JavaScript Syntaxe

JVS-IN - TP

m2iformation.fr

Sommaire

1 Insertion de JavaScript dans une page HTML

1.1 Via la balise script

L'insertion de code JavaScript dans une page html se fait généralement via la balise <script>. Cette balise délimite la section dans laquelle sera placé le code. Vous pouvez mettre autant de balises <script> que vous le souhaitez au sein d'une même page.

```
<script language="JavaScript"> … </script>
```

```
<head>
     <title>Insertion </title>
     <script language="JavaScript">
          alert("Affichage d'un message");
     </script>
</head>
```

On remarque que les instructions se terminent par ;

1.2 Via une librairie JavaScript

Une librairie JavaScript est un fichier *.js qui ne contient que du code JavaScript, il est possible depuis une page html de faire référence à un tel fichier par le biais de la balise script. Le code de cette librairie devient dès lors utilisable depuis la page html.

```
<script language="JavaScript" src="maLib.js" />
```

```
<head>
     <title>Insertion </title>
     <script language="JavaScript" src="maLib.js"/>
     <script>
          //Utilisation des fonctions de maLib.js
     </script>
</head>
```

1.3 Sur les attributs de certaines balises html

Certains attributs de balise html sont un peu spéciaux, ils correspondent à un événement qui se produit sur l'élément graphique correspondant, on peut leur associer un morceau de code à exécuter. Prenons l'exemple du bouton, il est représenté via la balise <input type="button" />, et lorsqu'on cliquera sur ce bouton l'attribut onclick="" permet d'associer un comportement au clic.

```
<input type="button" onclick="dosomething();" />
```

Ou en précisant le langage :

```
<input type="button" onclick="javascript:" />
```

```
<body>
    <input type="button" onclick="alert('Affichage d'un message');" />
</body>
```

Cette dernière solution est déconseillée car elle aboutit très souvent à un mélange peu maintenable de code HTML et JavaScript.

1.4 Commentaires

1.4.1 Commentaire simple ligne

```
// Voici un commentaire sur une seule ligne
```

1.4.2 Commentaire multi-lignes :

```
/*  Voici un commentaire
    qui s'étend sur
    trois lignes */
```

2 Les fonctions

2.1 Appels sans retour

Comme dans beaucoup de langages de programmation, les fonctions permettent de regrouper un ensemble d'instructions. C'est au concepteur de l'application de faire un bon découpage de ces regroupements de manière à rendre ces fonctions réutilisables. Lors d'un appel de fonction, il suffit de préciser le nom de la fonction, suivi entre parenthèses des arguments qu'elle attend.

```
nomDeLaFonction(arg1,arg2,arg3, … );
```

```
alert("Affichage d'un message");
```

Le rôle de la fonction alert est d'afficher une boîte de dialogue avec un message à l'intérieur. Lors de l'appel à cette fonction, nous lui passons l'argument `"Affichage d'un message"` pour qu'elle puisse remplir son rôle.

2.2 Appels avec retour

Certaines fonctions ont pour rôle de nous retourner un résultat, c'est le cas de la fonction prompt qui permet à l'utilisateur de saisir une valeur dans une boîte de dialogue. Pour récupérer cette valeur, il suffira lors de l'appel à la fonction de récupérer et stocker dans une variable la valeur retournée par la fonction. (Vous en saurez plus sur les variables par la suite)

```
resultat = nomDeLaFonction(arg1, … );
```

```
nom = prompt("Quel est ton nom ?");
```

2.3 Déclaration de fonction sans retour

Comme nous le disions précédemment, une fonction est un ensemble d'instructions qui ont été regroupées pour pouvoir être réutilisées. Lors de sa déclaration il faut lui donner un nom et on peut déclarer un certain nombre d'arguments, suivront l'ensemble des instructions placées à l'intérieur des accolades.

```
function nomDeLaFonction(arg1,arg2,arg3, … ){
    // instructions
}
```

```
function addition(valeur1, valeur2){
    alert(valeur1+valeur2);
}

//appel de la fonction addition
addition(2,3) ;
```

2.4 Déclaration de fonction avec retour

Pour qu'une fonction retourne une valeur il suffit de placer le mot clé **return** suivi de la valeur à retourner

```
function nomDeLaFonction(arg1,arg2,arg3, … ){
    // instructions
    return resultat;
}
```

```
function addition(valeur1, valeur2){
    return valeur1+valeur2;
}

//appel de la fonction addition
total = addition(2,3) ;
```

2.5 Fonctions anonymes

Une fonction javascript peut être anonyme. La déclaration suivante est très courante :

```
var addition = function (valeur1, valeur2){
     return valeur1+valeur2;
}

//appel de la fonction anonyme grâce à la variable addition
total = addition(2,3) ;
```

3 Déclaration d'une variable

La variable est une manière de stocker une valeur simple en mémoire et de pouvoir y accéder par la suite grâce au nom qu'on lui aura donné. Lorsqu'on souhaite associer plusieurs valeurs au même nom, on préfère les structures complexes : tableau, tableau associatif et objet.

En JavaScript, le type de la variable n'est pas précisé lors de sa déclaration, on parle de langage faiblement typé, de plus son type évoluera au gré des affectations, le typage est donc aussi dynamique. Bref, JavaScript est un langage à typage faible et dynamique.

3.1 Déclaration avec initialisation

```
var nom = valeur d'initialisation;
```

```
var d = 4.567;
```

3.2 Déclaration sans initialisation

```
var nom;
```

```
var b;
```

ATTENTION, dans cet exemple b est initialisée à *undefined*.

3.3 Déclarations multiples de variables

```
var nom1, nom2, nom3, …;
```

```
var i, j, k, l;
```

3.4 Les types simples

Parmi les types simples on trouve : Boolean, Number, String

Du fait du typage dynamique, une variable peut prendre successivement ces types.

```
i= 123;

i= "Hello world";

i= true;
```

3.5 Fonctions de conversion

Le typage dynamique est puissant mais dangereux. En effet il est dur de prévoir le résultat d'une opération telle que : `"12"+1`

La solution en est une bonne illustration puisque c'est : `"121"`

Il est donc parfois nécessaire de recourir à des fonctions de conversion comme Number, String, parseInt ou parseFloat.

```
i= "123";
Number(i); // donne 123 (NaN si impossibilité)
parseInt(i); // donne 123 (NaN si impossibilité)
```

Number fonctionne pour différents types (booléens, chaînes, dates) alors que parseInt fonctionne avec une chaîne de caractères.

3.6 Les types complexes ou objets

Les types complexes correspondent souvent à un ensemble de valeurs stockées en mémoire et sur lesquelles on souhaite pouvoir faire des opérations « complexes ». A la différence des types simples, lorsqu'on souhaite affecter une valeur complexe à une variable on utilisera l'opérateur **new**

```
var nom = new Nom_du_type_complexe();
```

```
var maDate = new Date();
```

Comme les objets sont souvent un ensemble de valeurs, on souhaite initialiser ces valeurs au moment où on créée le type complexe. On passe alors les valeurs attendues à la fonction d'initialisation, qu'on appelle également constructeur.

```
var nom = new Nom_du_type_complexe(arg1, arg2, arg3,…);
```

```
maDate = new Date(1977,5,28);
```

JavaScript a quelques objets prédéfinis comme Array, Date, Math ou RegExp.

3.7 La portée des variables

Par défaut les variables sont visibles depuis n'importe où, elles appartiennent au contexte global représenté par l'objet window. Attention donc aux effets de bords avec deux variables de même nom utilisées à des fins différentes.

Toutefois en plaçant le mot clé **var** devant une déclaration de variable, sa portée se trouve restreinte à la fonction dans laquelle elle se trouve. Prenez l'habitude de déclarer vos variables avec le mot-clé var cela vous évitera bien des soucis.

```
var nom = valeur d'initialisation;
```

```
function addition(valeur1, valeur2){
     resultat_1 = valeur1+valeur2;
     var resultat_2 = valeur1+valeur2;
}

//appel de la fonction addition
addition(2,3) ;
alert(resultat_1); //Affiche 5
alert(resultat_2); //Erreur
```

4 Les opérateurs

Opérateurs arithmétiques : + - * % /

Opérateurs booléens (logiques) : && || !

Opérateurs de comparaison : == != > < <= >=

Opérateurs d'affectation : =, +=, -=, *=, /=, %=, ++, --

Attention : l'opérateur + signifie concaténation pour des chaînes de caractère et addition pour des nombres.

JavaScript possède un opérateur inhabituel le triple égal (===) qui permet de comparer le type des variables avant leur valeur.

Exemple :

```
''==0 // true (car 0 et la chaine vide sont évalués tous deux à false)
''===0 // false car on compare une chaine et un nombre
```

5 Les tableaux

Les tableaux bien qu'étant de type complexe peuvent être déclarés de deux manières différentes :

5.1 Déclaration classique de tableau

La première notation se fait grâce au type complexe Array et à l'opérateur new

```
nomDuTableau = new Array();
```

```
tableau = new Array();
```

Il est possible d'initialiser le tableau lors de sa création

```
nomDuTableau = new Array(val1, val2, val3,…);
```

```
saisons = new Array("automne","hiver","printemps","été");
```

5.2 Déclaration simplifiée de tableau

Pour cela, on utilise la notation JSON.

```
nomDuTableau = [val1, val2, val3,…];
```

```
saisons = ["automne","hiver","printemps","été"];
```

5.3 Utilisation classique

Les tableaux sont des structures complexes et comme toute structure complexe ils disposent d'attributs et de méthodes.

Pour connaître la taille d'un tableau, on utilise l'attribut length :

```
nomDuTableau.length;
```

```
alert(saisons.length);
```

Pour ajouter un élément à un tableau, on peut utiliser la méthode push() :

```
nomDuTableau.push(val1, val2, …);
```

```
saisons.push("été");
```

5.4 Utilisation simplifiée

Comme pour leur déclaration, les tableaux disposent d'une écriture simplifiée pour être lus ou alimentés.

```
nomDuTableau[position];
```

```
saisons[0];  //retourne la première valeur de ce tableau
```

```
nomDuTableau[position]=valeur;
```

```
saisons[0]= "automne"; //intègre la valeur "automne" à la position 0
```

6 Structures de contrôle

6.1 Bloc d'instructions

```
{
    instruction1;
    instruction2;
    instruction3;
    // ...
}
```

Un bloc d'instruction est délimité par une accolade ouvrante et une accolade fermante. Il peut contenir une ou plusieurs instructions. Il peut également contenir d'autres blocs d'instructions.

7 Instructions de contrôle

REMARQUE 1 : dans ce qui suit, une « **expression-booléenne** » est une expression retournant **true** ou **false**. Toute autre expression doit être testée avec un des opérateurs relationnels afin de renvoyer true ou false. **"statement"** représente quant à lui, un bloc d'instructions.

7.1 Test if

```
if(expression-booléenne)
    statement
```

```
if( valeurTeste > 7) {
   alert("Eureka !"); }
```

7.2 Test if-else

```
if(expression-booléenne)
    statement

else
    statement
```

```
if(valeur > cible)          // Plus grand
    return +1;
else if(valeur < cible)     // Plus petit
    return -1;
else                        // Egalité
    return 0;
```

Attention : il est donc recommandé de toujours utiliser des accolades.

7.3 Opérateur if ternaire

expression-booléenne ? valeur0 : valeur1

```
resultat = i < 10 ? i * 100 : i * 10;
```

Si `i` est inférieur à 10, `resultat` vaut 100 fois i, sinon résultat vaut 10 fois i.

7.4 Boucle for

for(initialisation; expression-booléenne; pas)

statement

```
for(var i = 0; i < 128; i++) {
    alert("val: " + i );
}
```

Cette boucle va afficher :

val: 0

val: 1

val: 2

...

val: 126

val: 127

7.5 Boucle while

```
while(expression-booléenne)
    statement
```

```
r = 0;
while(r < 0.80) {
   r = Math.random();
   alert(r);
}
```

Cette boucle va s'exécuter et afficher la valeur de r tant que celle-ci est inférieure à 0.8. Dès que r prendra une valeur supérieure ou égale à 0.8, elle sera affichée puis la boucle s'arrêtera.

7.6 Boucle do-while

```
do
    statement

while(expression-booléenne);
```

Le do-while fonctionne comme le while sauf que « **`statement`** » sera exécuté au moins une fois.

7.7 switch

```
switch(integral-selector) {
   case integral-value1 : statement; break;
   case integral-value2 : statement; break;
   case integral-value3 : statement; break;
   // ...
   default: statement;
}
```

```
saisons = ["automne","hiver","printemps","été"];


switch(saisons[0]) {
   case "automne" : case "hiver" :
      alert("Saison automne/hiver"); break;
   case "printemps": case "été":
      alert("Saison printemps/été"); break;
   default:
      alert("Saison inconnue");
}
```

7.8 break et continue

Le break sort définitivement de la boucle. Le continue interrompt l'itération courante et passe à la suivante.

```
for(i = 0; i < 100; i++) {
   if(i == 74) break;         // Fin de la boucle
   if(i % 9 != 0) continue;   // itération suivante
   alert(i);
}
```

8 Exceptions

8.1 Lever une exception

```
throw "Chaine_de_caratères_précisant_la_nature_de_l_exception";
```

```
   if(solde < 0) throw "le joueur est fauché";
```

8.2 Attraper une exception

Lorsqu'une exception est levée, le déroulement du programme est interrompu jusqu'à ce qu'il rencontre un bloc qui traitera cette exception. Ce bloc est défini à l'aide des mots clé **try** et **catch.**

```
try {
   //Bloc susceptible de lever une exception
  //directement ou dans une fonction appelée
}catch(e){
  //Bloc permettant de traiter l'exception si
  //elle intervient
}
```

```
try{

debiterLeJoueur(1000) ;

}catch(e){
   alert(e);
}
```

9 Objets et JSON

9.1 Tableaux associatifs

En javascript les objets sont représentés en mémoire par des tableaux associatifs. On y associe des propriétés et des valeurs (simples, tableaux, objets ou aussi fonctions).

La notation JSON (JavaScript Object Notation) permet de rendre cette déclaration simple.

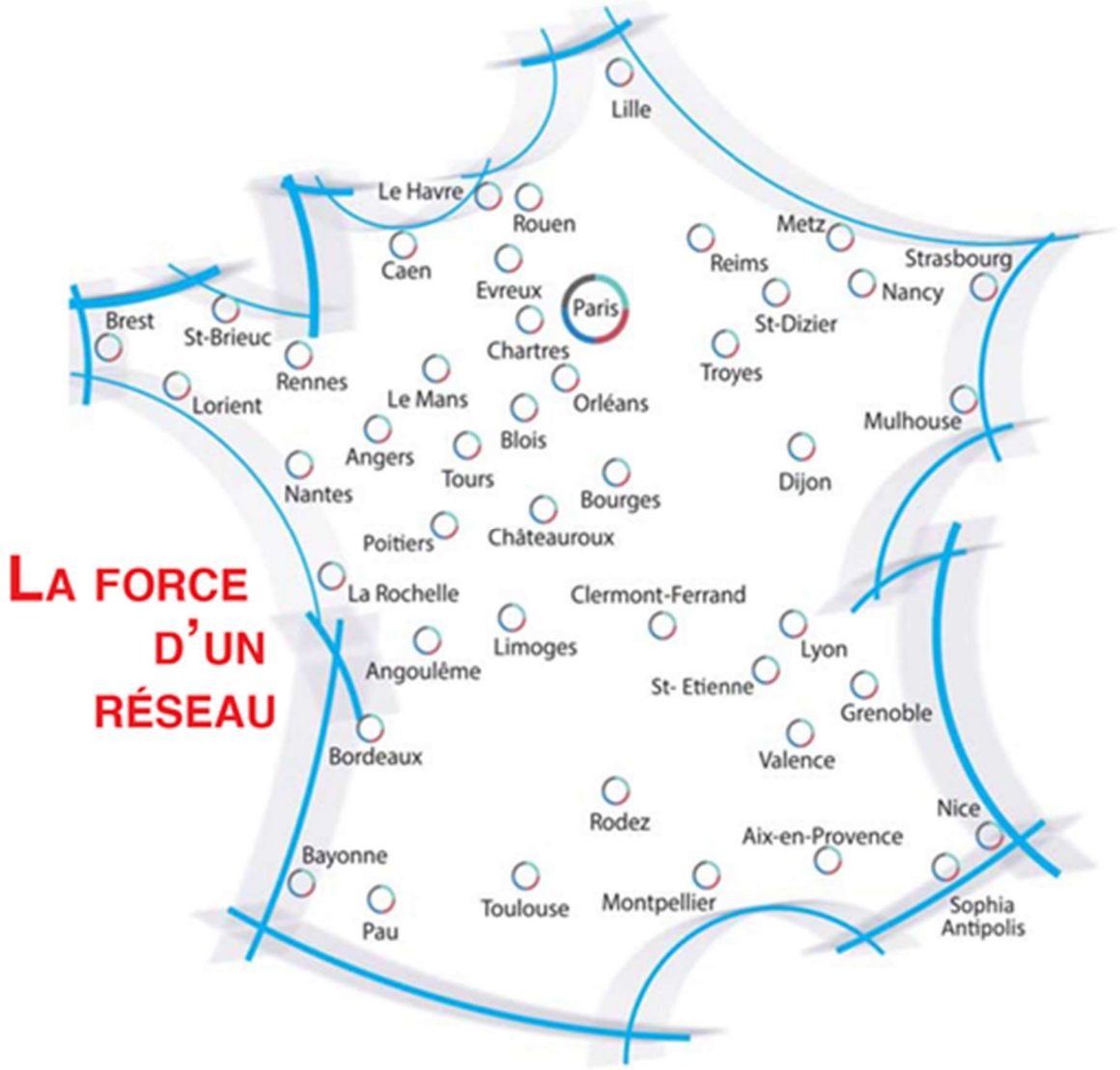
```
var loto = {"joueur" : "xavier", "grille" :[12,24,36,3,6,9]} ;
//exemple d'utilisation
console.log(loto.joueur) ;
```

Les propriétés ne sont pas forcément entourées de guillemets doubles mais c'est fortement recommandé pour les échanges asynchrones.

9.2 Méthode

Une fonction peut être associée à un objet de la manière suivante.

```
var loto = {"joueur" : "xavier", tirage : function(){return "perdu" ;}} ;
//exemple d'utilisation
console.log(loto.tirage()) ;
```



Formation

N°Azur 0 810 007 689

PRIX D'UN APPEL LOCAL DEPUIS UN POSTE FIXE

Découvrez également l'ensemble des stages à votre disposition sur notre site

http://www.m2iformation.fr