EVOGUE

SUPPORT DE COURS JAVASCRIPT / JQUERY / AJAX

Sommaire

➢ Présentation

QU'EST CE QUE LE JAVASCRIPT ?

Javascript est un langage de programmation qui permet d'animer et de dynamiser des pages Web. A ne surtout pas confondre avec le Java qui est un autre langage informatique.
Javascript est surtout utilisé côté client, c'est-à-dire que c'est le navigateur qui lit et interprète le code que l'on écrit en Javascript (voir l'architecture client / serveur pour plus de détails). Même si Javascript est connu pour le Web, il peut être utilisé pour créer d'autres applications. Tom Tom en est un exemple.

HISTORIQUE

1995 est l'année qui marque le début de l'histoire du Javascript.
Il y a 20 ans, le leader sur le monde du Web était la société Netscape Communications avec son navigateur Web Netscape Navigator qui était utilisé par plus de 90% des internautes. Le tout premier navigateur Web s'appelait Mosaïc et vient aussi de la société Netscape.
A l'époque, Netscape est partenaire d'une société informatique du nom de Sun Microsystems, un fabricant d'ordinateurs et éditeur de logiciels dont la réputation n'est plus à faire auprès des professionnels.

Sun développa un langage informatique performant appelé JAVA. Encore très utilisé aujourd'hui par de nombreux systèmes, ce langage est destiné exclusivement à la conception de logiciels. Ce qui le rendait trop puissant et compliqué pour une utilisation au sein d'applications Web.
Ayant flairé ce nouvel eldorado virtuel, Sun et Netscape décidèrent de créer un langage exclusivement réservé au Web. Un langage moins puissant que le JAVA mais beaucoup plus simple d'utilisation, ce qui le rendrait très accessible à qui veut l'apprendre.
Cette mission fût confiée à Brendan EICH, génie technique qui rejoint l'équipe de Netscape en avril 1995.


Brendan EICH

Il avait 10 jours pour créer un langage qui s'adapterait à la nouvelle version du navigateur Netscape. Ce langage fût baptisé Livescript.

Le Livescript était un langage révolutionnaire qui donna naissance aux premières pages web dynamiques.

Par soucis marketing et voulant profiter de la notoriété de son partenaire Sun autour du langage Java, Netscape décide de renommer Livescript en Javascript durant la même année. Ainsi les clients feraient le lien entre Java de Sun et Javascript.

Netscape soumet Javascript à ECMA International (European association for standardizing information and communication systems) pour une standardisation de son langage. ECMA est une organisation active de standardisation dans le monde informatique, similaire au W3C pour le HTML.

ARRIVEE DE JQUERY

JQuery est ce qu'on appel une bibliothèque ou une API (application programming interface) Javascript, c'est à dire un ensemble de fonctions qui permettent d'écrire de manière condensée, les lignes de codes les plus utilisées en Javascript.
Exemple : nous allons souvent utiliser les fonctions Javascript qui permettent d'attraper les balises HTML et les animer, ainsi en Javascript nous allons écrire d'une certaine manière et en JQuery d'une manière plus condensée.

Javascript : *document.getElementsByTagName("LI")[0].innerHTML="Lait";*
JQuery : *$("LI:first").text("Lait");*

Les deux codes font donc la même chose mais ne s'écrivent pas de la même façon.

La version 1.0 stable de JQuery a été crée en aout 2006 par un jeune homme de 22 ans du nom de John RESIG. Aujourd'hui nous en sommes à la 1.11.


John RESIG

➢ Mes premiers mots en Javascript

OU ECRIRE DU CODE JAVASCRIPT ?

Il est possible d'écrire du code JS à deux endroits :

Soit dans un fichier **.js** puis en l'appelant dans notre fichier HTML comme on le ferait avec un fichier style.css

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>Jqueri UI resizable</title>
        <link rel="stylesheet" href="style.css" />
        <script type="text/javascript" src="mon-fichier-javascript.js"></script>
    </head>
    <body>
        <h1>Titre de la page</h1>
    </body>
</html>
```

Soit directement dans notre fichier HTML entre deux balises **<script></script>**

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>Jqueri UI resizable</title>
        <link rel="stylesheet" href="style.css" />

        <script type="text/javascript">

        Écrire mon code JAVASCRIPT ici

        </script>

    </head>
    <body>
        <h1>Titre de la page</h1>
    </body>
</html>
```

Il n'est plus nécessaire d'indiquer le « type = 'text/javascript' » sous HTML 5. Mais cet attribut est obligatoire pour HTML 4 et versions antérieures.

LES OUTILS POUR PRATIQUER

Pour commencer à coder en JS, les seules applications dont nous avons besoin sont :

- Un éditeur de texte (Notepad ++ fera l'affaire)
- Un navigateur (Firefox est recommandé)
- Le module Firebug pour Firefox (pour voir notre code source dans le navigateur)
- Le module Web Developer pour Firefox (nous indique les erreurs JS)

SYNTAXE ET AFFICHAGES

Note importante : Le Javascript est sensible à la casse

Exemples :

```
<script type= « text/javascript »> alert('Bonjour');</script>
```

La fonction d'affichage alert(), permet d'écrire dans une boîte de dialogue,

Ouverture de la balise <script> pour écrire du JS,

Message affiché en **QUOTES** ou **GUILLEMETS**

Une instruction JS se terminera TOUJOURS par un point virgule,

Fermeture de la balise </script>

```
<script type="text/javascript"> confirm('Tu commences le JS');</script>
```

Cette fonction affiche une boîte de dialogue avec le bouton « OK »

```
<script type="text/javascript"> prompt('Quel âge as-tu ?');</script>
```

Affiche une boîte de dialogue avec une question et un champ pour remplir la réponse

```
<script type="text/javascript"> console.log('Voici la console');</script>
```

Affiche le message dans une console

LES COMMENTAIRES

Il est possible d'écrire des commentaires dans notre code JS, les commentaires sont des lignes qui ne seront pas interprétées. Elles servent uniquement à aider le développeur à se repérez et éventuellement informer ses collègues de ce qu'il a fait. Autrement dit, documenter le code.

```
// mon commentaire
```

Le double slash permet d'écrire des commentaires sur une seule ligne

```
/*  mes commentaires
sur plusieurs lignes */
```

Les commentaires sur plusieurs lignes s'écrivent entre slash étoiles. Comme en CSS

➢ Les variables

DEFINITION

« Emplacement de stockage nommé qui peut contenir des données pouvant être modifiées pendant l'exécution du script. Chaque variable a un nom qui l'identifie de façon unique à l'intérieur de son niveau de portée. »
Plus simplement, une variable est une zone en mémoire du système, qui sert à conserver une valeur.

Les variables sont des sortes de boîtes dans lesquelles on stock des données.
Une fois les données stockées dans cette variable, nous pouvons manipuler le contenu de cette variable, le contenu est appelé la « valeur ».

DECLARATION ET AFFECTATION

Exemple :

```
var maBoite = 'Salut toi !';
alert(maBoite);  // Affichera son contenu DONC 'Salut toi !'
```

Pour déclarer une variable on utilise le mot clé « **var** ».
Le signe « **=** » permet d'affecter la variable d'une valeur. Ainsi chaque fois que l'on manipulera une variable cela veut dire que l'on manipule son contenu, donc sa valeur.
Pour faire comprendre à notre machine que l'on utilise une variable et non du texte brut on écrit la variable sans quottes ni guillemets.

Exemple :

```
var monMessage= 'Coucou !';
alert(monMessage);  // Affichera son contenu DONC 'Coucou !'
alert('monMessage');  // Affichera 'monMessage' car se trouve entre quottes
```

Nous pouvons stocker plusieurs sortes de données dans les variables

```
var monPrenom = "Pierre";
alert(monPrenom) ; // affichera Pierre
var monAge = 27;  //les chiffres ne prennent pas de quottes
var monAge = prompt('Quel âge as-tu ?'); // Ici j'affecte la valeur que l'internaute rentrera dans le champ généré par prompt
var monPrenom = "Marie"; // Ici Marie prendra la place de Pierre dans la variable
alert(monPrenom) ; // affichera Marie
```

LES PORTEES DES VARIABLES

Lorsqu'on affecte une variable, nous pouvons manipuler la valeur affectée à cette variable.
Elle nous est donc accessible. Cependant, cette accessibilité dépend de l'endroit où la variable a été affectée. On parle donc de la portée de la variable.

- Une variable déclarée en dehors de toute fonction peut être utilisée n'importe où dans le script. On parle alors de ***VARIABLE GLOBALE***.
- Une variable déclarée dans une fonction aura une portée limitée à cette seule fonction, c'est-à-dire quelle est inutilisable ailleurs. On parle alors de ***VARIABLE LOCALE***.

Pour déclarer une variable à l'intérieur d'une fonction et la rendre GLOBALE donc accessible de partout dans le script, celle-ci doit être déclarée sans le mot clé ***var***. *Exemple :*

```
monMessage = 'Salut' ;
```

Pour bien comprendre l'utilité de la portée des variables ainsi que leur utilisation, se reporter au chapitre des fonctions.

LES TYPES DE VARIABLES

En Javascript comme dans tous les langages de programmation il existe plusieurs types de données.
La fonction **typeof()** permet d'obtenir le type d'une donnée

Les nombres :

```
var a = 36 ;
alert(typeof(a)) ; // Affichera « number »
```

Les nombres simples sont dits « integer » et les nombres à virgule sont dits « float » ou « double ».
On remarquera que lorsqu'on affecte un nombre, on ne met **pas de quottes ni de guillemets**.

Les chaines de caractères (textes) :

```
var a = 'Salut !' ;
alert(typeof(a)) ; // Affichera « string »
```

Les chaines de caractères ou textes sont appelés « string ».

Les booléens :

```
var a = true ;
var b = false ;
```

Une donnée a toujours une valeur, c'est-à-dire qu'elle est soit VRAIE soit FAUSSE (true ou false).
Si var = 0, sa valeur est considérée comme FALSE,
Si var = 1, ou n'importe quelle autre valeur ('salut' ou autre..), sa valeur est considérée comme TRUE.

➢ Les structures conditionnelles

DEFINITION

Nous appelons structure conditionnelle, la structure qui permet de réaliser une (ou plusieurs) instruction(s) sous certaines conditions. Elle nécessite l'utilisation des opérateurs de comparaisons et parfois des opérateurs logiques.
En Javascript, comme dans tout langage de programmation, nous utilisons des conditions.
Mettre une condition consiste à écrire du code Javascript, qui ne sera exécuté que si la personne rempli la condition.

Par exemple, à l'aide du code JS, je demande à l'internaute quel est son sexe. Celui-ci me répond « féminin » ou « masculin ».
S'il répond « féminin », j'affiche un background rose.
Sinon, s'il répond « masculin », j'affiche un background bleu.
Sinon s'il répond autre chose, ou ne répond pas, je lui écris « tu ne réponds pas à la question ».

SYNTAXE DES CONDITIONS IF, ELSE IF, ELSE

```
if (ma condition) // voir les opérateurs de comparaisons, sous-chapitre suivant.
            { // Ne pas oublier l'ouverture d'accolade
                  alert("Mon message"); // Ce code ne sera exécuté QUE si la
condition est remplie, NE PAS OUBLIER LE POINT VIRGULE
            } // IMPORTANT : Ne pas oublier les ACCOLADES, JAMAIS de POINT VIRGULE
à la fermeture d'accolades
      else if (mon autre condition)
            {
                  alert("mon autre message");
            }
      else
            {
                  alert("mon dernier message");
            }
```

```
var question = prompt('De quel sexe es-tu ?') ;  // Ici la réponse de l'internaute sera stockée dans la variable « question ».
if(question == 'masculin') //Si la valeur de la variable est « masculin »
	{
		alert("Tu es un homme"); // Ce code ne sera exécuté QUE si la réponse est « masculin »
	}
 else if(question == 'feminin') // Sinon si la valeur de la variable est « feminin »
	{
		alert("mon autre message"); // Le code précédent ne sera pas exécuté et CE code ci ne sera exécuté QUE si la réponse est « féminin »
	}
else // Sinon, c'est-à-dire dans tous les autres cas.
	{
		alert("Ah .. Tu n'es ni l'un ni l'autre ?");
	}
```

LES OPERATEURS DE COMPARAISON

Lorsqu'on pose une condition, pour que l'ordinateur comprenne que la condition est valide ou invalide. Le résultat de cette condition est forcément true ou false.

Exemple dans le cas précédent, notre condition était if(question == 'masculin') ;

Si la condition est remplie, le résultat sera true.

Si la condition n'est pas remplie, le résultat sera false.

Le résultat est donc un booléen.

Les opérateurs de comparaison permettent de tester, l'égalité ou la différence entre deux données.

Tous les tests sont à placer dans la condition if entre les deux parenthèses.

Égal (**==**) Renvoie true si les valeurs des données sont égales.

```
if(3 == var1)
if("3" == var1)
if(3 == '3')
```

Différent (***!=***) Renvoie true si les valeurs des données ne sont PAS égales.

```
if(var1 != 4)
if(var2 != "3")
```

Strictement égal (**===**) Renvoie true si les valeurs des données sont égales ET du même type.

```
if(3 === var1)
```

Strictement différent (***!==***) Renvoie true si les valeurs des données ne sont PAS égales et/ou PAS du même type.

```
if(var1 !== "3")
if(3 !== '3')
```

Plus grand que (**>**) Renvoie true si la valeur de gauche est plus grande que la valeur de droite.

```
if(var2 > var1)
if("12" > 2)
```

Plus grand ou égal (**>=**) Renvoie true si la valeur de gauche est plus grande ou égale à la valeur de droite.

```
if(var2 >= var1)
if(var1 >= 3)
```

Plus petit que (**<**) Renvoie true si la valeur de gauche est plus petite que la valeur de droite.

```
if(var1 < var2)
if("12" < var2)
```

Plus petit ou égal (**<=**) Renvoie true si la valeur de gauche est plus petite ou égale à la valeur de droite.

```
if(var1 <= var2)
if(var2 <= 5)
```

LES OPERATEURS LOGIQUES

Tous les langages de programmation utilisent les opérateurs logiques. Ce sont des opérateurs qui viennent se placer entre deux évaluations dans une condition.

Exemple, je veux tester si mon internaute est masculin ET qu'il a plus de 18 ans.

Cela s'écrira :

```
if(sexe == 'masculin' && age > 18)
```

L'opérateur logique && vérifie que les deux conditions sont bien remplies en même temps.

JavaScript compte 3 opérateurs logiques :

- && qui veut dire « ET »
- || qui veut dire « ET/OU »
- ! qui veut dire « INVERSE »

L'opérateur && :

Il évalue le résultat en TRUE si les 2 expressions sont TRUE. Si lune des deux expressions testées est FALSE alors le résultat sera FALSE.

```
true && true // => true
true && false // => false
false && true // => false
false && false // => false
```

L'opérateur || :

Il évalue le résultat en TRUE si une ou l'autre ou les 2 expressions sont TRUE. Si les 2 expressions sont FALSE alors le résultat sera FALSE.

```
true || true // => true
true || false // => true
false || true // => true
false || false // => false
```

L'opérateur ! :

Il transforme ce qui est TRUE en FALSE et vice versa.

```
!true // => false
!false // => true
```

SYNTAXE DES CONDITIONS DE CHOIX SWITCH

La syntaxe des conditions en SWITCH intervient quand on veut tester plusieurs valeurs sur une même variable.
Exemple, si l'on demande à un internaute de nous dire quelle est sa couleur préférée, il serait très redondant d'écrire des if et des else if pour écrire chaque couleur. Alors on utilise une condition SWITCH.

Dans cet exemple, nous demandons à l'internaute d'écrire une couleur. Nous stockons sa couleur dans la variable « saCouleur ».
Ensuite, nous exécutons des instructions de code, en fonction de la couleur qu'il aura renseigné.
Notre ordinateur va tester la variable en comparant sa valeur avec chaque couleur, jusqu'à ce qu'il trouve la couleur correspondante.
Une fois la couleur trouvée, il va exécuter les instructions de code rentrées pour ladite couleur.

```
var saCouleur = prompt("Quelle est ta couleur ?");
      switch(saCouleur){  // j'indique ici la variable que je vais tester dans mon switch
              case 'jaune' : // ici ne pas oublier les deux points
                    alert('tu aimes le jaune') ; // point virgule a la fin des instructions
             break ; // le BREAK pour indiquer qu'on a fini les instructions pour ce cas.
              case 'rouge' :
                    alert('tu aimes le rouge') ;
             break; // point virgule à la fin de chaque break
             case 'vert' :
                    alert('tu aimes le vert') ;
             break ;
             case 'bleu' :
                    alert('tu aimes le bleu') ;
             break ;
             default : // le « default » sert à indiquer tous les autres cas
                    alert('ta couleur n\'est pas dans la liste') ; // (vous remarquerez que pour écrire une apostrophe à l'intérieur de deux quottes, nous devons mettre un « \ ». Cela s'appel un caractère d'échappement qui permet de dire à l'ordinateur de ne pas le comptabiliser comme une fermeture de quotte.
             break ;
      }  // fermeture d'accolade
```

➢ Structure itérative : les boucles

DEFINITION

Nous appelons structure itérative (ou récursive), la structure qui permet d'exécuter plusieurs fois les mêmes instructions. Elle permet de faire "en boucle" un bloc d'instructions.
En Javascript comme dans tous les langages de programmation. Nous utilisons les boucles.
Ce sont des instructions de code qui se répètent tant que la condition est respectée.
Exemple, je veux écrire tous les nombres de 1 à 99.
Au lieu de tous les écrire à la main, je vais demander à l'ordinateur de le faire pour moi en lui disant « Tant que mon chiffre est inférieur à 99, écris-le et ajoute lui 1 ».

LA BOUCLE FOR

```
for(var monChiffre = 1 ; monChiffre < 100 ; monChiffre++)
{
	document.write(monChiffre) ;
}
```

Dans les parenthèses de la boucle for : départ ; condition ; incrémentation
Au départ, j'affecte une valeur à ma variable. Cette valeur est un chiffre.
Ma condition dit « tant que ce chiffre est inférieur à 100 ».
Mon incrémentation consiste à ajouter +1 à chaque tour de boucle.

Dans mes accolades, ce code sera exécuté à chaque tour de boucle, donc il sera exécuté 99 fois.
Note : la méthode document.write() permet d'écrire dans notre page HTML. Nous verrons cette méthode dans un chapitre consacré au Document Object Model.

LA BOUCLE WHILE

La boucle WHILE est une autre manière d'écrire une boucle. Basée sur le même principe que la boucle for, ses instructions seront exécutées tant que la condition est remplie.
La syntaxe de la boucle WHILE est plus proche de l'humain que celle de la boucle FOR même si la boucle FOR est très utilisée par les développeurs.

Reprenons notre exemple du compteur de chiffre. Cette fois écrit en version WHILE :

```
var monChiffre = 1; // déclaration et affectation de ma variable
	while(monChiffre < 100) // je pose ma condition ici, qui sera vérifiée à chaque tour de boucle (tant que mon chiffre est inférieur à 100)
	{
		document.write(monChiffre); // mes instructions à l'intérieur des accolades (écris ce chiffre)
		monChiffre++; // (ajoute lui 1)
	}
```

ATTENTION : « monChiffre ++ » est l'instruction qui change ma variable. Si je retire cette instruction alors la condition sera toujours remplie car 1 sera toujours inférieur à 100. Donc je rentrerai dans une boucle infinie qui fera planter mon navigateur.

LA BOUCLE DO WHILE

La boucle DO WHILE, est une boucle qui exécutera d'abord une première fois le code SANS tester la condition ; PUIS ensuite, n'exécutera le code que si la condition est remplie pour toutes les fois suivantes.

Exemple :

```
	var maCondition = false; // (souvenez vous les booleens true / false)
	do{
		document.write('ce code ne s\'exécutera qu\'une seule fois parce-que je ne  rempli pas la condition…');
	} while(maCondition == true); // Remarque : ici je mets un point virgule car j'écris ce   code après la fermeture d'accolade.
```

➢ Les fonctions

DEFINITION

JavaScript met à notre disposition un ensemble très vaste de fonctions prédéfinies, couvrant les besoins de base propres à toute programmation. Il y en a un certain nombre.
A l'inverse et en complément, une fonction utilisateur est une fonction écrite par le développeur.
Une fonction est un ensemble d'instructions exécutées à l'appel de celle-ci.
L'intérêt des fonctions est de permettre la réutilisation à l'envi d'un ensemble d'instructions. Cela diminue alors le nombre de lignes de codes (donc le poids des fichiers) et surtout cela simplifie la vie du développeur.
Une fonction se fait toujours en deux étapes : la déclaration, puis l'exécution.

En Javascript comme dans tous les langages de programmation, nous utilisons des fonctions.
On distingue les fonctions prédéfinies (ce sont les fonctions déjà présentent dans le code Javascript), et les fonctions utilisateurs (ce sont des fonctions que nous créons nous même).
L'objectif des fonctions est de nous simplifier la vie.
Typiquement, lorsque l'on répète souvent le même bloc d'instructions, nous rentrerons ce bloc une seule fois dans une fonction que nous créons (utilisateur) et chaque fois que l'on va exécuter cette fonction, cela aura pour effet d'exécuter toutes les instructions qui s'y trouvent sans devoir les réécrire à chaque fois.

La fonction prédéfinie alert() que nous avons vu permet de générer une boîte de dialogue.
La fonction prédéfinie typeof() que nous avons vu aussi permet d'obtenir le type d'une donnée.
Important : les fonctions s'écrivent toujours avec des parenthèses.

LES FONCTIONS PREDEFINIES

```
var maVariable = 15;
typeof(maVariable); // fonction typeof() qui m'indique le type de ma donnée
isNaN(maVariable); // fonction isNaN() qui teste ma variable et me renvoie « NaN »
(pour Not a Number) si ce n'est pas un nombre
maVariable.substring(3,10); // me renverra une erreur car la fonction substring()
attend un type STRING et non NUMBER dans ses parenthèses
```

```
maVariable = 'Hamburger';
maVariable.substring(3,9); // affichera « burger »
```

Tout ce qui est entré entre parenthèses s'appel des arguments ou paramètres (ce sont des synonymes).

Les arguments sont les données à entrer dans les parenthèses, données sur lesquelles notre fonction va travailler.

Exemple pour la fonction typeof(). Je rentre en argument la variable dont je veux connaitre le type.

Lien utile qui référence les fonctions prédéfinies par type : www.toutjavascript.com/reference/

LES FONCTIONS UTILISATEUR

Déclarer une fonction utilisateur sans argument :

```
var coucou = function(){
	alert('salut) ;
} ;
```

Executer cette fonction :

```
coucou() ;
```

J'ai crée une fonction qui me permet d'écrire 'salut' chaque fois que je l'exécute.

Déclarer une fonction avec 1 argument :

```
var carre = function(nombre){
	return nombre * nombre ;
} ; // ATTENTION au point virgule
```

Executer cette fonction :

```
carre(5) ; // affichera 25
```

J'ai crée une fonction avec un argument. Qui me permet de calculer le carré d'un nombre chaque fois que celui-ci est indiqué dans les parenthèses lors de l'exécution.

Déclarer une fonction avec plusieurs arguments

```
var calculTva = function(montant, taux){
	return montant * taux;
};
```

Executer cette fonction :

```
calculTva(10, 1.2) ; // affichera 12
```

Autre manière de declarer une fonction :

```
function carre(nombre){
	return nombre * nombre;
} // ATTENTION, DANS CETTE MANIERE IL N'Y A PAS DE POINT VIRGULE
```

Le mot clé RETURN, permet de sortir le résultat de la fonction.

Les opérateurs de calculs :

- `+` pour l'addition
- `-` pour la soustraction,
- `/` pour la division,
- `*` pour la multiplication

LES FONCTIONS ET LA PORTEE DES VARIABLES

Selon l'endroit où une variable est déclarée et affectée, celle-ci pourra être accessible (visible) de partout dans le script ou bien uniquement dans une portion confinée du code, on parle de **« portée » d'une variable.**

Lorsqu'une variable est déclarée, **SANS LE MOT CLÉ VAR**, c'est-à-dire de façon implicite, **(peu importe l'endroit où elle est déclarée, que ce soit dans la fonction ou pas)**, elle est **ACCESSIBLE DE PARTOUT DANS LE SCRIPT** (n'importe quelle fonction du script peut faire appel à cette variable). On parle alors de **VARIABLE GLOBALE**

La portée d'une variable déclarée **AVEC LE MOT CLÉ VAR**, c'est-à-dire de façon explicite, dépend de l'endroit où elle est déclarée :

* Une variable **déclarée explicitement en dehors de toute fonction**, pourra être utilisée n'importe où dans le script. On parle de ***VARIABLE GLOBALE***.

* Une variable **déclarée explicitement à l'intérieur d'une fonction** aura une portée limitée à cette seule fonction, c'est-à-dire quelle est inutilisable ailleurs. On parle alors de ***VARIABLE LOCALE***.

Voici deux exemples permettant de l'illustrer :

```
<script type="text/javascript">
var a = 12;
function MultipliePar2(b) {
   var a = b * 2;
   return a;
}
document.write("Le double de 4 est ",MultipliePar2(4));
document.write('<br />');
document.write("La valeur de a est ",a);
</script>
```

Dans l'exemple ci-dessus, la variable a est déclarée explicitement en début de script, ainsi que dans la fonction.

Voici le résultat qu'affiche le script précédent :

```
"Le double de 4 est 8"
"La valeur de a est 12"
```

Voici un autre exemple dans lequel a est déclarée implicitement dans la fonction :

```
<script type="text/javascript">
var a = 12;
function MultipliePar2(b) {
   a = b * 2;
   return a;
}
document.write("Le double de 4 est ",MultipliePar2(4));
document.write('<br />');
document.write("La valeur de a est ",a);
</script>
```

Voici le qu'affiche le second script :

```
"Le double de 4 est 8"
"La valeur de a est 8"
```

Ces exemples montrent la nécessité de déclarer systématiquement des nouvelles variables avec le mot clé var.

➢ Les tableaux de données (Arrays)

DEFINITION

Un tableau (array en anglais) est une structure de données de base qui regroupe un ensemble d'éléments (des variables ou autres entités contenant des données), auquel nous avons accès à travers un numéro d'index (ou indice).
Là où une variable stocke une unique valeur, un tableau peut en stocker une multitude. Chaque valeur est alors repérée par un indice (ou encore : une clé, un index).

SYNTAXE ET UTILISATION

Plusieurs manières de déclarer et remplir un Array :

```
var mon_tableau = ["pierre", 'julie', "sarah", 10, 6];
```

```
var mon_tableau = Array();
mon_tableau = ["pierre", 'julie', "sarah", 10, 6];
```

```
var mon_tableau = new Array();
mon_tableau = ["pierre", 'julie', "sarah", 10, 6];
```

Les données de type STRING (chaine de caractères), s'écrivent entre " " ou entre ' ' et les données de type NUMBER s'écrivent sans quottes ni guillemets.
INDEX : chaque donnée contenue dans notre Array possède un INDEX.
L'index est le référent qui va permettre de cibler et d'appeler une donnée dans notre tableau.
L'index correspond à la position où se trouve la donnée qu'on veut atteindre.
Dans l'exemple du haut Pierre est à la première position, Julie à la seconde etc. ...

Il suffit donc de compter les positions pour savoir comment cibler notre donnée.

Précision IMPORTANTE : l'ordinateur compte à partir de 0. Donc la position de Pierre est 0, celle de Julie est 1, etc. ...

Pour cibler Julie nous procédons comme suit :

```
mon_tableau[1]; // ici j'obtiens Julie
```

Donc pour l'afficher :

```
document.write(mon_tableau[1]) ; // affichera Julie
```

Petite parenthèse : L'indexation est aussi valable pour les données seules de type « string »

Exemple :

```
var monNom = "Ziad";
alert(monNom[0]); // affichera "Z";
```

Comment faire si notre Array contenait 200 données ? donc autant de positions index.

ON utilise une boucle FOR

```
	var mon_tableau = ["pierre", "arnaud", "leila", "julie", "linda", "eric"];

	for(var index = 0 ;  i < mon_tableau.length ;  index++)  // length pour un STRING sert à donner le nombre de caractères, et length pour un ARRAY sert à donner le nombre d'index dans le tableau.
	{
		document.write(mon_tableau[index] + " est quelqu'un de bien");
	} // le "+" présent dans le code sert à concaténer deux morceaux de code entre eux, c'est-à-dire qu'on met l'un à côté de l'autre un morceau de code qui a besoin de quottes et un autre qui n'en a pas besoin.
```

Explication du code :

Je déclare un tableau et le rempli,

Je crée une boucle qui porte comme condition « tant que ma variable index est inférieure au nombre d'index de mon tableau » (la propriété length me donne un chiffre qui correspond au nombre d'index du tableau)

Les instructions de ma boucle sont d'afficher mon_tableau[le chiffre index],

DONC, toutes les valeurs correspondantes à chaque index.

Les Arrays peuvent même contenir des arrays qui eux mêmes contiennent des arrays etc. ... Un peu comme les poupées russes. On appel ça des tableaux multidimensionnels (tableaux qui contiennent des tableaux).

Pour déclarer et remplir un Array multidimensionnel :

```
var deuxDimensions = [ [1, 'pierre', 3], [1, 1, 'laurie'] ]; // ici nous avons crée un tableau de 2 dimensions avec 2 lignes et 3 colonnes.
document.write(deuxDimensions[0][1]) ; // affiche pierre
```

Il est possible de remplir les données du tableau une par une comme pour une variable normale.

```
var mon_tableau[0] =  'pierre' ;
var mon_tableau[1] = 'marie' ;
var mon_tableau[0] = 'eric' ; // comme avec les variables simpls, Eric prendra la place de Pierre
var mon_tableau['etudiant'] = 'arnaud' ; // il est possible de remplacer l'index numérique par un nom. Notre index est alors nommé. Cela ne change pas la règle pour le cibler ou l'afficher
document.write(mon_tableau['etudiant']) ; // affichera arnaud
```

Note importante : Lorsqu'on crée un Array qui ne porte aucun index numérique mais seulement des index nommés, beaucoup de développeurs vont alors parler de « tableau associatif ».
Mais il faut savoir que lorsqu'on crée ce genre d'Array, les fonctions prédéfinies pour les tableaux Arrays comme concat(), ou la propriété length que l'on a vu précédemment, ne fonctionnent pas sur ces tableaux associatifs. Pour qu'elles fonctionnent il faut que les index restent numériques.

➢ Les Objets

DEFINITION

Comme dans la langue française, nous avons les NOMS (assimilables aux "choses") et les VERBES (assimilables aux "actions").
Dans le langage informatique c'est pareil, nous avons aussi les noms : données d'information comme les number ou les string ET les verbes : qui correspondent aux fonctions.
Jusqu'ici, les deux étaient séparés.
Avec les objets, nous pouvons stocker notre information ET les fonctions qui utilisent cette information en même temps.
Les informations contenues dans notre objet se nomment des **propriétés.** Et les fonctions (donc les actions faisables) contenues dans notre objet se nomment des **méthodes**.
Prenons par exemple l'objet voiture. A l'intérieur de cet objet nous allons retrouver des propriétés : la couleur, les pneus, les sièges. ET, nous allons retrouver des méthodes : démarrer, rouler, tourner à gauche, tourner à droite.

SYNTAXE

On peut considérer que les objets sont une combinaison de clé-valeur. Un peu comme avec les index dans les arrays.
Les valeurs des objets peuvent être de n'importe quel type (string, number, array ou même objet)

```
var monObjet = {
            name : 'pierre', // valeur de type string
            age : 27, // valeur de type number
            interets : ['jouer', 'rire', true, 25] // valeur de type objet
            }; // IMPORTANT : le POINT VIRGULE à la fin du stockage, les DEUX
POINTS pour chaque rapport clé : valeur. La VIRGULE à la fin de chaque déclaration
de clé-valeur sauf pour la dernière déclaration.
```

Déclaration et remplissage de l'objet

```
var monObjet = {}; // déclaration d'un objet vide
```

Autre syntaxe pour créer un objet, appelée « the object constructor »

```
var monObjet = new Object(); // on crée l objet. ATTENTION au « O » majuscule
monObjet["name"] = "Charlie"; // on rempli l objet.
monObjet.name = "Charlie"; // autre facon de remplir l objet.
```

Exemple de création ET remplissage d objet.

```
var contactRepertoire = {}; // On crée l'objet
contactRepertoire.nom = 'Pierre DUPONT'; // syntaxe clé-valeur. nom est la clé
contactRepertoire.numero = '06 68 62 54 95';
contactRepertoire.telephoner = function() {
            document.write('Appel ' + this.nom + ' au ' + this.numero + '...');
            };
```

contactRepertoire est l'objet,

nom est une propriété,

numero est une propriété

telephoner est une méthode

Important : Le mot clé « this » signifie « lui-même » (référence à l'objet)

cibler une propriété ou une méthode :

```
contactRepertoire.telephoner() ; // affiche Appel Pierre DUPONT au 06 68 62 54
95...
contactRepertoire.nom ; // affiche Pierre
```

LE MOT CLE « THIS »

Le mot-clé THIS, fonctionne à la manière d'un PLACEHOLDER permettant d'indiquer à notre méthode quelle devra fonctionner pour tous les objets faisant appel à elle. En clair, THIS est une sorte d'objet témoin.

Exemple :

```
var vieillir = function(nouvelAge) {
        this.age = nouvelAge; // ici THIS va servir pour tous les objets qui vont utiliser la méthode « vieillir »
      }; // REMARQUE IMPORTANTE : Nous avons créé notre méthode en DEHORS de tout objet.
      var pierre = new Object(); // Maintenant nous créons notre objet
      pierre.age = 30; // Nous lui donnons un âge initial
      pierre.changeAge = vieillir; // nous donnons à notre objet la méthode qui permet de changer d'âge.

      pierre.changeAge(50); // en appelant la propriété changeAge, celle-ci appel la méthode vieillir, qui s'occupe de faire le changement pour l'objet concerné grâce à THIS.
```

PARCOURIR UN OBJET AVEC LA BOUCLE FOR

Pour connaitre les propriétés et les méthodes que contient un objet, nous utilisons la boucle For avec une syntaxe un peu particulière destinée à parcourir un objet.

```
for(var prop in contactRepertoire) {
                    document.write(prop);
                    document.write('<br />');
                }
```

prop est l'argument qui va définir les propriétés et méthodes,

le mot clé ***in*** est obligatoire pour respecter la syntaxe,

contactRepertoire est l'objet que nous voulons parcourir

CE QU'IL FAUT RETENIR SUR LES OBJETS

Tous les éléments rencontrés dans le Javascript sont des objets.

CREER UN OBJET : 2 façons

```
var monObjet = {}; // 1ere façon
var monObjet = new Object(); // 2eme façon
```

REMPLIR UN OBJET : 3 façons

```
monObjet = {
            prenom : 'Arnaud',
            loisirs : {
                                sport : 'football',
                                cinema : 'the artist'
                            }, // ici la valeur de la 2eme propriété est un objet
            nom : 'DELACRE',
            }; // 1ere façon
monObjet.propriete = 'valeur'; // 2eme façon
monObjet['propriete'] = 'valeur'; // 3eme façon
```

CIBLER LA VALEUR CONTENU DANS LA PROPRIETE DUN OBJET : 2 façons

```
monObjet.saPropriete; // 1ere façon
monObjet['saPropriete']; // 2eme façon attention aux quotes ! (sauf si la propriété est contenue dans une variable)
monObjet.prenom.loisir.sport ; // ici je cible le football, je descends toute la hiérarchie
```

Nous pourrions développer très longuement sur le chapitre des objets, parler des classes, des prototypes et de la programmation orientée objet. Mais cela n'est pas nécessaire pour la suite du cours et l'utilisation qui nous intéresse à savoir JQuery.

Connaitre l'existence des propriétés, des méthodes et savoir utiliser leur syntaxe pour les cibler suffira pour comprendre et manipuler JQuery.

➢ Le DOM (Document Object Model)

DEFINITION

Le DOM (Document Object Model) est un programme interne à notre document HTML (ou XML) qui fait en sorte que chaque élément (balises <h1>, <body>, <p> ...), chaque événement (clic de la souris, chargement de la page ...), chaque attribut (href, alt, title ...) est RÉCUPÉRABLE, MANIPULABLE et EXPLOITABLE par un langage de programmation.
Dans notre cas, c'est le Javascript qui va récupérer chaque "composite" de notre document HTML et l'exploiter.
Cette manipulation est possible car l'API (Interface de Programmation) DOM fait en sorte que tout ce qui compose notre document HTML soit un objet qui peut être récupéré et exploité par le Javascript (ou un autre langage de programmation).
Autrement dit, tous les éléments, attributs, et événements qui composent notre document HTML et le document HTML lui-même sont des objets utilisables par le Javascript.
Note importante : Lorsqu'on est dans le DOM, les balises HTML portent le nom d'éléments. Nous allons donc parler d'éléments.

UTILISATION DU DOM

A l'aide de Javascript :

- je sélectionne la balise <a> (donc l'élément <a>) et je modifie sa couleur (DOM document + DOM élément)
- je sélectionne l'événement "clic gauche" et j'émets un message (chaque fois que je clic un message apparait) (DOM document + DOM evenement)
- je sélectionne les attributs "title" et change leur contenu (je remplace title="image2" par title="beau tigre") (DOM document + DOM attribut)

- au moment où ma page HTML a fini de se charger, j'émets un message 'coucou' (DOM document + DOM élément)

Exemples d'utilisation du DOM avec le code Javascript

Le DOM « document » qui permet de sélectionner :

```
document.getElementById() ; // par son ID
document.getElementsByName() ; // par son attribut « name »
document.getElementsByTagName() ; // par son nom de balise HTML
```

Le DOM « element » qui permet de faire une action sur les balises qu'on a sélectionné (exemple supprimer l'attribut) :

```
var mesDivs = document.getElementsByTagName("div"); // selectionne toutes les divs
mesDivs.removeAttribute("style"); // supprime leur attribut « style »
```

Le DOM « events » qui permettent de faire une action lors d'un événement (exemple au clic de la souris)

```
<button alt="coucou" id="boutton">essai</button>
<script>
document.getElementById("boutton").onclick=function(){alert('allo');};
</script> // affiche « allo » au clic de la souris
```

Aujourd'hui la bibliothèque JQuery a permis de remplacer tous les « getElementsBy » par le signe « $ ».

Typiquement :

```
document.getElementById("boutton").onclick ; // version Javascript pure
$("boutton").click ; // version JQuery
```

➢ JQuery : write less do more

LES SELECTEURS JQUERY

Tout d'abord il faut télécharger la bibliothèque JQuery ici :

http://code.jquery.com/jquery-1.11.0.min.js

Après avoir copié / collé ce code dans un fichier .js (l'appeler jquery-1.11.0.min.js pour le reconnaitre)

L'intégrer dans votre document HTML comme on le fait pour un fichier CSS :

```
<!DOCTYPE html>
<html lang="fr">
    <head>
        <title>JQuery</title>
        <link rel="stylesheet" href="css/style.css" />
        <script type="text/javascript" src="../lib/jquery-1.11.0.min.js"></script>

    <script> <!--  BALISE pour écrire du Javascript -->

            Écrire mon code JQUERY ici

    </script>
    </head>
    <body>
        <h1>Titre de la page </h1>
    </body>
</html>
```

Il n'y a plus qu'à écrire du code JQuery …

Le signe « $ » de JQuery remplace absolument tout les « getElementsBy » du DOM en Javascript.

Il permet à lui seul de sélectionner n'importe quel « composite » de notre document HTML (balises, attributs, ID, classes, pseudo-éléments etc …)

Les sélecteurs principaux :

```
$("*")  // permet de sélectionner tous les éléments HTML
$("h1") // sélectionne toutes les balises H1
$("#nom-de-ID") // permet de sélectionner ls éléments par leur ID
$(".nom-de-la-classe") // par leur classe
$("h1, a, #nom-id, .nom-classe, div,p") // plusieurs éléments mélangés (classes, id, nom)
$("div > p") // sélectionne P, enfants directs de DIV
$("div  p") // sélectionne P, dans la déscendance de DIV (qu'il soit enfant, sous-enfant ou sous-sous-enfant …)
$("input:text") // le type « text » des input
$(":text") // tous les types text
$("img[alt]") // tous les IMG ayant l'attribut ALT
$("[alt]") // tous les attributs ALT
$("img[title='tigre']") // tous les IMG ayant l'attribut ALT qui porte la valeur « tigre »
```

Ce qu'il faut retenir pour les sélecteurs c'est que pour 90% d'entre eux, la manière de les cibler est exactement de la même manière qu'on le ferait dans une feuille de style CSS.
Lien recensant tous les sélecteurs JQuery :
http://www.w3schools.com/jquery/jquery_ref_selectors.asp

Exemple d'utilisation d'un sélecteur JQuery :

```
<h1>Mon Titre</h1>
<script>
$("h1").click(
      function(){alert("Hello !"); } // je sélectionne <H1> et au clic (sur lui-même), j'exécute cette fonction.
) ;
</script>
```

LES EVENEMENTS JQUERY

Les événements JQuery sont les événements de type « au clic de la souris », « au survol de la souris » etc. …

Les événements principaux :

```
click() // au clic de la souris
dblclick() // au double clic de la souris
mouseenter() // au moment où la souris entre dans un espace alloué à un élément
mouseleave() // au moment où la souris quitte un espace alloué à un élément
hover() // au moment où la souris entre dans un espace alloué à un élément et le quitte
mousedown() // au moment où l'on clic et maintient le clic de la souris
mouseup() // au moment où l'on relâche le clic de la souris
focus() // lorsque l'on clic sur un champ input
blur() // lorque l'on clic en dehors d'un champ input
```

Lien recensant tous les événements JQuery :

http://www.w3schools.com/jquery/jquery_ref_events.asp

Exemple d'utilisation d'un événement JQuery :

```
<h1>Mon Titre</h1>
<button>Clic sur moi</button>

<script>
$("button").click(
	function(){$("h1").text("Nouveau titre"); } // au clic sur le boutton, je change mon titre H1.
) ;
</script>
```

LES FONCTIONS JQUERY

La bibliothèque JQuery nous offre l'accès à plusieurs fonctions qui permettent d'animer notre site web, d'accéder et modifier le CSS directement au moment d'un événement et bien d'autres choses encore …

```
$("h3").hide(); // fait disparaitre tous mes éléments H3
$("h3").show() ; // fait apparaitre tous mes éléments H3
$("#button").click(function(){$("h3").toggle();}); // à chaque clic sur mon bouton, la fonction .toggle() fait disparaitre et apparaitre tous mes éléments H3
$("h3").show('slow') ; // l'agument 'slow' permet de faire disparaitre lentement. 'fast' pour rapidement et '100' pour un temps de 100 millisecondes (tous les autres nombres seront considérés comme des millisecondes)
```

Les fonctions qui touchent au CSS :

```
$("a").css("display","block"); // pour écrire une propriété CSS
$("label").css({"float":"left",
              "font-style":"italic",
              "width":"100px"
              }); // pour écrire plusieurs propriétés CSS
$("h1").addClass("rouge"); // pour ajouter une classe CSS à un élément
$("h1").removeClass("rouge"); // pour retirer une classe CSS à un élément
$("button").click(function(){
              $("h1").toggleClass("rouge");
                }); // permet de retirer et ajouter une classe CSS à chaque clic sur le bouton
```

Lien recensant toutes les fonctions JQuery :

http://www.w3schools.com/jquery/jquery_ref_effects.asp

➢ Les Plugins JQuery

A travers le Web, il existe des centaines voire des milliers de plugins JQuery, adaptables à notre site et nous permettant de l'animer en fonction de ce que propose le plugin.
Parallax, Slider, Zoombox, Flipbook, Carousel … Derrière tous ces titres tendances du Web actuel se cache un plugin ; qu'il soit gratuit ou payant …

Le plugin est un gros bloc de code développé par un ou plusieurs développeurs Javascript – Jquery, qui a été travaillé pour être adaptable, paramétrable et accessible à n'importe quelle personne souhaitant l'utiliser sur son site et ne connaissant rien (ou très peu) à la programmation.

Comme je le disais, il existe des milliers de plugin et n'importe quel développeur peut rendre disponible son plugin sur la toile, c'est pourquoi il est assez difficile de choisir le bon plugin, assez professionnel pour pouvoir travailler avec.

DOCUMENTATION

Le point commun à tous les plugins [sérieux …] est la documentation. Un plugin fournit sans documentation donc sans notice, est d'ores et déjà disqualifié.
Soit celle-ci se trouve sur le site web du développeur, soit dans un fichier HTML à l'intérieur du dossier du plugin.

How to use

Create a wowBook in your site requires 4 steps.

1 - Find the directory 'wow_book' that comes in the zip file you downloaded from CodeCanyon. Upload this directory to your webserver document root.

2 - Include JQuery 1.7.1, the wowBook script and the wowBook CSS on your page:

```
<script type="text/javascript" src="./js/jquery-1.7.1.min.js"></script>
<script type="text/javascript" src="./wow_book/wow_book.min.js"></script>
<link rel="stylesheet" href="./wow_book/wow_book.css" type="text/css" />
```

3 - Create the book content using html. Create a div to hold the book pages. Inside that div, create one div for each page of the book.

La documentation explique à l'utilisateur comment installer et paramétrer sont plugin. Elle regroupe toutes les informations permettant d'utiliser les options de celui-ci sans avoir à manipuler les algorithmes qui ont permis de développer le code de ce plugin.

INSTALLATION

Tous les plugins s'installent de la même manière.

Il s'agit d'un fichier .JS que l'on appel dans son <head> exactement comme on le fait avec la bibliothèque JQuery ou notre fichier CSS.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xh
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <title>Plugin</title>
    <script type="text/javascript" src="js/jquery-1.7.1.min.js"></script>

    <script type="text/javascript" src="wow_book/wow_book.min.js"></script>

    <link rel="stylesheet" href="wow_book/wow_book.css" type="text/css" />
    <link rel="stylesheet" type="text/css" href="css/style.css"></head>
<body >
    <header>
```

90% des plugins une fois téléchargés sont accompagnés de fichiers HTML pour la démonstration et l'utilisation de celui-ci.

Ces fichiers qui servent d'exemples sont un excellent point de départ pour comprendre comment intégrer notre site avec ce plugin.

UTILISATION ET PARAMETRAGE

Très souvent les plugins sont accompagnés de fichiers CSS propres à leur utilisation qu'il faut aussi intégrer. Ce sont des styles généralement sobres mais qui permettent leur intégration sans avoir de surprise liée au CSS (exemple : une div qui dépasse ou qui décale vos propres div)
Le code qui gère le plugin JQuery se trouve souvent dans les balises <script> du document HTML. Généralement celui-ci est un objet qui regroupe plusieurs propriétés et méthodes qui permettent d'agir sur le plugin lorsqu'on en a modifié les valeurs. Ce sont ces propriétés et méthodes que l'on retrouve dans la documentation.

Voici un exemple montrant des propriétés et des méthodes dans le code JQuery, permettant de modifier les paramètres de notre plugin (vitesse d'animation, nombre de slides etc …)

```
<script>
    $(document).ready(function() {
        $('#book').wowBook({
             width  : 800 // largeur de l'album, modifier pour montrer ce qui change
            ,height : 540 // hauteur de l'album
            ,centeredWhenClosed : true // position de l'album centré à la fermeture (false si
            ,hardcovers : true // couvertures de l'album rigides (false si effet feuille soupl
            ,numberedPages   : [2,-4] // LES PAGES A NUMEROTER. Où ça commence, et où ça finit
            DES PAGES
            ,turnPageDuration : 1000 // VITESSE DURANT LE CHANGEMENT DE PAGE EN MILLISECONDES
            ,pageNumbers : true // PERMET DE DESACTIVER LA NUMEROTATION DES PAGES
            ,slideShowDelay : 1000 // DELAI DU SLIDESHOW AUTOMATIQUE ENTRE CHAQUE PAGE
            ,pauseOnHover : true // MET PAUSE AU SLIDESHOW QUAND ON PASSE LA SOURIS DESSUS
            ,controls : {
                 back : '#back' // le bouton à cibler qui correspond à page-précedente (voir l
                ,next : '#next' // le bouton à cibler qui correspond à page-suivante (voir lig
                ,zoomIn : '#plus' // PROPRIETE A AJOUTER DANS LA VERSION ETUDIANT POUR ACTIVER
                ,zoomOut : '#moins' // PROPRIETE A AJOUTER DANS LA VERSION ETUDIANT POUR ACTIV
                ,slideShow :'#lecture' // NE PAS PRESENTER CAR BOUTON ABSENT // PROPRIETE A AJ
                SLIDESHOW AUTOMATIQUE ATTENTION, D'ABORD CREER LE BOUTTON DANS LE HTML
            }
```

Une bonne recherche sur un moteur de recherche avec les bons mots clés permet de retrouver le plugin qui répond à ses besoins.

Attention toutefois à ne pas tomber sur les sites pièges qui proposent de télécharger des fichiers.exe pour injecter un virus.

Voici un site connu proposant une liste de très bon plugins mais cependant payants :
http://codecanyon.net/

➢ Divers

LES SHIM ET LES POLYFILL

Au fil du temps, les applications évoluent et les codent de programmation qui les exploitent évoluent aussi. C'est le cas des navigateurs web comme n'importe quelles applications informatiques.
Pour assurer cette compatibilité à travers le temps. Les développeurs ont eu l'idée de créer des plugins générant un code « ancienne version » permettant de reproduire l'effet désiré de la « nouvelle version ».
Ces plugins s'appellent des SHIM.

Un Shim est une bibliothèque qui permet l'exécution d'un nouveau code dans un ancien environnement. En clair, on écrit le code dans la langue de l'ancien environnement et cela donne le même effet que s'il avait été implanté dans un nouvel environnement.

Prenons l'exemple des Polyfill, ce sont des plugins JS ou Flash qui permettent d'assurer l'utilisation de fonctions des nouveaux navigateurs sur les anciens.
Exemple : la balise <audio> ou <video> appartient au HTML5 et ne fonctionne pas sur les anciens navigateurs comme IE6. En installant un code JS (Polyfill) correspondant aux balises <audio> et <video> dans notre script, on pourra écrire <audio> ou <video> et ces balises seront compatibles tout navigateur même IE6.

Polyfill = Shim. La seule différence est qu'on utilise le mot "Polyfill" quand on pratique cette technique dans le web. Le mot "Shim" lui, est utilisé dans tout l'univers informatique.

DEGRADATION PROGRESSIVE

La dégradation progressive ou « graceful degradation » est une méthode d'intégration web qui consiste à "accepter" le fait que le rendu ne sera pas le même sur les anciens navigateurs par rapport au nouveaux.
Typiquement, on va intégrer tout notre site web en se basant sur les nouveaux navigateurs puis, si le rendu n'est pas le même sur l'ancien navigateur, on s'assure simplement que c'est fonctionnel même si le visuel n'est pas le même.
Exemple : Je mets un border radius CSS3 pour que mes <input> soient plus esthétiques.
Je me rends compte que l'effet arrondi ne passe pas sur IE7 mais qu'il passe bien sous Firefox, Chrome et IE9.

Et bien tant pis, je m'assure simplement que mon formulaire fonctionne correctement sous IE7 même si mes <input> sont carrés.

Cette technique d'intégration est plus rendable en terme de temps que celle dite du « pixel perfect », qui consiste à reproduire un site web pour chaque navigateur au pixel près et ainsi s'assurer que le rendu sera strictement le même quelque soit la version du navigateur.

** Ce support « JavaScript/Jquery » constitue l'introduction au JavaScript/Jquery pour être en adéquation avec le module de cours dispensé. D'autres supports de cours sont disponibles et approfondissent d'avantage le langage ainsi que ses possibilités.*