Web Avancé - Séance 1
Javascript

Sommaire

0. Installation d'un environnement de développement

Télécharger la dernière version d'Eclipse (Kepler) : Eclipse Classic 4.3.x sur http://www.eclipse.org/downloads/

Dézipper le tout dans votre répertoire local personnel se trouvant dans le répertoire /gfs.

Lancer eclipse, choisissez un nouveau workspace dédié à Javascript

Dans le menu Window > Preferences, la rubrique General > Network, veuillez passer en mode "manual" pour l'active provider et mettre "cache-etu.univ-lille1.fr" & 3128 pour HTTP et HTTPS.

Dans le menu Help > Install new software

Ajouter une "location" : http://download.eclipse.org/releases/kepler/

Puis installer les 2 plug-ins suivants (utiliser le filtre pour trouver plus facilement).

- Web, XML, Java EE... > Javascript Development Tools
- Web page editor

Voilà vous êtes prets !

1. Utilisation de JavaScript dans une page Web

Nous allons travailler en XHTML. Et ça tombe bien car la création d'une page HTML dans notre environnement correspond à ce type de page :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Insert title here</title>
</head>
<body>

</body>
</html>
```

Tout code code javascript doit se trouver dans une balise <script>.

```
<script type="text/javascript">
alert("bonjour");
</script>
```

Le code peut se trouver au sein de la balise `<script>` ou bien dans un fichier référencé alors par l'attribut `src`. Il est important d'avoir une balise `script` ouvrante et une balise `script` fermante.

```
<script src="autreFichier.js"></script>
```

L'extension *js* n'est pas obligatoire, c'est une convention.
Vous pouvez mettre une url.

2. Quatre fonctions à connaître pour commencer

Un avertissement

```
alert("message");
```

La fonction `alert` affiche un pop-up avec le message en paramètre (optionel).

Une boite de dialogue

```
valeur=prompt("Invitation","valeur de base");
```

La fonction `prompt` affiche un pop-up avec un message (premier paramètre, optionel) et un champ de texte afin d'y saisir une valeur. Une valeur par défaut peut être indiquer en second paramètre.

L'écriture dans la page HTML

```
<script type="text/javascript">
document.write("bonjour");
</script>
```

La fonction `write` de l'object HTML `document` permet d'écrire directement dans la page Web courante. Une page web peut ainsi être écrite totalement en javascript.

L'écriture dans la console javascript

```
console.log("bonjour");
```

La fonction `log` de l'object `console` permet d'écrire du texte dans la console Javascript du navigateur.

Exo 2-1 : Écrire une page html index2-1.html (sans fichier annexe js) qui affiche "bonjour à tous" où "bonjour à" est écrit via la fonction document.write.

Exo 2-2 : Écrire une page html index2-2.html (sans fichier annexe js) qui prévient l'internaute qu'il va écrire "bonjour", qui affiche "bonjour" (toujours via document.write) et qui affiche dans la console le fait qu'il a écrit bonjour.

Exo 2-3 : Écrire une page html index2-3.html (sans fichier annexe js) qui, en une phrase, demande à l'internaute ce qu'il doit écrire et l'affiche ensutie.

Exo 2-4 : Écrire une page html index2-4.html identique à index2-1.html … mais cette fois-ci le code se trouve en source2-4.js.

3. Les variables

Le typage est dynamique en javascript, donc pas de typage statique comme en Java. La seule chose à retenir est la création d'une variable locale ou globale.

Variable locale : utilisation du mot-clé `var`

Une variable locale a une portée qui se limite à la zone où elle a été définie. Si elle est définie en dehors d'une fonction ou d'un objet, elle est globale. Dans ce cas, elle peut être définie dans un fichier et utilisée dans un autre fichier (si celui-ci est interprété après le fichier où est définie la variable).

var x=3;

Variable globale : aucun mot clé devant l'affectation

Une variable est accessible à n'importe quel endroit. L'inconvénient d'une définition globale sans mot-clé c'est que si on pense avoir défini une variable x comme locale et que ce n'est pas le cas, x devient global et peut influencer le reste du code involontairement. Penser à une variable i qu'on ne définit pas comme globale -> les boucles seront pertubées.

Vous pouvez créer dynamiquement une variable globale (cad que vous ne connaissez pas le nom de la variable au moment du codage) en affectant un attribut - avec le nom souhaité - à l'objet HTML `Console`.

Ex :

```
nomVariable=prompt("Nom de la variable globale");
window[nomVariable]=2
```

Si à l'exécution vous répondez v1, vous aurez une variable globale v1 égale à 2.

Les opérateurs `++`, `--`, `-=` ou `+=` sont aussi disponibles en javascript.

Exo 3-1 : Écrire une page html index3-1.html (sans fichier annexe js) qui reprend l'objectif de l'exo 2-3, mais en 2 lignes cette fois : une qui stocke dans la variable `texteAEcrire` ce que souhaite afficher l'internaute et la deuxième ligne pour l'affichage.

Exo 3-2 : Écrire une page html index3-2.html (sans fichier annexe js) identique à l'exo 3-1 mais la création de la variable `texteAEcrire` se fait par l'objet window.

4. Les boucles

Le mot clé for permet de définir une boucle. Sa syntaxe est proche de java :

```
for (var i=0;i<10;i++) {
  ...
}
```

N'oubliez pas le mot-clé `var` afin d'éviter des erreurs par la suite.

Exo 4-1 : Écrire une page html index4-1.html avec le code JS dans source4-1.js qui écrit, via document.write et les balises table-tr-td, la table de multiplication (1 à 10) sans les entêtes.

Le tableau doit avoir :

- un trait de contour de 1 de large (attribut border)
- pas d'espacement entre les cellules (attribut cellspacing)
- une marge intérieur de 5 (attribut cellpadding)

Les cellules (td) n'ont pas d'attribut particulier.

Exo 4-2 : Écrire un couple index4-2.html, source4-2.js identique à l'exo 4-1 mais en utilisant l'attribut style (style = "libelle1 : valeur1; libelle2 :valeur2; …") pour les balises table et td.

le tableau :

- la bordure (border : "couleur largeur style") de couleur #000088, d'1px de large, et en pointillé.
- le contour des cellules (left-right-top-bottom > largeur style couleur) tout d'1px de large en trait plein :
  * bordure haute (border-top) rouge
  * bordure basse (border-bottom) verte
  * bordure gauche (border-left) bleue
  * bordure droite (border-right) grise

Note : N'utilisez pas encore de feuilles de style.

5. Les types de valeurs

Les types de valeurs sont les suivants :

- les nombres (`number`)
- les chaînes de caractères (`string`)
- les booléens (`boolean`)
- les fonctions (`function`)
- les objets (`Object`)
- les tableaux (`Array`)

Pour connaître le type d'une valeur, il faut utiliser la fonction `typeof(valeur)` => renvoie la chaîne de caractères indiquée plus haut entre parenthèses.

`typeof(variable)` renvoie "undefined" si la variable n'existe pas.

6. Les nombres et leurs fonctions

Un nombre peut être un entier (ex : 3), un flottant (ex : 3.13) ou hexadécimal (ex : Oxff).

Version objet

On peut utiliser les nombres comme suit :

```
var a = new Number(3);
```

Dans ce cas, pour utiliser la valeur, il faut passer par valueOf(), car a est un objet :

```
var b=2+a.valueOf();
```

Quelques fonctions utiles :

- La fonction `isNaN(valeur)` permet de savoir si `valeur` n'est pas un nombre.
- La fonction `isFinite (flottant)` permet de savoir si `flottant` est fini.
- La fonction `parseFloat (chaine)` renvoie la valeur flottante codée dans la `chaine` si c'est possible ou NaN sinon.
- La fonction `parseInt (chaine)` renvoie la valeur entière codée dans la `chaine` si c'est possible ou NaN sinon.

Les classiques fonctions mathématiques :

- abs() : Retourne la valeur absolue d'un nombre réel.
- acos() : Retourne l'arccosinus d'un nombre réel.
- asin() : Retourne l'arcsinus d'un nombre réel.
- atan() : Retourne l'arctangente d'un nombre réel.
- ceil() : Retourne l'entier le plus proche, supérieur ou égal au réel passé en paramètre.
- cos() : Retourne le cosinus d'un nombre réel.
- exp() : Retourne l'exponentielle du réel donné en paramètre.
- floor() : Retourne l'entier inférieur ou égal le plus proche du paramètre.
- log() : Retourne le logarithme népérien (ln) du réel passé en paramètre.
- max() : Retourne le maximum parmi deux nombres.
- min() : Retourne le minimum parmi deux nombres.
- pow() : Retourne x à la puissance y (paramètres : x et y).
- random() : Retourne un nombre aléatoire entre 0 et 1.
- round() : Retourne l'arrondi (entier) du réel passé en paramètre.
- sin() : Retourne le sinus d'un nombre réel.
- sqrt() : Retourne la racine carrée.
- tan() : Retourne la tangente d'un nombre réel.

Exo 6-1 : Reparter de l'exo 4-2 pour écrire une page html (index6-1.html,source6-1.js) qui demandera à l'internaute un multiplicateur. Tant que le texte rentré ne sera pas un nombre, la page redemandera le multiplicateur (`do { } while ();`). Ensuite, la page affichera la table de multiplication en multipliant chaque case par la partie entière du nombre rentrée.

7. Les chaînes de caractères

On peut créer une chaîne de caractères de 2 façons :

```
var s1="bonjour";
var s2=new String("salut");
```

Il n'est pas nécessaire de passer par `valueOf()` pour utiliser la "vraie" chaîne de caractères.

Pour la concaténation, c'est l'opérateur + qu'on utilise comme en Java.

Une chaîne est un objet de type String. Voici les méthodes associées au type String :

- anchor() : Retourne le texte en créant une ancre.
- big() : Retourne le texte en police plus grosse.
- blink() : Retourne le texte en mode clignotant.
- bold() : Retourne le texte en gras.
- charAt() : Retourne le caractère à la position spécifiée.
- charCodeAt() : Retourne le code ASCII d'un caractère.
- concat() : Retourne la concaténation de 2 chaînes.
- fixed() : Retourne le texte en police à chasse fixe.
- fontcolor() : Retourne le texte colorié.
- fontsize() : Retourne le texte dimensionné.
- fromCharCode() : Crée une chaîne à partir d'une série de code ASCII.
- indexOf() : Retourne la position d'une sous-chaîne.
- italics() : Retourne le texte en italique.
- lastIndexOf() : Retourne l'indice de la dernière occurrence de sous-chaîne.
- link() : Retourne le texte en créant un lien.
- match() : Vérifie la concordance d'un motif d'expression régulière.
- replace() : Remplace un motif d'expression régulière.
- slice(debut, fin) : Extrait une sous-chaîne.
- small() : Retourne le texte en police plus petite.
- split(motif) : Retourne un tableau de chaînes découpées par un séparateur.
- strike() : Retourne le texte barré.
- sub() : Retourne le texte en indice.
- substr(debut, longueur) : Extrait une sous-chaîne d'une chaîne.
- substring(debut, fin) : Extrait une sous-chaîne d'une chaîne.
- sup() : Retourne le texte en exposant.
- toLowerCase() : Retourne le texte en minuscules.
- toUpperCase() : Retourne la chaîne en majuscules.

Exo 7-1 : Reparter de l'exo 6-1 pour écrire une page html (index7-1.html,source7-1.js) qui met en gras les multiples de 10 et barre les nombres supérieurs à 100. Ceci en utilisant les fonctions vues précédemment.

8. Les booléens

On affecte une valeur booléenne à une variable soit :

- directement

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

- indirectement (par une expression ou une fonction)

```
a = (c==5);
```

Encore une fois, on peut utiliser une version objet, mais en utilisant valueOf() pour les calculs entre booléens.

```
a = new Boolean(true);
t = a.valueOf() && b;
```

9. Les tableaux

Les tableaux en javascript se définissent comme suit :

```
var tab = new Array(3, "bonjour", true);
ou
var tab = [ 3, "bonjour", true ];
```

`tab[0]` >> 3

`tab[1]` >> "bonjour"

`tab[2]` >> true

On connaît la longueur via l'attribut `length` du tableau.

`tab.length` >> 3

Voici les méthodes d'un tableau :

Méthodes

- concat(2ème tableau) : Retourne la concaténation de 2 tableaux.
  - `tab.concat(t2)` >> t1+t2
- join() : Convertit un tableau en chaîne.
  - `tab.join(",")` >> "3, bonjour, true"
- pop() : Supprime le dernier élément du tableau.
  - `tab.pop()` >> true et tab vaut {3, "bonjour"}
- push(valeur) : Ajoute des éléments en fin de tableau.
  - `tab.push ("hello")` >> tab vaut {3, "bonjour", true, "hello" }
- reverse() : Inverse l'ordre des éléments.
- shift() : Supprime le premier élément du tableau.
- slice(début, fin) : Retourne une tranche de tableau.
  - `tab.slice(1,2)` >> {"bonjour", true}
- sort() : Trie les éléments d'un tableau.
- splice(début, nb d'éléments) : Ecrase une tranche de tableau.
  - `tab.splice (1,1)` >> {3, true}
- unshift(valeurs) : Insère des éléments en début de tableau.
  - `tab.unshift (1, "truc")` >> {1, "truc", 3, "bonjour", true}

Exo 9-1 : Reparter de l'exo 7-1 pour écrire une page html (index9-1.html,source9-1.js) qui demande à l'internaute les intitulés des colonnes et des lignes comme suit :

- Colonnes : entrer une série de nombres séparés par des virgules

- Lignes : entrer une série de nombres séparés par des virgules

Avec comme valeur par défaut "2,4,6,8,10,12,19" pour les colonnes et "3,6,9,12,15,18,21" pour les lignes.

Ces 2 listes stockées dans les variables `intitules_colonnes` et `intitules_lignes` sont ensuite traduites en tableau dans les variables colonnes et lignes pour être affichées comme entête du tableau (pas d'utilisation de la balise TH).

Pour l'affichage de la première ligne (entête des colonnes), n'utilisez pas de boucle mais la fonction `join`.

Le style des entêtes doit être le suivant : `<td style='background-color : #00AAAA'>`

Mettre en gras les multiples de 10 et barrer les nombres supérieurs à 100. Ceci en utilisant les fonctions vues précédemment.

Exo 9-2 : Reparter de l'exo 9-1 pour écrire une page html (index9-2.html,source9-2.js) qui fait la même chose mais avec une feuille de style (OUF !) :

Au sein de la balise HEAD d'index9-2.html

```
<LINK rel="stylesheet" type="text/css" href="style.css">
```

La feuille de style (fichier style.css) contient

```
@CHARSET "UTF-8";

TABLE.multiplication {
	border: #000088 1px dotted;
	border-collapse: separated
}

TD.intitule {
	background-color: #00AAAA
}

TD.valeur {
	border-top: 1px solid #880000;
	border-bottom: solid 1px #008800;
	border-left: solid 1px #000088;
	border-right: solid 1px #888888
}
```

L'utilisation d'une classe CSS se fait comme suit (exemple) : `<td class='label'>`

10. Les fonctions

Définir une fonction passe la syntaxe suivante :

```
function nomFonction (arg1, arg2, ...) {
  ...
}
```

C'est le mot clé `return` qui est utilisé pour retourner une valeur.

Types :

Le type des arguments est bien entendu absent.

Le type de retour est absent aussi.

Arguments :

On peut accéder aux arguments par la variable locale implicite `arguments` de la fonction. C'est un objet qui se comporte comme un tableau (`length` présent & accès par indice).

On peut mettre plus d'arguments à une fonction lorsqu'on l'invoque que le nombre indiqué dans sa définition. On peut mettre moins d'arguments qu'indiqué dans la définition (les arguments restants seront nulles).

Ex :

```
function addition () {
  var resultat = 0;
  for (var i=0;i<arguments.length;i++)
    if (typeof(arguments[i]) == "number")
      resultat+=arguments[i];
  return resultat;
}
```

Si un argument x n'est pas donné lors d'une invocation, x reste une variable locale que l'on peut utiliser au sein de la fonction. Par exemple,

```
fonction f1 (x) {
    if (typeof(x) == "undefined")
          x=2; // cela ne crée pas une variable globale
    ...
}
```

Exo 10-1 : Reparter de l'exo 9-2 pour écrire une page html (index10-1.html,source10-1.js) qui fait la même chose mais en utilisant des fonctions et en gagnant ainsi des points en terme de réutilisabilité.

texteSimple(texte) { return new Array(texte,null); }

function intitule(texte) { return new Array(texte,"intitule"); }

function valeur(texte)

- la fonction genererTableauMultiplication (xs, ys) qui prend en paramètres 2 tableaux correspondant aux entêtes/valeurs pour les multiplications. Cette fonction génère un tableau de valeurs (pas le tableau html) correspondant aux multiplications (x*y). Chaque valeur est en fait un tableau de 2 valeurs : la première est le texte à afficher et la deuxième est la classe CSS à utiliser en cas d'affichage.
  Cette fonction contient 3 sous fonctions
  - texteSimple(texte) qui renvoie un tableau (texte, classeCSS) où la classeCSS est nulle.
  - intitule(texte) > idem mais la classeCSS vaut "intitule"
  - valeur(texte) > idem mais la classeCSS vaut "valeur"
- la fonction ecrireTableau (tableauValeursClasses, classeCssTableau) qui écrit, via document.write, un tableau HTML avec les valeurs contenues le paramètre tableauValeursClasses et avec la classe CSS qui y est indiqué > chaque élément de tableauValeursClasses est un tableau à 2 éléments : le texte et la classe CSS.
  Cette fonction aura les sous-fonctions suivantes :
  - recupValeur(tableau) > où tableau est de type [valeur, classeCSS] et où la valeur renvoyée sera valeur
  - recupClasse(tableau) > idem mais valeur renvoyée est classeCSS
  - ecrireDebutBalise (nomBalise,classeCSS) > écrit via document.write l'expression suivante "<nomBalise class='classeCSS'>"
  - function ecrireFinBalise (nomBalise) > écrit "</nomBalise>"
  - function ecrireBalise (nomBalise,classeCSS, contenu) : utilise les 2 précédentes versions et écrit contenu entre la balise ouvrante et la balise fermante.

11. Les objets

Des dictionnaires

Les objets javascript sont d'abord des dictionnaires.

```
var client = { nom  : "Le Pallec", prenom : "Xavier" , age : 35 };
```

On accède aux éléments de l'objet comme suit :

```
client.nom >> "Le Pallec"
client["nom"] >> "Le Pallec"
```

Avec une boucle on dispose du for i in :

```
for (var i in client) {
    document.write(i+":"+client[i]+"<BR/>");
}
```

On peut aussi créer un objet comme suit :

```
var client = new Object();
client.nom = "Le Pallec";
client.prenom = "Xavier";
client["age"] = 35;
```

Des fonctions dans un objet

Il est possible d'avoir des fonctions comme valeur d'une propriété. Le mot clé this permettra d'appeler, au sein de ces fonctions, les autres propriétés de l'objet sur lequel est invoquée la fonction.

```
var couple = {
      x : 3,
      y : 4,
      xPlusY : function () { return this.x + this.y ; }
}
couple.y=5;
couple.xPlusY () > 8
```

La suite sur les objets en séance 2...

Exo 11-1 : Réécrivez l'exo 10-1 (index11-1.html,source11-1.js) avec les 2 objets

`ecriteurTableau` et `generateurTableauMultiplication` dont la structure est :

```
ecriteurTableau = {
          // fonction principale
      ecrire : function(tableauValeursClasses, classeCssTableau)... ,

       ecrireDebutBalise : function(nomBalise, classeCSS) ... ,
       ecrireFinBalise : function(nomBalise) ... ,
       ecrireBalise : function(nomBalise, classeCSS, contenu) ...
};

generateurTableauMultiplication = {
       generer : function(xs, ys) ... , // fonction principale
       texteSimple : function(texte) ... , // voir note plus bas
       intitule : function(texte) ... ,// voir note plus bas
       valeur : function(texte) ... // voir note plus bas
};

var intitules_colonnes, intitules_lignes;
intitules_colonnes = prompt(
             "Colonnes : entrer une série de nombres séparés par des virgules",
             "2,4,6,8,10,12,19").split(",");
intitules_lignes = prompt(
             "Lignes : entrer une série de nombres séparés par des virgules",
             "3,6,9,12,15,18,21").split(",");
ecriteurTableau.ecrire( generateurTableauMultiplication.generer(intitules_colonnes,
intitules_lignes), "multiplication");
```

A noter : les éléments du tableau de valeurs n'est plus de type `[ valeur , classeCSS ]` mais de type `{ valeur : valeur, classeCSS : classeCSS }`.

12. Les objets HTML d'une page web

Des noeuds XML

Tous les éléments d'une page web sont manipulables en javascript sous forme d'objets.

Tout objet est d'abord un noeud XML. Pour cette raison, il dispose des propriétés et méthodes suivantes :

Propriétés

- attributes : attributs.
- childNodes : nœud enfant.
- data : données en caractères.
- firstChild : premier nœud enfant.
- lastChild : dernier nœud enfant.
- nextSibling : prochain nœud d'un type.
- nodeName : nom du nœud.
- nodeType : type du nœud.
- nodeValue : valeur/contenu du nœud.
- parentNode : nœud parent.
- previousSibling : nœud précédent d'un type.

Méthodes

- appendChild(noeud) : ajouter un nœud enfant.
- cloneNode() : copier un nœud.
- getAttribute(nomAttribut) : rechercher la valeur d'un nœud attribut.
- hasChildNodes() : vérifier l'existence de nœuds enfants.
- insertBefore(noeudAprès) : insérer un nœud.
- removeAttribute() : effacer la valeur d'un nœud attribut.
- removeAttributeNode() : effacer un nœud attribut.
- removeChild(noeudASupprimer) : effacer un nœud.
- replaceChild(nouveauNoeudFils, ancienNoeudFils) : remplacer un nœud enfant.
- setAttribute(nom, valeur) : fixer la valeur d'un nœud attribut.

Si on reprend quelques méthodes de l'objet `document` :

- createAttribute(nom, valeur) : créer un nœud d'attributs.
- createElement(nomBalise) : créer un nœud d'éléments.
- createTextNode() : créer un nœud de texte.
- getElementById(IdATrouver) : Accès à l'élément HTML par l'attribut Id.
- getElementsByName(nom) : Accès aux éléments HTML par l'attribut nom.
- getElementsByTagName(nomBalise) : Accès aux éléments HTML par liste d'éléments.

On retrouve le fonctionnement habituel de manipulation de noeud XML avec une fabrique qui est le document.

Référence : http://fr.selfhtml.org/javascript/objets/index.htm

Ajout de propriétés Javascript à une balise HTML

```
a= document.getElementById("DD");
a.monAttribut=12;
b=document.getElementById("DD);
b.monAttribut >> 12 !
```

Exo 12-1 : Réécrivez l'exo 11-1 (index12-1.html,source12-1.js) mais cette fois-ci, à la place de document.write, utiliser l'API DOM de Javascript pour construire le tableau (enfin !) que vous ajouterez à la balise body.

Vous pouvez utiliser la structure suivante :

en utilisant avec les 2 objets `ecriteurTableau` et `generateurTableauMultiplication` dont la structure est :

```
html = {

	getBody : function() ..., // renvoie la balise body du document

	ajouterBalise : function(baliseMere, nomBalise) ...,
	// ajouter une balise nomBalise à la balise baliseMere
	// il est possible de passer des attributs et leur valeur en paramètres à l'infini
	// ex : ajouterBalise (document.getElementById("i0"),
		"P","align","center","valign","middle");

	creerBalise : function(nomBalise) ...,
	// créer et renvoie une balise nomBalise
	// il est possible de passer des attributs et leur valeur en paramètres à l'infini
	// ex : ajouterBalise (document.getElementById("i0"),
		"P","align","center","valign","middle");

	ajouterTexte : function(balise, texte) ...,
	// ajoute (et renvoie) une balise textuelle à balise
	// ex : ajouterBalise (document.getElementById("i0"),"bonjour");
};

table = {
	creer : function(tableauValeursClasses, classeCssTableau) ...
	// comme ecriteurTableau.ecrire mais via DOM et
	// renvoie la balise table correspondante
};
```

```
generateurTableauMultiplication = ... / ne change pas


/*
Pourquoi une fonction go ? il faut attendre que la page soit chargée pour que la balise body soit atteignable
Il faut donc ajouter attribut onload="go()" à la balise body
*/
function go() {
	var intitules_colonnes, intitules_lignes;
	intitules_colonnes = prompt(
			"Colonnes : entrer une série de nombres séparés par des virgules",
			"2,4,6,8,10,12,19").split(",");
	intitules_lignes = prompt(
			"Lignes : entrer une série de nombres séparés par des virgules",
			"3,6,9,12,15,18,21").split(",");
	html.getBody().appendChild(
			table.creer(generateurTableauMultiplication.generer(
					intitules_colonnes, intitules_lignes),
"multiplication"));
}
```

13. Les événements sur les éléments HTML

l y a 4 façons de gérer les événements sur les éléments HTML.

Dans le code HTML

```
<BUTTON ID="ZZ" onclick="maFonctionJavaScript("ZZ");">
```

On indique ici que quand on clique sur le bouton dont l'identifiant est ZZ, alors on exécute le code javascript `maFonctionJavaScript("ZZ")`.

C'est facile, mais difficilement exploitable avec des objets javascript qui contiennent les données de votre application.

Associer une fonction en javascript

On peut utiliser la propriété javascript correspondante à l'événement.
Sur l'exemple précédent :

```
document.getElementById("ZZ").onclick = maFonctionJavaScript;
```

Le problème est qu'on ne peut pas indiquer des paramètres particulier.

Il faut savoir que le premier argument sera obligatoirement un objet de type event, dont les propriétés sont :

- altKey, ctrlKey, shiftKey : touches particulières/Microsoft.
- clientX, clientY : coordonnées écran/Microsoft.
- keyCode : code clavier/Microsoft.
- layerX, layerY : coordonnées relatives à l'objet/Netscape.
- modifiers : touches particulières/Netscape.
- offsetX, offsetY : coordonnées relatives à l'objet/Microsoft.
- pageX, pageY : coordonnées relatives à la fenêtre/Netscape.
- screenX, screenY : coordonnées écran/Netscape.
- which : code clavier et souris/Netscape.
- target : l'objet HTML concerné par l'événement.
- currentTarget : l'objet HTML qui traite actuellement l'événement (fil de propagation).
- type : type de l'événement/Netscape.
- x,y : coordonnées relatives à l'élément parent/Microsoft.

La méthode stopPropagation permet d'arrêter la propagation de l'événement.

La liste des événements/attributs possibles pour les objets HTML sont : onclick, ondblclick, onkeydown, onkeypress, onkeyup, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup.

Si un this est utilisé dans la fonction invoquée, il fait référence à l'objet HTML sur lequel l'événement est intervenu.

Ajouter une liste de fonctions auditrices

Le problème de la solution précédente c'est qu'il n'y a qu'une fonction auditrice. Pour pallier à cela, on utilise la fonction addEventListener et removeEventListener sur l'objet HTML (cf. attachEvent pour IE).

objetHTML.addEventListener (type, fonction, propogation) :

- type = abort, blur, change, click, dblclick, dragdrop, error, focus, keydown, keypress, keyup, load, mousedown, mousemove, mouseout, mouseover, mouseup, move, reset, resize, select, submit, unload.
- fonction = fonction invoqué lors de l'événement
- propagation = propagé ou non l'événement ensuite

Ex :

```
document.getElementById("ZZ").addEventListener("click",maFonctionJavaScript,fal
se);
...
document.getElementById("ZZ").removeEventListener("click",maFonctionJavaScript,
false);
```

Ajouter des objets auditeurs

L'avantage d'ajouter directement un objet en tant qu'auditeur est d'avoir un contexte, ce qu'on n'a pas avec une simple fonction.

Avec Firefox, on utilise la même fonction que précédemment

```
objetHTML.addEventListener ( type, objet, propogation )
```

Sauf qu'on y passe un objet qui dispose d'une fonction `handleEvent` qui peut avoir un argument de type event (avec les propriétés altKey, keyCode...). Ceci se rapproche du fonctionnement Java/AWT/Swing.

Le paramètre `propagation` indique si l'auditeur doit être appelé plutôt que l'auditeur d'une de ces balises mère/grand-mère/etc.. (propagation = false). Propagation à true indique que si la balise à une balise mère/gd-m/etc.. avec un auditeur, celui-ci sera préféré.

`removeEventListener` (type, objet) est l'opposé d'`addEventListener`.

Exemple :

Avec peu de réutilisation

```
objetHTML.addEventListener (
      "click",
      { handleEvent : function (event) {
            alert("ne touche pas à la cellule n°"+this.numeroCellule);
        },
        numeroCellule : 3

      },
      propogation );
```

Avec un peu plus de réutilisation

```
function monAlerte(event) {
            alert("ne touche pas à la cellule n°"+this.numeroCellule);
            }
objetHTML.addEventListener (
      "click",
      { handleEvent : monAlerte,
        numeroCellule : 3
      },
      propogation );
```

Exo 13-1 : Réécrivez l'exo 12-1 (index13-1.html,source13-1.js) pour qu'un click sur une cellule demande à l'utilisateur quelle nouvelle valeur il souhaite y mettre. La fenêtre de dialogue indique quelle est la coordonnée de la cellule. L'auditeur du click doit être sur la cellule. Il y en aura un par cellule.

Exo 13-2: Comme le 13-1 mais (index13-2.html,source13-2.js) mais il y a cette fois-ci un seul auditeur qui est sur le tableau. On pourra associer des propriétés Javascript à chaque cellule.

Exo 13-3: Comme le 13-1 mais (index13-3.html,source13-3.js) mais sans boite de dialogue : un click sur une cellule transforme le texte en zone de saisie (<INPUT TYPE=”text” VALUE=”...”>) et lorsque l’utilisateur entre une nouvelle valeur suivi de la touche RETURN, la zone de saisie disparaît pour laisser la place à du texte correspondant à la nouvelle valeur.

Note : Pour éviter plein de zone de saisie, on pourra mettre le focus sur la nouvelle zone de saisie (focus()) et écouter la perte de focus (événement “blur”) pour simuler le RETURN. Et n’oubliez pas de retirer des auditeurs de click (removeEventListener(“click”,auditeur) qui pourraient vous gêner.

Exo 13-4: Ecrire un tableau de 10 colonnes sur 10 lignes qui représente la table de multiplication classique. Ceci à l’aide de l’API DOM

L’édition de chaque cellule sera possible comme suit : un click sur une cellule transforme le texte en zone de saisie (<INPUT TYPE=”text” VALUE=”...”>) et lorsque l’utilisateur entre une nouvelle valeur suivi de la touche RETURN, la zone de saisie disparaît pour laisser la place à du texte correspondant à la nouvelle valeur.