example.com",
    },
    {
      "nom": "vishaal",
      "email": "vish@example.com",
    }
  ]
}
```

```

},
{
  "nom": "mike",
  "email": "grady@example.com",
},
{
  "nom": "jamie",
  "email": "dusted@example.com",
}
]
}

```

Ce fichier sera enregistré dans un dossier data. Voici la structure des dossiers pour cet exemple :



#### 4.1. Envoyer la requête

L'Ajax dialogue avec le serveur, le serveur ne parle pas à l'Ajax. À cause de cela, l'Ajax fait deux choses :

- L'envoi une requête au serveur
- Le traitement des données renvoyé du serveur.

La première chose à réaliser après l'instanciation de l'objet est de lui envoyer une requête. L'envoi d'une requête Ajax ouvre de nouvelles propriétés à consulter comme readyState que nous allons voir dans un instant.

#### 4.2. open()

La méthode open() est la seconde étape de démarrage de votre appel Ajax. Elle fait penser à une configuration de l'appel Ajax. Elle ne fait que préparer la suite des opérations.

```

var request = getHTTPObject();

/* Préparation de l'appel */
request.open("GET", "data/contacts.json", true);

```

Il y a trois paramètres à la méthode :

- La méthode
- Le fichier ou l'URL
- Un booléen pour définir la méthode asynchrone.

Il existe aussi deux paramètres optionnels :

- L'utilisateur
- Mots de passe

#### I. méthode

Le premier paramètre dans la méthode `open()` est la méthode que vous voulez utiliser pour l'appel Ajax, soit GET ou POST (je ne vous explique plus la différence !)

## II. fichier ou URL

Le paramètre fichier ou URL est l'endroit où on note le nom du fichier ou l'URL HTTP complète de la source de données. Si le fichier est local comme dans notre cas, le chemin est relatif à votre document HTML qui contient l'appel Ajax.

## III. Le booléen

Ce booléen stipule de type de communication, `True` pour asynchrone qui est conseillé et `False` pour synchrone.

## IV. Les optionnels

Les deux derniers paramètres sont réservés pour le nom de l'utilisateur et le mot de passe. Vous utiliserez ces paramètres pour une connexion en Ajax (par exemple). Attention si vous utiliser ces paramètres de bien protéger par cryptage le mot de passe.

### 4.2. `send()`

Après la configuration avec la méthode `open()`, vous allez utiliser la méthode `send()` pour envoyer les données et la requête et attendre avec la propriété `readyState` si ont peu continué.

```
var request = getHTTPObject();
/* Préparation de l'appel */
request.open("GET", "data/contacts.json", true);

/* Initialise une requête */
request.send(null);

/* OU - initialise l'appel avec des données */
request.send("coucou les données");
```

Vous avez remarqué que nous pouvons passer `null` dans l'appel, ce qui veut dire que nous n'envoyons aucune donnée avec l'appel Ajax, nous ne voulons que le fichier.

Si vous utilisez un processus sur le serveur (PHP), vous pouvez passer un système de filtrage par la méthode `send()`.

```
var request = getHTTPObject();
/* Préparation de l'appel */
request.open("GET", "data/contacts.json", true);
/* initialise la requête */
request.send("recherche=hill");
```

ce qui ferait une URL du type : `recherche.php?recherche=hill`.

## 5. Réception des données

Après l'envoi de la requête, une information sera envoyée sur serveur, en espérant avec les données demandées. Comme je l'ai expliqué il existe une propriété `readyState`, qui est attachée à un évènement appelé `readystatechange`, qui continuellement scrutent tous les appels Ajax et renvoi si les données sont utilisables.

### 5.1. `readystate`

Cette propriété d'un appel Ajax qui vous indique le statut de l'évolution de cet appel :

- 0 - La méthode open n'a pas été appelée (pas initialisé)
- 1 - La méthode open a été appelée , mais pas la méthode send (chargement)
- 2 - La méthode open a été appelée et la requête est en court d'envoi (chargé)
- 3 - La réponse est en cours d'envoi (communication)
- 4 - La requête est terminée (fin)

Chaque fois que l'état du `readystate` change, l'évènement `readystatechange` est activé. Connaissant cela, vous pouvez lier cet évènement et attendre l'état «4» avant de traiter les données reçues.

On pourrait gérer tour les états mais cela n'est pas nécessaire pour l'utilisateur.

```
var request = getHTTPObject();

request.onreadystatechange = function() {
    // test si le statut = 4
    if ( request.readyState === 4 ) {
        // le traitement a réalisé
    }
}
```

### 5.2. Statut du serveur

Le `readyState` est très pratique, car il donne l'évolution du processus de l'appel Ajax. Il ne donne aucune information si la requête à réussi.

L'objet Ajax renvoie également une propriété appelée `status`, qui renvoie une valeur de retour du serveur comme 404, 200, 304, 500 etc. Il permet de savoir s'il y a un problème sur le serveur, la valeur courante est :

- 400 - Ne trouve pas la page
- 304 - pas modifié
- 500 - erreur interne du serveur
- 200 - touty ok sur le serveur

On pourrait gérer toutes ces conditions, mais dans notre cas, nous allons uniquement vérifier la valeur 200 et le combiné avec le `readyState` dont la valeur est 4

```
var request = getHTTPObject();

request.onreadystatechange = function() {
    // test si le statut = 4 et le serveur est ok
    if ( request.readyState === 4 && request.status === 200) {
```

```

    // le traitement a réalisé
}
}

```

## 6. La réponse du serveur

A l'intérieur de l'évènement `onreadystatechange`, et après avoir vérifié si la requête est terminée, vous pouvez finalement récupérer les données récupérer après l'appel Ajax.

En plus du `readyState` et `status`, l'objet Ajax retourne aussi les données comme une propriété. Il l'a renvoyé comme une chaîne de caractère ou du XML, cela dépend du choix que vous avez défini.

### 6.1. Comme une chaîne

Si la réponse est en chaîne, il sera retourné comme `responseText`. C'est la méthode classique de retour des données, Il pourra être traité facilement avec les méthodes JavaScript. Il peut contenir du texte simple, du HTML du JSON (hum !!).

### 6.2. Comme du XML

Si vous voulez récupérer en XML, il sera retourné comme `responseXML`.

```

var request = getHTTPObject();

request.onreadystatechange = function() {
    // test si le statut = 4 et le serveur est ok
    if ( request.readyState === 4 && request.status === 200) {
        // affiche les données récupérer
        console.log(request.responseText);
    }
}

/* Préparation de l'appel */
request.open("GET", "data/contacts.json", true);
/* Initialise une requête */
request.send(null);

```

## 7. Tout dans une fonction

Maintenant que nous avons un appel Ajax qui fonctionne, vous voulez l'utiliser plus d'une fois, comme dans notre exemple, pour afficher la liste ou au remplissage du champ de recherche. Pour que la fonction soit le plus compatible, nous allons envoyer la source/url en paramètre à cette fonction. Voici la fonction :

```

/* définition de la fonction d'appel Ajax */
function ajaxCall(dataUrl) {
    var request = getHTTPObject();
    request.onreadystatechange = function() {
        // test si le statut = 4 et le serveur est ok
        if ( request.readyState === 4 && request.status === 200) {
            // affiche les données récupérer
            console.log(request.responseText);
        } // fin test Statut
    } // fin onreadystatechange
    /* Préparation de l'appel */

```

```

request.open("GET", "dataUrl", true);
/* Initialise une requête */
request.send(null);
}

```

## 8. Récupérer les données

Maintenant les données sont dans le gestionnaire `onreadystatechange`. Cela n'est ce dont nous avons besoin!

Si je veux pouvoir utiliser cette fonction et gérer les données récupérer de différente façon, il suffit d'envoyer le nom de la fonction à utiliser pour le traitement des données. Cette fonction est appelée `callback` en anglais.

```

function ajaxCall(dataUrl,callback) {
    var request = getHTTPObject();
    request.onreadystatechange = function() {
        // test si le statut = 4 et le serveur est ok
        if ( request.readyState === 4 && request.status === 200) {
            // sauvegarde les données dans une variable
            var contacts = JSON.parse(request.responseText);
            // vérifie si callback est bien une fonction
            if (typeof callback === "function") {
                callback(contacts);
            } fin test fonction
        } // fin test Statut
    } // fin onreadystatechange
    /* Préparation de l'appel */
    request.open("GET", "dataUrl", true);
    /* Initialise une requête */
    request.send(null);
}

```

Vous avez remarquer une nouvelle méthode «`typeof`», celle ci permet vérifier le type d'objet que l'on traite. Le retour est soit `string`, `number`, `Boolean`, `undefined` ou comme dans notre cas `function`.

Vous avez (j'espère) encore remarquer une nouvelle méthode : `JSON.parse`. Comme je l'avais déjà dit précédemment, le retour `responseText` est toujours une chaîne de caractère , la méthode `JSON.parse` va convertir ce texte en un objet JSON réutilisable.

Voici l'appel à la fonction `ajaxCall` et une fonction `callback` anonyme:

```

ajaxCall("data/contacts.json", function(data){
    /* Voici le contenu de la fonction qui sera appelée
    le paramètre data est la liste des contacts au format JSON
    Il suffit (oups) de boucler dedans */
});

```

## 9. Appel Ajax à répétition

Créer une illusion de donnée en temps réel est une chose que le JavaScript et en particulier l'Ajax est fait pour cela. Nous avons déjà appris à faire un appel Ajax (heu ! j'espère) qui change le contenu de la page au retour d'une action de l'utilisateur. Mais nous pouvons également faire un système qui va automatiquement faire des appels Ajax.

La méthode setInterval() permet justement d'exécuter un block de code à un intervalle définit, elle reçoit deux paramètres : le premier est la fonction à exécuté à chaque fois et la seconde est l'intervalle en millisecondes.

```
/* affiche une boîte d'alert toutes les 5 secondes */
setInterval('alert("Appel ajax", 5000);
```

La même technique peut être utilisée avec un appel Ajax

```
/* faire un appel Ajax toutes les 5 secondes*/
setInterval('ajaxCall("data/contacts.json", function(){
    console.log("fait un appel");
})', 5000); // 5000 millisecondes = 5 seconds (pour les non matheux hihi)
```

Attention d'utiliser ceci avec parcimonie et avec un intervalle pas trop court pour ne pas saturer le serveur.

Voici le code js final :

```
function getHttpObject() {
    var xhr;
    if (window.XMLHttpRequest) { // vérifie si compatible
        xhr = new XMLHttpRequest();
    } else {
        xhr = new ActiveXObject('Microsoft.XMLHTTP');
    }
    return xhr;
}

function ajaxCall(dataUrl, target, callback) {
    var request = getHttpObject();
    // préparer l'appel
    request.open('GET', dataUrl, true);
    // initialisé la requête
    request.send(null);
    // gestion du retour des données
    request.onreadystatechange = function() {
        if (request.readyState === 4 && request.status === 200) {
            // traitement le contenu json
            var contacts = JSON.parse(request.responseText);
            // utiliser la fonction de traitement
            if(typeof callback === 'function') {
                callback(contacts);
            }
        }
    }
}

// déclaration d'un objet qui contient toutes les méthodes

var adr = {
    toutAfficher : function() {
        ajaxCall('data/lecture.php', target, function(data) {
            var nb = data.length;
            var i;
            target.innerHTML = '';
            for (i = 0; i < nb; i++) {
                target.innerHTML += data[i];
            }
        });
    }
}
```

```

    if (nb > 0 ) {
        for(i = 0; i < nb; i++) {
            var buffer = data[i];
            target.innerHTML += buffer.nom + ' '+ buffer.titre +'<br>';
        }
    });
},
// fin de la fonction toutAffiche

ajouteActive : function() {
    this.parentNode.setAttribute('class','active');
},
supprimeActive : function() {
    this.parentNode.removeAttribute('class','active');
},
survol : function() {
    formulaire.setAttribute('class','auDessus');
},
quitte : function() {
    formulaire.removeAttribute('class','auDessus');
},

// methode de recherche
recherche : function() {
    ajaxCall('data/lecture.php',target, function(data) {
        var nb = data.length;
        // sauvegarde de la valeur à rechercher
        var valeurRecherche = champRecherche.value.toUpperCase(),
        i;
        // suppression de la gestion par défaut
        event.preventDefault();
        target.innerHTML = '';
        if (nb > 0 && valeurRecherche !== '') {
            for(i = 0; i < nb; i++) {
                var buffer = data[i];
                var trouver = buffer.nom.toUpperCase().indexOf(valeurRecherche);
                if (trouver != -1 ) {
                    target.innerHTML += buffer.nom+' - '+buffer.titre+'<br>';
                } // fin du if
            } // fin de la boucle
        } // fin du test
    });
    // fin fonction callBack
}
// fin methode recherche

} // fin de l'objet adr

// sauvegarde des noeuds
var btn          = document.getElementById('Tous');
var champRecherche = document.getElementById('q');
var formulaire   = document.getElementById('formRecherche');
var target       = document.getElementById('Cible');

// ajout des listener

```

```

btn.addEventListener('click',adr.toutAfficher, false);
champRecherche.addEventListener('focus',adr.ajouteActive, false);
champRecherche.addEventListener('blur',adr.supprimeActive, false);
formulaire.addEventListener('mouseover',adr.survol, false);
formulaire.addEventListener('mouseout',adr.quitte, false);
formulaire.addEventListener('keyup',adr.recherche, false);

removeActiveSection : function() {
    this.parentNode.removeAttribute("class");
},
addHoverClass : function() {
    formulaire.setAttribute("class", "auDessus");
},
removeHoverClass : function(){
    formulaire.removeAttribute("class");
}
} // fin object addr
/* Activation des gestions d'évènement */
champRech.addEventListener("keyup", addr.recherche, false);
champRech.addEventListener("focus", addr.setActiveSection, false);
champRech.addEventListener("blur", addr.removeActiveSection, false);
boutonTous.addEventListener("click", addr.afficheTous, false);
formulaire.addEventListener("mouseover", addr.addHoverClass, false);
formulaire.addEventListener("mouseout", addr.removeHoverClass, false);
formulaire.addEventListener("submit", addr.recherche, false);
})(); // Fine fonction anonyme
// fin du fichier js/1201.js

```

# 13

## Travailler avec des librairies



Dans ce chapitre, je vais vous rendre la vie de programmeur plus facile. J'ai essayé de prendre le meilleur du JavaScript mais vous avez remarqué que cela n'est pas toujours piqué des vers. Comme le Javascript et le Web a encore beaucoup d'évolution en vue, il n'est pas toujours nécessaire de tout devoir créer. Nous allons voir une introduction à une série de librairie qui vont vous aidée et qui sont souvent très bien documenté.

Mais n'oublie pas ... elle sont toutes entièrement écrite en JavaScript!

Etudier une librairie est similaire à un l'apprentissage d'un nouveau langage, en fait, vous allez apprendre de nouveaux mots (méthode) et ces paramètres.

Les librairies sont souvent spécialisé dans une partie spécifique de la programmation sous JavaScript pour la gestion des mobiles, les méthodes de sélection plus rapide, l'animations etc..

## 1. Jquery

Jquery est sûrement la librairie JavaScript qui est utilisé par plus de 50% des meilleurs sites sur le web. Elle a été créé par John Resig et sortie en 2006. Elle est actuellement activement mise à jour par un petit groupe de développeur. Elle est open source et gratuite.

Jquery est une librairie dans le sens vrai du terme, c'est une collection de fonctions et méthodes prédefinie chargé dans votre document à travers d'un script externe. Jquery est une librairie très large et a de nombreux avantages :

- S'occupe de rendre les fonctions compatible avec les anciens navigateur.
- La syntaxe est très simple
- Est faite de raccourci pour vous simplifier la vie
- Permet des animations très simple
- Est très populaire et mise à jour couramment
- Permet de travailler avec des système portable (tablette & smartphone)
- Simplifie grandement l'utilisation de la technologie Ajax
- Etc.

Je vous conseil de télécharger la version production qui est minimalisé et compressé. Je ne pense pas que vous modifierai le code qui se trouve dans cette librairie !

Téléchargement à l'adresse : [jquery.com/download/](http://jquery.com/download/)

Pour plus de clarté j'ai changer le nom en jquery.js. Ce fichier sera placer dans le dossier js, et sera appelé avec tout code js personnel

```
<!doctype html>
<html lang="fr">
<head>
  <title>Chapter 13</title>
  <meta charset="utf-8">
</head>
<body>
  <script src="js/jquery.js"></script>
  <script src="js/messScript.js"></script>
</body>
</html>
```

## 2. Exemple d'utilisation de jQuery

### Fichier html

```
<!doctype html>
```

```

<html lang="fr">
<head>
    <title>Exemple utilisation Jquery</title>
    <meta charset="utf-8">
    <link href="css/demo.css" rel="stylesheet" media="all" type="text/css">
</head>
<body>
    <button type="button">Ne clique pas sur le bouton !</button>
    <div id="container">
        <div class="module">
            <p>Bonjour module un!</p>
        </div>
        <div class="module">
            <p>Bonjour module deux!</p>
        </div>
        <div class="module">
            <p>Bonjour module trois!</p>
        </div>
    </div>

    <script src="js/jquery.js"></script>
    <script src="js/script.js"></script>
</body>
</html>
<!-- Fin fichier demo.html --&gt;
</pre>

```

## La feuille css

```

#container {
    overflow: hidden;
}

.module {
    width: 33%;
    float: left;
    background: #eee;
}

```

Au coeur de la librairie jQuery, et ce qui fait sa renommée, est la méthode `ready()`. La méthode `ready()` est lié au document et ne prend qu'un seul paramètre, qui est le code complet du JavaScript que est attaché au document HTML. C'est une méthode du type `listener` (vu précédemment) qui ne sera exécuté que lorsque le DOM sera entièrement construit. Précédemment nous avons utilisé la méthode `load()` pour exécuter un script au chargement de la page. L'évènement `load` ne démarrera que lorsque tout est chargé, même les images. Il attendra parfois longtemps pour certaines images. La méthode jQuery `ready` par contre diffère de la méthode native `load`, il attend uniquement que le DOM soit construit pour démarrer, ce qui est beaucoup plus fluide.

```

$(document).ready(function () {
    // votre code ici
}); // fin document.ready

```

Vous avez sûrement remarqué que nous n'utilisons pas une méthode normale pour attacher la méthode au document, nous aurions dû utilisé une `document.getElementById()`.

En lieu et place nous avons le signe \$ et des parenthèses qui entourent notre élément du dom. Bienvenu dans le monde de la librairie jQuery.

Ceci s'appelle le jQuery selector, il vous permettra de sélectionner n'importe quel élément de votre DOM comme vous allez le voir de suite.

## Sélecteur

La barrière infranchissable qu'ont les designer avec le JavaScript a été détruite avec la librairie jQuery. jQuery la résout avec son moteur de sélection, qui permet d'utiliser la syntaxe CSS pour sélectionner les éléments du DOM.

Comme dit précédemment la syntaxe jQuery est centré sur le signe \$( ) . Le signe dollar \$ signal à jQuery de mettre son moteur de sélection en route (vroum vroum ) Voici trois exemples de sélection d'un Id dans une page

Méthode native du JavaScript :

```
var container = document.getElementById("container");
```

Méthode native du JavaScript - nouvelle version

```
var container = document.querySelector("#container");
```

Méthode jQuery

```
var container = $("#container");
```

Exemple avec une classe

```
// native JavaScript version  
var module = document.getElementsByClassName("module");  
// native JavaScript - nouvelle version  
var module = document.querySelectorAll(".module");  
// native JavaScript - nouvelle version, retourne un tableau  
var module = document.querySelectorAll(".module");  
// jQuery version  
var module = $(".module");
```

## Se déplacer dans le DOM

En partant de la page HTML de la page 86 nous allons voyager dans l'arborescence du DOM comme nous l'avons vu dans le chapitre 7, mais avec des raccourcis de jQuery.

Voici les méthodes pour naviguer :

- parent() - sélectionne le nœud parent immédiat
- parents()- voyage dans tous les nœuds parent jusqu'à ce qu'il corresponde au sélecteur
- find()- navigue vers le bas jusqu'à ce qu'il trouve.
- sibling() - récupère les frères qui correspondent au sélecteur

```
// remonte d'un niveau  
$(".module").parent().addClass('module-parent');
```

```
// remonte de plusieurs niveau
$("p").parents("#container").addClass('p-parents');

// descent
$("#container").find(".module").addClass('container-find');

// voyage au même niveau
$(".module").siblings(".module").addClass('module-siblings');
```

Vous avez sûrement remarqué la méthode `addClass()`. Je pense que vous pouvez comprendre son utilisation ?

Voici d'autre méthode de navigation fournie par jQuery :

```
// sélectionne le premier module
$(".module").first().addClass('first-module');

// sélectionne le suivant
$(".module").first().next().addClass('second-module');

// sélectionne le dernier
$(".module").last().addClass('last-module');

// sélectionne le précédent
$(".module").last().prev().addClass('second-to-last-module');
```

## Ajouter du style

Même si il n'est pas conseillé de changer la CSS d'un élément en JavaScript, la méthode de l'ajout d'une class sera plus mieux ..

Voici quand même comment faire :

```
// Change la hauteur de tous les nœuds qui ont une classe module
$('.module').css({
    'height': '300px',
    'color': 'red'
});
```

## Gestion d'évènement

L'attachement d'un évènement se réalise presque de la même façon qu'en JavaScript. Il existe des méthodes en jQuery qui font exactement la même chose qu'en JavaScript, comme `click`, `mouseover`, `mouseout`, `submit`, `change`, `blur`, `focus`, `keyup`, `keydown` et tout autre évènement style tactile.

Comme vu précédemment chaque événement en Jquery peut être passé avec une fonction callback, qui contientrait le traitement que vous voulez réaliser

Exemple de gestion du click sur un bouton

```
// lie un click sur le seul bouton dans notre page
$('button').click( function () {
    // fonction anonyme (callback qui va afficher un message
    console.log('Tu m'a eu!');
});
```

## Animation

La capacité de combiner des événements et des méthodes `css()` pour créer des animations JavaScript est une des actions les plus populaires de jQuery. Pour cela une méthode `animate()` est dans la bibliothèque.

La méthode `animate()` reçoit trois paramètres :

- La première est la propriétés CSS que vous voulez animer
- Le second est le durée de l'animation en miliseconde
- Le troisième (optionnel ) est une fonction callback qui sera exécuté quand l'animation sera terminée.

```
// lie un click sur le seul bouton dans notre page
$('button').click(function () {
    // anime la hauteur à 0 de tout les modules
    $('.module').animate({
        'height': '0px' }, 500, function () {
        // après l'animation change le texte du bouton
        $('button').text('Et maintenant hein!');
    });
    // fermeture fonction si terminé
}); // ferme la méthode du click
```

## Ajax avec jQuery

Souvenez vous toutes les boucles et testes que nous avons réalisé dans notre fichier contacts avec l'ajax pour la gestion de l'auto remplissage de la liste. Vous allez être surpris de voir comme il est facile de réalisé des requêtes Ajax avec jQuery.

Voici la méthode qui fonctionne pour notre gestion de contacts

```
$.ajax({
    type: 'GET',           // méthode d'envoi GET ou POST
    url: 'data/contacts.json', // Url des données
    dataType: 'json',       // type: xml, json, script, ou html
    success: function(data) {
        // Fait ceci si réussit
        console.log(data.addressBook);
    },
    error: function () {
        // en cas d'erreur fait cela ...
        alert('il y a une erreur ajax');
    }
});
```

Encore plus cool ! Il existe une méthode jQuery qui traite directement des appels Ajax avec un retour en JSON

La méthode getJSON() , qui reçois deux paramètres; le premier est le chemin des données JSON (ou programme qui renvois des donnée en JSON, style PHP), et le deuxième est comme dab, une fonction caalback qui va traité ces données.

```
$.getJSON('data/contacts.json', function (data) {
    var addrBook = data.addressBook,
        count = addrBook.length;
    // Efface le contenu de la cible
    $('#Cible').empty();
    // vérifie le nombre de ligne
    if (count > 0) {
        console.log(addrBook);
    } // fin test nombre
});
```

## Etendre jQuery avec d'autre librairie

Le meilleur exemple dans notre cas est la librairie jQuery-ui qui est spécialisé sur l'interface.

Vous n'allez pas le croire mais l'exercice précédent peut se réalisé avec une seul ligne de JavaScript ... oui oui

Voici le code html :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <meta charset="utf-8" />
        <title>test autocomplete</title>
        <script src="js/jquery.js" type="text/javascript"></script>
        <script src="js/jquery-ui.js" type="text/javascript"></script>
        <link href="css/site.css" type="text/css" rel="stylesheet" />
        <link href="css/jquery-ui.css" type="text/css" rel="stylesheet" />
    </head>
    <body>
        <form method="post">
            <label for="nom">nom du membre : </label>
            <input type="text" name="nom" id="nom" />
        </form>
        <script type="text/javascript">
            $(function(){
                $('#nom').autocomplete({source:'data/membre.php'});
            });
        </script>
    </body>
</html>
```

Heu pas besoin de JS, la ligne dans l'HTML suffit !!

Voici le code php

```
<?php
$info = "mysql:host=localhost;dbname=5xcli";
$user = "root";
$pass = "root";
$utf  = array(PDO::MYSQL_ATTR_INIT_COMMAND => "SET NAMES utf8");
try {
    $pdo = new PDO($info, $user, $pass, $utf); }
catch (Exception $e) {
    echo 'Une erreur de type : '.$e;
    die();
}
$tempo = $pdo->query("SELECT nom AS label FROM membres WHERE nom LIKE '%".$_REQUEST['term']."%' ");
$tempo->execute();
$buffer = $tempo->fetchAll(PDO::FETCH_ASSOC);
echo json_encode($buffer);
?>
```

Super ... voyons voir ce que nous pouvons encore faire avec jQuery et ces différents ajout.

## 3. Jquery UI

### 3.1. Le drag & drop

La librairie jQuery UI, fourni un système de Drag & Drop composé de deux partie, la méthode daggable et droppable.

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Document</title>
    <script src="js/jquery.js" type="text/javascript"></script>
    <script src="js/jquery-ui.js" type="text/javascript"></script>
    <link href="css/dragetdrop.css" rel="stylesheet">
</head>
<body>
<div class="bloc1"></div>
<div class="bloc2"></div>
<script>
$(function() {
    $('.bloc1').draggable( {
        axis : 'x'
    });
    $('.bloc2').draggable( {
        axis : 'y'
    });
});
</script>
</body>
</html>
```

La page css

```
.bloc1 {
    height: 50px;
    width: 50px;
    background-color: #00ffcc;
}

.bloc2 {
    height: 50px;
    width: 50px;
    background-color: #fc0101;
}
```

Vous pouvez remarquer que le bloc vert (le bloc1) se déplace sur l'axe des X et que le second rouge (bloc2) se déplace sur l'axe des Y. Il existe un autre paramètre qui est le nom du «container» qui sera la limite de déplacement.

Ajouter ceci dans votre page html

```
<div id="zone">
    <div class="bloc3"></div>
</div>
```

ceci dans la css

```

.bloc3 {
    height: 50px;
    width: 50px;
    background-color: #53ffff;
}

#zone {
    height: 150px;
    width: 350px;
    position: absolute;
    top:100px;
    left: 200px;
    border: 1px solid #000;
}

```

et la gestion du déplacement en dans la partie script

```

$('.bloc3').draggable( {
    containment : '#zone'
});

```

Super ...

### 3.1.I. Magnétisme

Il est possible de magnétiser vos éléments afin qu'il se colle automatiquement au autre :

```

$('.bloc2').draggable( {
    snap : true,
    axis : 'y'
});

```

il est également possible de limité le magnétisme à certain élément :

```

snap : '.toto' // sera magnétique avec les éléments qui ont une classe toto

```

Il reste un paramètre : grid qui va définir une grille de déplacement , ou un pas X ou Y.

### 3.1.II. Retour automatique

Le paramètre revert si il est mis à True fera revenir a sa position initial au lâcher de la souris .

```

$('.bloc3').draggable( {
    containment : '#zone',
    revert : true
});

```

Le système drag & drop se réalise en deux étape :

- Le glisser, réalisé avec le draggable, et que nous venons de voir ;
- Le déposer, que nous allons apprendre maintenant

Une nouvelle page html

```

<!doctype html>
<html lang="fr">
<head>
<meta charset="UTF-8">

```

```

<title>Document</title>
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.js" type="text/javascript"></script>
<link href="css/dragetdrop2.css" rel="stylesheet">

</head>
<body>
<div class="bloc1"></div>
<div class="bloc2"></div>
<div id="zone">
    déposé ici
</div>
<script>
$(function() {
    $('.bloc1').draggable();
    $('.bloc2').draggable();
    $('#zone').droppable( {
        drop : function() {
            alert('Action réussie');
        }
    });
});
</script>
</body>
</html>

```

Cela fonctionne très bien ... Il est aussi possible de limiter les éléments qui sont accepté dans la zone de dépôt :

```

$('#zone').droppable( {
    accept : '.bloc1',
    drop : function() {
        alert('Action réussie');
    }
});

```

### I. Retour a la maison !

Non non! Le cours n'est pas finit hihi; nous avons vu précédemment comment renvoyer un élément a sa place de départ avec le paramètre `revert`, voici la technique pour remettre a sa place tout élément qui n'est pas déposé dans la zone de dépôt. Nous allons utiliser deux mots clés qui sont utilisable avec le paramètre `revert` : `valid` et `invalid`. Le premier permet à l'élément d'être renvoyé a sa place si il est déposé dans la bone zone, le second fait le contraire.

```

$('.bloc1').draggable({
    revert : 'valid'
});
$('.bloc2').draggable({
    revert : 'invalid'
});

```

### II. Sélection et re-dimensionnement

```

<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">

```

```

<title>Document</title>
<script src="js/jquery.js" type="text/javascript"></script>
<script src="js/jquery-ui.js" type="text/javascript"></script>

<link href="css/jquery-ui.css" rel="stylesheet">
<link href="css/resize.css" rel="stylesheet">

</head>
<body>
<div id="bloc1"></div>

<script>
$(function() {
    $('#bloc1').resizable();
});
</script>
</body>
</html>

```

et notre css

```

#bloc1 {
    height: 50px;
    width: 50px;
    background-color: #00ffcc4;
}

```

Super ... la mise à dimension se fait sur les 2 axes séparément ou simultanément. Il existe quelques paramètres intéressants comme `aspectRatio` qui peut avoir ces valeurs :

- `true` qui permet de modifier la taille proportionnellement
- `16 / 9` respecte le ratio largeur/hauteur

```

$('#bloc1').resizable({
    aspectRatio : true
});

```

D'autres paramètres intéressants :

- `maxWidth` définit la largeur maximale
- `minWidth` définit la hauteur minimale
- `maxHeight` définit la hauteur maximum
- `minHeight` définit la hauteur minimale

Nous avons vu dans le drag & drop la possibilité de limiter le déplacement à une zone, ceci est aussi possible pour la taille avec la même technique : `containment`.

```

$('#code1').resizable({
    containment : '#zonedéfine'
});

```

### 3.1.V Utilisation d'un helpers

Les helpers permettent d'afficher l'élément d'origine et de voir la nouvelle zone en redimensionnant transparente

Il vous faut ajouter une règle css :

```
.help{  
    border : 1px solid #1877D5;  
    background : #84BEFD;  
    opacity : 0.3;  
}
```

et ajouter dans votre script le paramètre :

```
$('#bloc1').resizable({  
    helper : 'help',  
    ..  
    animate : true,
```

Il est aussi possible d'animer ces changements, mais déconseillé car peu devenir vite très moches...

## 4. Sélection d'élément

La librairie jQuery fournit une méthode Selectable qui permet à l'utilisateur de sélectionner un ou des éléments pour les exploiter (récupération de valeurs, ajouts multiples dans un panier, gestionnaire d'images, etc.)

Nous devons créer une liste <li></li> pour cette méthode, seule façon de l'utiliser.

```
<!doctype html>  
<html lang="fr">  
<head>  
    <meta charset="UTF-8">  
    <title>Selection</title>  
    <script src="js/jquery.js" type="text/javascript"></script>  
    <script src="js/jquery-ui.js" type="text/javascript"></script>  
  
    <link href="css/jquery-ui.css" rel="stylesheet">  
    <link href="css/selectable.css" rel="stylesheet">  
</head>  
<body>  
    <ul id="selection">  
        <li>1er élément</li>  
        <li>2ème élément</li>  
        <li>3ème élément</li>  
    </ul>  
    <script>  
        $('#selection').selectable();  
    </script>  
</body>  
</html>
```

jQuery ui utilise deux règles css pour changer les éléments sélectionné et non sélectionné

ajouté ceci dans votre nouvelle page selectable.css

```
.ui-selecting{  
    background:white;  
}  
.ui-selected{
```

```
background:black;
color:#fff;
}
```

Sélectionner des éléments dans la page est bien, mais exploiter les données, c'est plus mieux!

Ajouter cette ligne a votre html

```
<div id="resultat"></div>
```

Ajoutons l'évènement stop qui sert a exécuter le code qui suit quand la sélection est terminé.

```
$('#selection').selectable({
    stop : function(){
        $('#resultat').empty();
        $('.ui-selected').each(function(){
            $('#resultat').append( $(this).text() + ' ; ' );
        })
    }
});
```

La méthode empty() que nous avons déjà utilisé va vider la zone définie par l'id resultat.

La sélection \$('.ui-selected') va nous retourné un tableau avec tout les éléments qui ont la classe ui-selected.

Le reste ... vous devriez comprendre ?

Il est possible aussi de réalisé une action dans le cas d'une dé-sélection !

```
$('#selection').selectable({
    unselected : function() {
        console.log('Vous venez de désélectionner un élément du DOM');
    }
});
```

## 5. Exercice

Création d'un petit système de panier,

- choix de 5 articles
- utilisation du drag & drop.
- Retrait du magasin si dans le panier.
- Possibilité de supprimer du panier (avec retour dans le magasin)

### Solution :

*Le code HTML*

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Mon premier panier</title>
    <script src="js/jquery.js" type="text/javascript"></script>
    <script src="js/jquery-ui.js" type="text/javascript"></script>
    <link href="css/jquery-ui.css" rel="stylesheet">
```

```

<link href="css/panier.css" rel="stylesheet">
</head>
<body>
<div id="panier">
<p>Votre panier</p>
<ul class="resultat"></ul>
</div>
<ul id="articles">
<li class="objet">Imac Tournesol</li>
<li class="objet">MacBook Pro</li>
<li class="objet">Ipad mini</li>
<li class="objet">MacPro</li>
<li class="objet">Apple Tv</li>
</ul>
<script src="js/panier.js"></script>
</body>
</html>

```

## La CSS

```

.objet {
    text-align: center;
    list-style-type: none;
    height: 40px;
    width: 150px;
    background-color: #7ce6a1;
    border: 1px solid;
    border-radius: 4px;
}

#panier {
    position: absolute;
    top: 10px;
    left: 500px;
    height: 150px;
    width: 300px;
    background-color: #e3dded;
}

```

## Le js

```

$(function() {
    // définition des objets déplaçable et changement du curseur
    $('.objet').draggable({
        revert : 'invalid',
        cursor : 'move'
    });

    // Gestion du panier
    $('#panier').droppable({
        accept : '.objet',
        drop : function(event, deplace){
            // traitement de l'objet déplacé
            var current = deplace.draggable; // sauvegarde de l'objet courant
            var resultat = $('.resultat'); // sauvegarde de la cible
        }
    });
}

```

```

        current.fadeOut(); // on fait disparaître notre élément
        resultat.append('<li class="obj-panier">' + current.html() + '</li>');
        // on ajoute dans le panier et on définit une classe à chaque élément
    }
});

// gestion du clic sur élément du panier
$('.resultat').selectable({
    stop : function(){
        $('.ui-selected', this).each(function(){
            // on traite chaque élément ayant été sélectionné
            $('#articles').append('<li class="objet">' + $(this).text() + '</li>');
            // on ajoute le contenu de l'article dans le magasin
            $(this).remove(); // on le supprime ensuite du panier

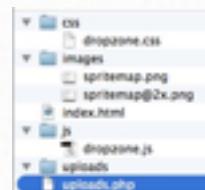
            $('.objet').draggable({ // et redéfinir le plugin !
                revert : 'invalid',
                cursor : 'move'
            });
        });
    }
});

```

## Upload de fichier avec dropzone.js

La librairie dropzone.js est indépendante de jQuery.js. elle en reprend les bases en interne, elle est fournie avec une css et deux images (en anglais).

Voici la structure de notre projet :



*Notre fichier html :*

```

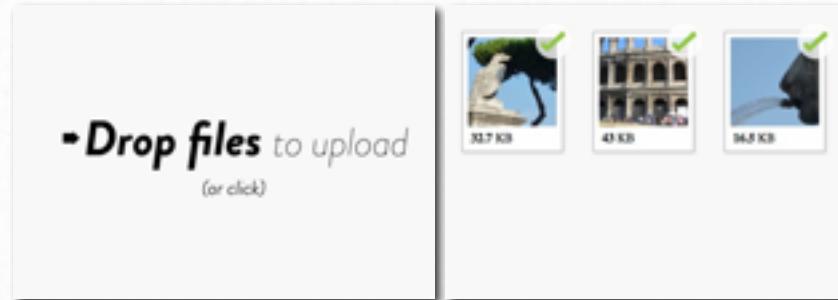
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Upload de fichier avec dropzone</title>
    <link rel="stylesheet" href="css/dropzone.css">
    <script src="js/dropzone.js"></script>
</head>
<body>
    <form action="uploads.php" class="dropzone"></form>
</body>
</html>

```

Le fichier php :

```
<?php  
$ds = DIRECTORY_SEPARATOR;  
$dossier = 'uploads';  
if (!empty($_FILES)) {  
    $fichierTemp = $_FILES['file']['tmp_name'];  
    $cheminCible = dirname( __FILE__ ).$ds.$dossier.$ds;  
    $fichierCible = $cheminCible.$_FILES['file']['name'];  
    move_uploaded_file($fichierTemp,$fichierCible);  
}  
?>
```

Cool plus rien à faire ... ( heu y'a pas de contrôle ni de mise à dimension)



Les fichiers glissés dans la zone grise sont automatiquement uploadé dans le dossier définit dans le code PHP.

Il existe d'autre librairie lié a jQuery :

- **jQuery-File-Upload** : <https://github.com/blueimp/jQuery-File-Upload>
- **uploadify** : <http://www.uploadify.com/demos/>
- **Pekabyte** : <https://github.com/pekebyte/pekeUpload>

## Graphique

Nous allons voir 2 librairies qui permette de réaliser des graphiques :

### 1. flot

**Url** : <http://www.flotcharts.org>

#### 1.1. Graphique basic

**html** :

```
<!doctype html>  
<html lang="fr">  
<head>  
    <meta charset="UTF-8">  
    <title>flot version 1</title>  
    <script src="js/jquery.js"></script>  
    <script src="js/jquery.flot.js"></script>  
    <link rel="stylesheet" href="css/exemple1.css">  
</head>  
<body>  
    <h1>Exemple 1</h1>
```

```

<div class="container">
    <div id="graphic1" class="graphic1"></div>
</div>
<script>
    var sin = [];
    for (var i = 0; i < 13; i += 0.5) {
        sin.push([i, 5+Math.sin(i)]);
    }
    var tab1 = [[0, 3], [4, 8], [8, 5], [9, 13]];
    var tab2 = [[0, 12], [7, 12], null, [8, 2.5], [13, 2.5]];
    $ .plot('#graphic1',[tab1, tab2, sin]);
</script>
</body>
</html>

```

## La css

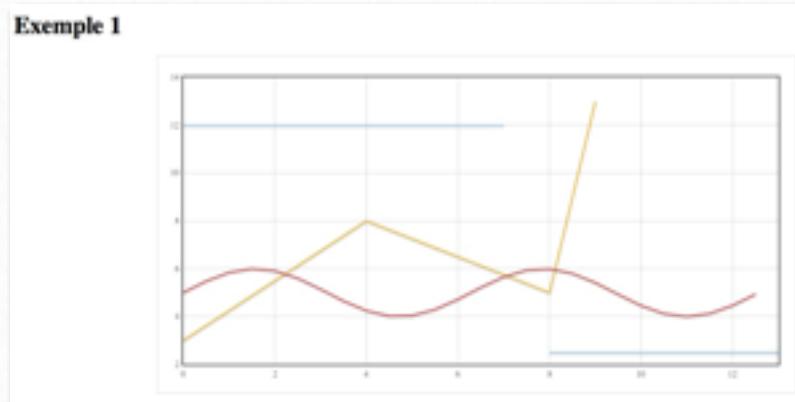
```

.container {
    box-sizing: border-box;
    width: 850px;
    height: 450px;
    padding: 20px 15px 15px 15px;
    margin: 15px auto 30px auto;
    border: 1px solid #ddd;
    background: #fff;
}

.graphic1 {
    width: 100%;
    height: 100%;
    font-size: 14px;
    line-height: 1.2em;
}

```

## Résultat :



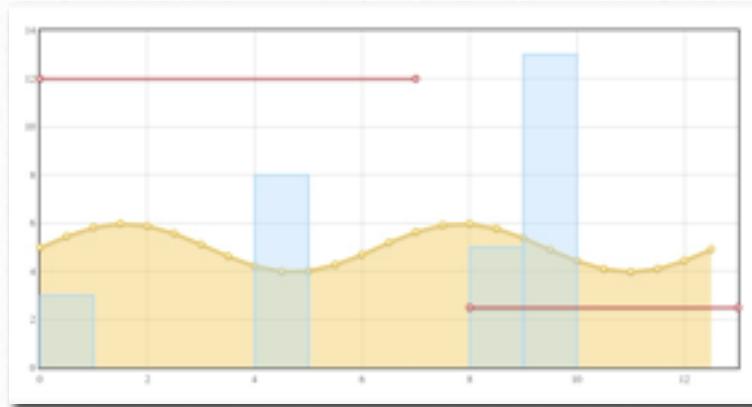
### 1.2. D'autre type de graphics et paramètre

```

$.plot('#graphic1',[{
    data : sin,
    lines : { show: true, fill: true},
    points: { show:true}
},{
    data : tab1,
    bars: { show: true}
}]

```

```
}, {
    data : tab2,
    lines: { show: true},
    points: { show:true}
});
```



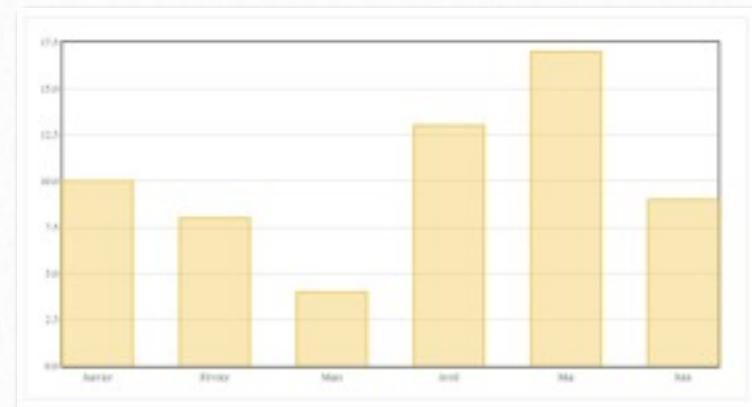
### 1.3. Avec du texte

Ajouter :

```
<script src="js/jquery.flot.categories.js"></script>
```

et voici un exemple :

```
$(function() {
    var data = [ ['Janvier',10], ['Février',8], ['Mars',4],
                ['Avril',13], ['Mai',17], ['Juin',9] ];
    $.plot('#graphic1',[data], {
        series: {
            bars : {
                show: true,
                barWidth: 0.6,
                align: 'center'
            }
        },
        xaxis: {
            mode: 'categories',
            tickLength: 0
        }
    });
});
```



### 1.4. Avec une légende

```
$(function() {
```

```

var graph1 = [];
var graph2 = [];
for (var i = 0; i < Math.PI * 2; i += 0.25) {
    graph1.push([i, Math.sin(i)]);
    graph2.push([i, Math.cos(i)]);
}

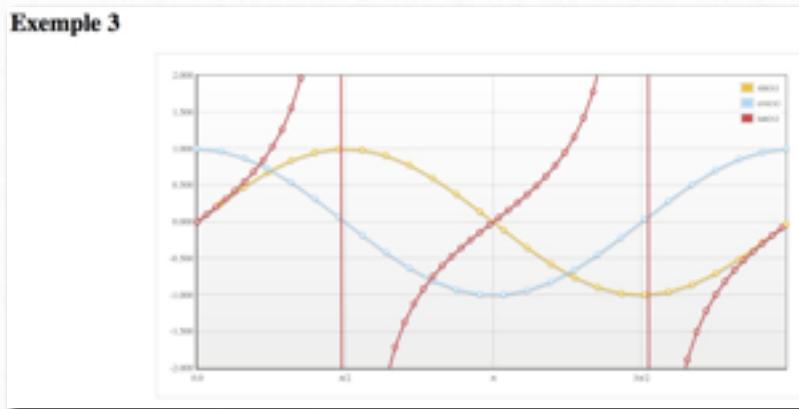
var graph3 = [];
for (var i = 0; i < Math.PI * 2; i += 0.1) {
    graph3.push([i, Math.tan(i)]);
}

$.plot("#graphic1", [
    { label: "sin(x)", data: graph1 },
    { label: "cos(x)", data: graph2 },
    { label: "tan(x)", data: graph3 }
], {
    series: {
        lines: { show: true },
        points: { show: true }
    },
    xaxis: {
        ticks: [
            0, [ Math.PI/2, "\u03c0/2" ],
            [ Math.PI, "\u03c0" ],
            [ Math.PI * 3/2, "3\u03c0/2" ],
            [ Math.PI * 2, "2\u03c0" ]
        ]
    },
    yaxis: {
        ticks: 10,
        min: -2,
        max: 2,
        tickDecimals: 3
    },
    grid: {
        backgroundColor: { colors: [ "#fff", "#eee" ] },
        borderWidth: {
            top: 1,
            right: 1,
            bottom: 2,
            left: 2
        }
    }
});
});

```

Résultat :

**Exemple 3**



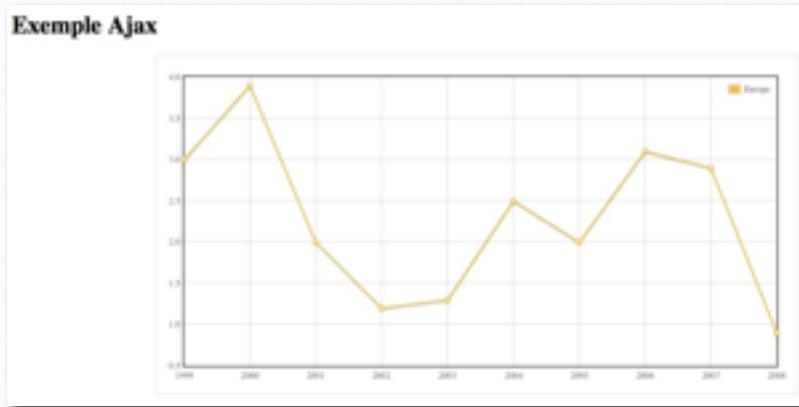
### 1.5. Avec récupération ajax et Json

Le fichier europe.json

```
{  
    "label": "Europe",  
    "data": [[1999, 3.0], [2000, 3.9], [2001, 2.0], [2002, 1.2], [2003, 1.3], [2004, 2.5], [2005, 2.0], [2006, 3.1], [2007, 2.9], [2008, 0.9]]  
}
```

Le javascript

```
$(function() {  
    var options = {  
        lines: {  
            show: true  
        },  
        points: {  
            show: true  
        },  
        xaxis: {  
            tickDecimals: 0,  
            tickSize: 1  
        }  
    };  
    var data = [];  
    $.plot("#graphic1", data, options);  
    var alreadyFetched = {};  
    function onDataReceived(series) {  
        if (!alreadyFetched[series.label]) {  
            alreadyFetched[series.label] = true;  
            data.push(series);  
        }  
        $.plot("#graphic1", data, options);  
    }  
    $.ajax({  
        url: "data/europe.json",  
        type: "GET",  
        dataType: "json",  
        success: onDataReceived  
    });  
});
```



### 1.6. Ajax avec timeOut

Trois fichier JSON -

donnee1.json :

```
{
  "label": "Europe",
  "data": [[1999, 3.0], [2000, 3.9], [2001, 2.0], [2002, 1.2], [2003, 1.3], [2004, 2.5], [2005, 2.0], [2006, 3.1], [2007, 2.9], [2008, 0.9]]
}
```

donnee2.json :

```
{
  "label": "Japan",
  "data": [[1999, -0.1], [2000, 2.9], [2001, 0.2], [2002, 0.3], [2003, 1.4], [2004, 2.7], [2005, 1.9], [2006, 2.0], [2007, 2.3], [2008, -0.7]]
}
```

donnee3.json :

```
{
  "label": "USA",
  "data": [[1999, 4.4], [2000, 3.7], [2001, 0.8], [2002, 1.6], [2003, 2.5], [2004, 3.6], [2005, 2.9], [2006, 2.8], [2007, 2.0], [2008, 1.1]]
}
```

Notre partie js

```
$(function() {
  var options = {
    lines: {
      show: true
    },
    points: {
      show: true
    },
    xaxis: {
      tickDecimals: 0,
      tickSize: 1
    }
  };
  var data = [];
  $.plot("#graphic1", data, options);

  function onDataReceived(series) {
    data.push(series);
    $.plot("#graphic1", data, options);
  }
});
```

```

        data = [ series ];
        $.plot("#graphic1", data, options);
    }

    var indice = 0;
    function recupererDonne() {
        ++indice;
        $.ajax({
            url: "data/europe"+indice+".json",
            type: "GET",
            dataType: "json",
            success: onDataReceived
        });
        if (indice > 3 ) {
            data = [];
            indice = 0;
        }
        setTimeout(recupererDonne, 2000);
    }
    setTimeout(recupererDonne, 1); // pour le premier graphique
});

```

Cool (attention de réaliser cela a partir d'un serveur hihi).

### 1.7. Autre symbole

#### Ajouter

```
<script src="js/jquery.flot.symbol.js"></script>
```

#### Js

```

$(function() {
    function creer(offset, amplitude) {
        var res = [];
        var debut = 0, fin = 10;
        for (var i = 0; i <= 50; ++i) {
            var x = debut + i / 50 * (fin - debut);
            res.push([x, amplitude * Math.sin(x + offset)]);
        }
        return res;
    }
    var data = [
        { data: creer(2, 1.8), points: { symbol: "circle" } },
        { data: creer(3, 1.5), points: { symbol: "square" } },
        { data: creer(4, 0.9), points: { symbol: "diamond" } },
        { data: creer(6, 1.4), points: { symbol: "triangle" } },
        { data: creer(7, 1.1), points: { symbol: "cross" } }
    ];
    $.plot("#graphic1", data, {
        series: {
            points: {
                show: true,
                radius: 3
            }
        }
    });
}

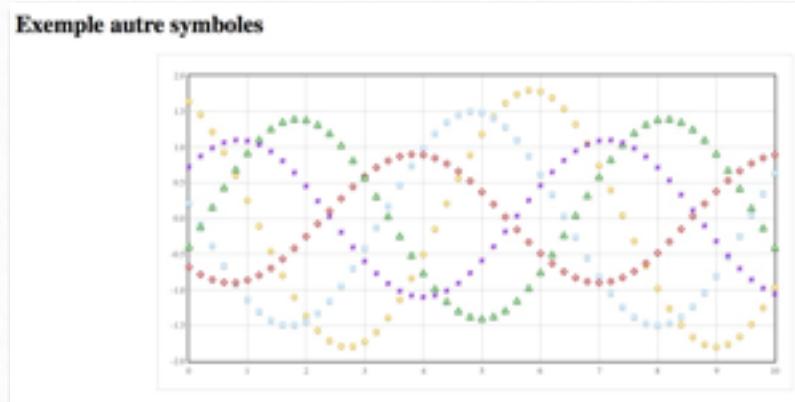
```

```

        },
        grid: {
            hoverable: true
        }
    });
});

```

Voici le résultat :



### 1.8. Camembert Pie charts

Ajouté la librairie :

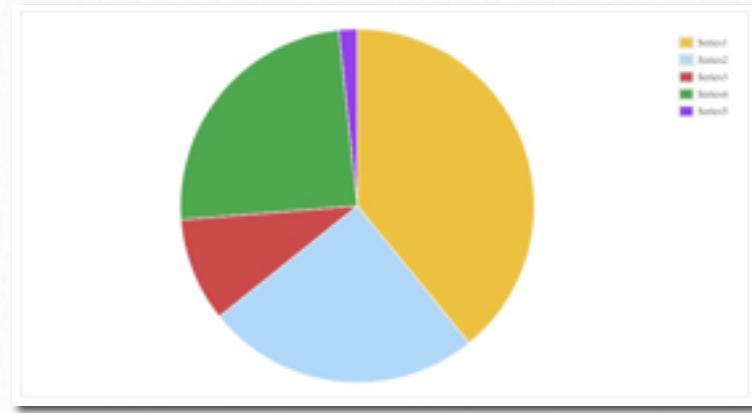
```
<script src="js/jquery.flot.pie.js"></script>
```

js pour le plus simple:

```

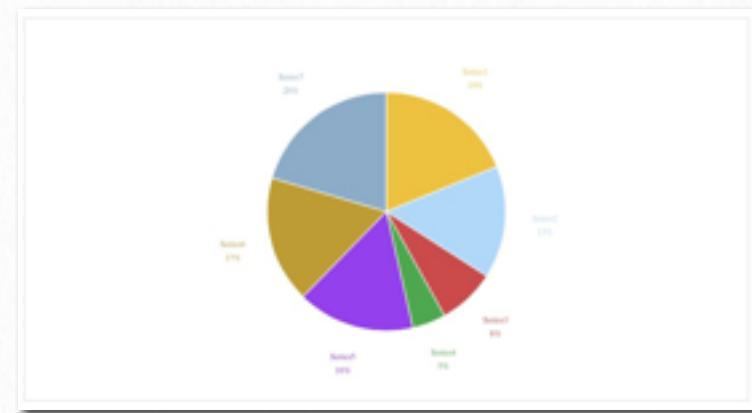
$(function() {
    // Série et valeur aléatoire
    var data = [],
        series = Math.floor(Math.random() * 6) + 3;
    for (var i = 0; i < series; i++) {
        data[i] = {
            label: "Series" + (i + 1),
            data: Math.floor(Math.random() * 100) + 1
        }
    }
    $.plot("#graphic1", data, {
        series: {
            pie: {
                show: true
            }
        }
    });
});
</script>
```

exemple :



sans légende :

```
$ .plot("#graphic1", data, {  
    series: {  
        pie: {  
            show: true  
        }  
    },  
    legend: {  
        show: false  
    }  
});
```



Il existe encore plein plein de possibilité a voir sur le site [www.flotcharts.org](http://www.flotcharts.org).

# 14

## Bootstrap



Bootstrap est une puissante boîte à outil. C'est un framework qui peut rendre votre vie de plus facile pour créer l'architecture d'une page web.

Mais Bootstrap va bien plus loin qu'offrir du code CSS déjà bien organisé et structuré. Il offre aussi des plugins jQuery de qualité pour enrichir vos pages.

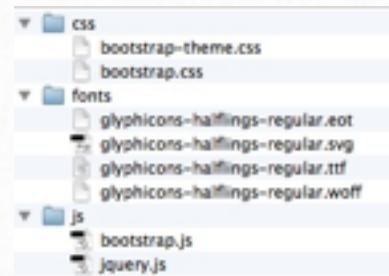
Vous êtes prêt ? Bon on y va.

## Installation

Récupérer le framework à l'adresse suivant <http://getbootstrap.com>, à la création de ce syllabus nous sommes à la version 3.02.

Une fois le fichier décompressé supprimer les versions *min* des js et css, comme bootstrap fonctionne plus mieux avec jQuery , ajouté une version supérieur à 9.00.

Voici les fichiers pour notre étude :



Tapez le code suivant pour vérifier si tout cela est bien fait :

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Test bootStrap</title>
    <link rel="stylesheet" href="css/bootstrap.css">
</head>
<body>
    <bouton class="btn btn-success">ok</bouton>
    <script src="js/jquery.js"></script>
    <script src="js/bootstrap.js"></script>
</body>
</html>
```

voici ce que vous devriez avoir :



Tout va bien ! on continue

# 15

## Annexes



Voici une liste de ressources et lien vers des sites réaliser avec le framework Codeigniter

## 1. Sources pour la réalisation du syllabus :

- <http://www.revillwebdesign.com/codeigniter-tutorial/>
- <http://sldesign.openwab.com/2010/07/20/codeigniter-tutorial-introduction-et-premiere-application/>
- <http://www.siteduzero.com/informatique/tutoriels/codeigniter-le-framework-au-service-des-zeros>
- <http://performancephp.developpez.com/#LII-C-1>

## 2. Livres



## 3. Documentations

Français : [http://codeigniter.fr/user\\_guide](http://codeigniter.fr/user_guide)

Anglais : <http://ellislab.com/codeigniter/user-guide/>

Forum français : <http://forum.codeigniter-france.com>

## 4. Tableau des classes utilitaires de Codeigniter

Nom de la classe utilitaire	Description
Array Helper	Cette classe utilitaire permet de manipuler des tableaux PHP. <u>Fonctions</u> : element() et random_element() <u>Chargement</u> : \$this->load->helper('array') ;
Compatibility Helper	Cette classe utilitaire permet de fournir des méthodes de compatibilité entre PHP 4 et PHP5. <u>Fonctions</u> : file_put_contents(), fputcsv(), http_build_query(), str_ireplace() et strip_tags() <u>Chargement</u> : \$this->load->helper('compatibilité') ;
Cookie Helper	Cette classe utilitaire permet de manipuler des cookies HTTP. <u>Fonctions</u> : set_cookie(), get_cookie() et delete_cookie () <u>Chargement</u> : \$this->load->helper('cookie') ;
Date Helper	Cette classe utilitaire permet de manipuler des dates en PHP. <u>Fonctions</u> : now(), mdate(), standard_date(), local_to_gmt(), mysql_to_unix(), unix_to_human(), human_to_unix(), timespan(), days_in_month(), timezones() et timezone_menu() <u>Chargement</u> : \$this->load->helper('date') ;
Directory Helper	Cette classe utilitaire permet de manipuler des répertoires. <u>Fonction</u> : directory_map () <u>Chargement</u> : \$this->load->helper('directory') ;

Email Helper	Cette classe utilitaire permet de gérer la validité et l'envoi d'emails. <u>Fonction</u> : email () <u>Chargement</u> : \$this->load->helper('email') ;
File Helper	Cette classe utilitaire permet de manipuler des fichiers. <u>Fonctions</u> : read_file(), write_file(), delete_files(), get_filenames(), get_dir_file_info(), get_file_info(), get_mime_by_extension(), symbolic_permissions() et octal_permissions() <u>Chargement</u> : \$this->load->helper('file') ;
Form Helper	Cette classe utilitaire permet de générer par programmation des formulaires HTML. <u>Fonctions</u> : form_open(), form_open_multipart(), form_hidden(), form_input(), form_password(), form_upload(), form_textarea(), form_dropdown(), form_multiselect(), form_fieldset(), form_fieldset_close(), form_checkbox(), form_radio(), form_submit(), form_label(), form_reset(), form_button(), form_close(), form_prep(), set_value(), set_select() et set_radio() <u>Fonctions</u> : <u>Chargement</u> : \$this->load->helper('form') ;
HTML Helper	Cette classe utilitaire permet de gérer des éléments HTML. <u>Fonctions</u> : br(), heading(), img(), link_tag(), nbs(), ol(), ul(), meta() et doctype() <u>Chargement</u> : \$this->load->helper('html') ;
Inflector Helper	Cette classe utilitaire permet de modifier des chaînes de caractères. <u>Fonctions</u> : singular(), plural(), camelize(), underscore() et humanize() <u>Chargement</u> : \$this->load->helper('inflector') ;
Language Helper	Cette classe utilitaire permet de gérer les traductions. <u>Fonction</u> : lang() <u>Chargement</u> : \$this->load->helper('language') ;
Number Helper	Cette classe utilitaire permet de traduire des valeurs en octets dans un format lisibles avec unité. <u>Fonction</u> : byte_format() <u>Chargement</u> : \$this->load->helper('number') ;
Path Helper	Cette classe utilitaire permet de récupérer le chemin absolu d'une ressource. <u>Fonctions</u> : set_realpath() <u>Chargement</u> : \$this->load->helper('path') ;
Security Helper	Cette classe utilitaire permet de faciliter la gestion de la sécurité au sein de votre application. <u>Fonctions</u> : xss_clean(), dohash(), strip_image_tags() et encode_php_tags() <u>Chargement</u> : \$this->load->helper('security') ;
Smiley Helper	Cette classe utilitaire permet de traduire des smileys en image à partir de ressources fournies par le projet Code Igniter <a href="http://codeigniter.com/download_files/smileys.zip">http://codeigniter.com/download_files/smileys.zip</a> <u>Fonctions</u> : get_clickable_smileys, smiley_js() et parse_smileys() <u>Chargement</u> : \$this->load->helper('smiley') ;
String Helper	Cette classe utilitaire permet de manipuler et générer des chaînes de caractères. <u>Fonctions</u> : random_string(), alternator(), repeater(), reduce_double_slashes(), trim_slashes(), reduce_multiples() et quotes_to_entities() <u>Chargement</u> : \$this->load->helper('string') ;
Text Helper	Cette classe utilitaire permet de manipuler du texte. <u>Fonctions</u> : word_limiter(), character_limiter(), ascii_to_entities(), entities_to_ascii(), word_censor(), highlight_code(), highlight_phrase() et word_wrap() <u>Chargement</u> : \$this->load->helper('text') ;
Typography Helper	Cette classe utilitaire permet de gérer la typographie d'un texte dans la page Web. <u>Fonctions</u> : auto_typography () et nl2br_except_pre () <u>Chargement</u> : \$this->load->helper('typography') ;
URL Helper	Cette classe utilitaire permet de manipuler et gérer des URLs. <u>Fonctions</u> : site_url(), base_url(), current_url(), uri_string(), index_page(), anchor(), anchor_popup(), mailto(), safe_mailto(), auto_link(), url_title(), prep_url() et redirect() <u>Chargement</u> : \$this->load->helper('url') ;
XML Helper	Cette classe utilitaire permet de gérer les caractères réservés en XML en les convertissant. <u>Fonction</u> : xml_convert () <u>Chargement</u> : \$this->load->helper('xml') ;

## 5. Tableau des librairies de Codeigniter

Nom de la classe utilitaire	Description
Benchmarking Class	Cette librairie permet de réaliser des mesures de temps de traitements de votre application. <u>Chargement</u> : \$this->output->enable_profiler(TRUE);
Calendar Class	Cette librairie permet de fournir des méthodes de gestion de calendrier. <u>Chargement</u> : \$this->load->library('calendar') ;
Cart Class	Cette librairie permet de gérer un caddy pour site de ecommerce. <u>Chargement</u> : \$this->load->library('cart') ;
Config Class	Cette librairie permet de gérer des fichiers de configuration. <u>Chargement</u> : Automatique
Database Class	Cette librairie permet de manipuler des données dans une base de données. <u>Chargement</u> : \$this->load->library('database') ;
Email Class	Cette librairie permet de manipuler et envoyer des emails avec des fonctionnalités avancées tel que les pièces jointes. <u>Chargement</u> : \$this->load->library('email') ;
Encryption Class	Cette librairie permet de manipuler le chiffrage et de déchiffrage de données. <u>Chargement</u> : \$this->load->library('encrypt') ;
File Uploading Class	Cette librairie permet de gérer l'upload de fichier depuis le navigateur client. <u>Chargement</u> : \$this->load->library('upload');
Form Validation Class	Cette librairie permet de gérer la validation des formulaires HTML. <u>Chargement</u> : \$this->load->library('form_validation') ;
FTP Class	Cette librairie permet de gérer des transferts de fichiers via FTP. <u>Chargement</u> : \$this->load->library('ftp') ;
HTML Table Class	Cette librairie permet de générer des tableaux HTML. <u>Chargement</u> : \$this->load->library('table') ;
Image Manipulation Class	Cette librairie permet de manipuler des images, créer des vignettes, retailler, effectuer des rotations et des fusions. <u>Chargement</u> : \$this->load->library('image_lib') ;
Input and Security Class	Cette librairie permet de gérer le filtrage et la manipulation sécurisée des données. <u>Chargement</u> : Automatique
Loader Class	Cette librairie permet de gérer le chargement des différents éléments dans Code Igniter. <u>Chargement</u> : Automatique
Language Class	Obsolète et remplacé par la classe utilitaire Language.
Output Class	Cette librairie permet de gérer le résultat final envoyé au client. <u>Chargement</u> : Automatique
Pagination Class	Cette librairie permet de gérer la pagination et la navigation dans vos pages. <u>Chargement</u> : \$this->load->library('pagination') ;
Session Class	Cette librairie permet de manipuler et de gérer les sessions. <u>Chargement</u> : \$this->load->library('session') ;
Trackback Class	Cette librairie permet de gérer et manipuler des retroliens ou trackback. <u>Chargement</u> : \$this->load->library('trackback') ;
Template Parser Class	Cette librairie permet de supporter un langage simple de template pour vos pages Web. <u>Chargement</u> : \$this->load->library('parser') ;
Typography Class	Cette librairie permet de gérer la typographie de vos textes. <u>Chargement</u> : \$this->load->library('typography') ;
Unit Testing Class	Cette librairie permet de réaliser et exécuter des tests unitaires. <u>Chargement</u> : \$this->load->library('unit_test') ;
URI Class	Cette librairie permet de manipuler des URI. <u>Chargement</u> : Automatique

User Agent Class	Cette librairie permet de manipuler l'en-tête User-Agent. <u>Chargement</u> : \$this->load->library('user_agent') ;
XML-RPC Class	Cette librairie permet de fournir un service XML au format XML-RPC. <u>Chargement</u> : \$this->load->library('xmlrpc') ;
Zip Encoding Class	Cette librairie permet de manipuler des données à au format ZIP. <u>Chargement</u> : \$this->load->library('zip') ;