Secteur Tertiaire Informatique
Filière étude - développement

Développer des scripts clients dans une page web

Le langage JavaScript

Accueil

Apprentissage

Période en entreprise

Evaluation

Support_De_Cours-JAVASCRIPT.docx

SOMMAIRE

I. INTRODUCTION

A. GENERALITES

Brendan Eich de la société Mosaic Communications Corporation a développé un langage de script coté serveur : le Livescript. Au changement de nom de cette société pour devenir Netscape Communications Corporation elle s'associe avec la société Sun pour développer un langage de script mais coté client. C'est en 1995 qu'est annoncé le langage JavaScript. Netscape intègre son moteur JavaScript dans la version 2.0 de son navigateur en 1996. Microsoft va répondre en développant son langage JScript qu'il inclut dans IE 3.0. Netscape soumet JavaScript à l'ECMA Internationnal pour une standardisation. C'est en 1997 que sont publiées les premières recommandations du langage ECMAScript : ES1.

L'objectif initial de JavaScript est d'introduire un peu d'interactivité dans les pages html et de faire effectuer des traitements simples directement par le poste de travail de l'utilisateur. Pour cela on introduit des « scripts », petits morceaux de programmes qui sont directement insérés dans le code html et qui sont exécutés par le navigateur qui gère la page. On dit qu'il s'agit donc d'une programmation « côté client » (à la différence des scripts Perl, PHP, ASP, JSP exécutés côté serveur).

Javascript est un langage faiblement typé et basé sur des objets (pas de classes), qui a une fausse réputation de simplicité. Il est même déroutant et difficile à maitriser. C'est le langage commun de tous les navigateurs et son usage est devenu incontournable.
Son code est intégré complètement dans le code html et est interprété par le navigateur client. C'est un langage événementiel : il reconnaît et réagit à des événements provoqués par l'utilisateur (comme un clic de la souris, une validation de formulaire, etc.)

Pour assurer la sécurité, l'exécution du code JS introduit dans une page html ne doit pas être intrusive pour la machine cliente. Pour cela le code ne contient aucune fonction capable de lire ou modifier des fichiers locaux, et du coté réseau, le code ne peut communiquer des données qu'avec le serveur HTTP de la page considérée. Il ne peut pas non plus aller chercher des informations dans les autres pages web ouvertes sur d'autres serveurs, ni fermer une fenêtre qu'il n'aurait pas lui-même ouverte.

Mais ce langage, à l'origine assez limité, a vu son champ d'action s'étendre ces dernières années avec les technologies "Web 2.0" et "AJAX" et surtout avec l'arrivée de nombreuses bibliothèques telles que « Prototype » et maintenant « JQuery ».
En effet assez récemment, on a redécouvert l'intérêt de faire plutôt travailler le client que le serveur. Cela se concrétise actuellement par l'explosion de techniques dites AJAX ou WEB 2 (alors que le WEB 3 est déjà annoncé).
De ce fait, les pages WEB s'enrichissent pour devenir des documents composites, dynamiques et forcément plus complexes. Le codage traditionnel en est bouleversé : la structure de la page html est assurée par le DOM (Document Object Model), sa mise en forme est confiée aux extensions CSS (Cascading Style Sheet) et l'interactivité globale interne/externe est assurée par JavaScript, avec les extensions AJAX.

II. OU PLACER LE CODE JS ?

Les séquences de code JS peuvent être placées à divers endroits dans la page WEB et sous des formes différentes :

A. DE FAÇON GENERALE, N'IMPORTE OU ENTRE LES BALISES <SCRIPT> ET </SCRIPT>

<**script type**="text/javascript">....</**script**>

- l'attribut « type » est optionnel en html5
- l'attribut « language » quelques fois présent, est non normalisé
- l'attribut « src » précise l'uri du fichier JavaScript externe (voir plus bas)

Il est conseillé d'insérer le code JS, les déclarations de variables globales et de fonctions dans la section <head> de la page html. Ce code est alors évalué au début du chargement de la page, sans pour autant être exécuté tout de suite. Les exécutions de ces fonctions surviendront en général lors d'une action de l'utilisateur, le plus souvent en réponse au déclenchement d'un événement.

```
<head>
  <script type="text/javascript">
    //code : déclarations des variables et des fonctions
  </script>
</head>
```

En revanche les instructions insérées dans la page html sont immédiatement exécutées en séquence, en même temps que le code html qui s'y trouve.

Exemple : (Cours-JS-Ch02_Exemple-01.html)

```
<html>
  <head>
    <title>Cours-JS-Ch02_Exemple-01 : calculer</title>
    <meta charset="UTF-8">
    <script type="text/javascript">
      alert("du code dans le head");
    </script>
  </head>
  <body>
    <h3>Avant le Javascript</h3>
    <script type="text/javascript">
      r = 7 * 9;
      alert('7*9 = ' + r);
    </script>
    <h3>Après le Javascript</h3>
  </body>
</html>
```

Si le code JavaScript devient important, il convient de le placer dans un fichier externe d'extension « .js » qui sera inclus à la page html dans la partie <head>

Ce fichier nommé ci-dessous *source.js* doit être un fichier accessible au moment de l'exécution, dans le répertoire courant ou à une adresse URL valide.

```
<script type="text/javascript" src="chemin/source.js" ></script>
```

Un tel fichier externe permet de réutiliser le code dans de multiples pages, sans avoir à l'inclure explicitement dans le code html de chaque page. Et surtout il n'y a pas besoin d'intervenir dans chacune des pages lors d'une évolution du code d'une fonction.

B. ASSOCIE A UNE BALISE HTML QUI GERE UN EVENEMENT (DOM 0)

Le code JS est généralement inséré sous forme d'un appel de fonction, affectée à un gestionnaire d'événement. Un tel gestionnaire apparait comme un nouvel attribut d'une balise.

L'exécution du code ou de la fonction JS (préalablement déclarée) constitue un traitement en réponse à l'événement détecté. L'un des problèmes qui se posent au programmeur est de savoir où afficher le résultat de ce traitement.

Un événement survient à l'initiative de l'utilisateur, par exemple en cliquant sur un bouton, ou après la saisie du texte dans un champ de formulaire.

```
<balise ... onEvenement="code JS"  | "fonction JS">
```

où

- balise » est le nom de la balise
- « onEvenement » est un nouvel attribut de la balise : onclick, onblur, …

Exemple : *(Cours-JS-Ch02_Exemple-02.html)*

```
<html>
  <head>
    <title>Cours-JS-Ch02_Exemple-02 : calculer</title>
  </head>
  <body>
    <input type="button" value="Calculer" onclick="r= 7 * 9;alert('7*9 = '+r) ;" >
  </body>
</html>
```

fonction — *évènement* — *code JavaScript*

C. ASSOCIE AU PSEUDO-PROTOCOLE JAVASCRIPT: DANS UNE URL.

Cette pseudo-URL permet de lancer l'exécution d'un script écrit en JS (souvent un appel de fonction).

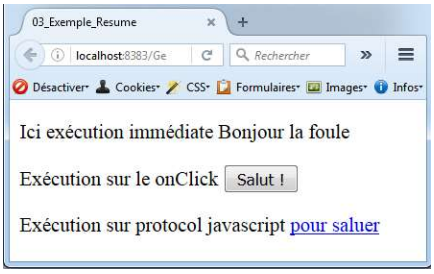
```
<a href="javascript:code JS" >texte ou image</a>
```

Exemple : (Cours-JS-Ch02_Exemple-03.html)

```
<html>
  <head>
    <title>Cours-JS-Ch02_Exemple-03 : calculer</title>
  </head>
  <body>
    .....
    <a href="javascript:r = 7 * 9;alert('7 * 9 = ' + r);">7 x 9 ?</a>
  </body>
</html>
```

Exemple général : (Cours-JS-Ch02_Exemple-04.html)

```
<html>
  <head>
    <title>Cours-JS-Ch02_Exemple-04 : resumer</title>
  </head>
<body>
  <p>Ici exécution immédiate
    <script type="text/javascript">
          document.write("Bonjour la foule !");
          // alert("Bonjour la foule !");
    </script>
  </p>
  <p>Exécution sur événement onclick
     <input type="button" value="Salut !" onclick="alert('Bonjour tout le monde !');"/>
    </p>
  <p>Exécution sur protocole javascript:
    <a href="javascript: alert('Bonjour tout le monde !')">pour saluer</a>
    </p>
</body>
</html>
```



III. ECRITURE DANS LE DOCUMENT HTML

Pour écrire un texte dans le document html, l'objet « document » nous offre une méthode « write() ».
Le texte peut contenir des balises html qui seront interprétées par le navigateur.

Attention lors de l'appel de cette méthode :

- s'il se fait au sein du code html (dans une balise <script> dans le <body>), le texte et/ou la balise seront intégrés dans la page html.
- s'il se fait à l'intérieur d'une fonction, à l'exécution de cette fonction, un nouveau document sera ouvert et le document courant sera fermé. (appel implicite à la méthode open() de l'objet document)

Exemple : (Cours-JS-Ch03_Exemple-01.html)

```
<html>
   <head>
      <title>3_Exemple-02_TexteImage</title>
      <meta charset="UTF-8">
   </head>
   <body>
      <script type="text/javascript">
         document.write("<p>Bonjour !<br>" + "Comment allez vous ", "aujourd'hui
</p>");
         document.write("<img src=\"LogoCDI.jpg\" />")
      </script>
   </body>
</html>
```

Exemple : (Cours-JS-Ch03_Exemple-02.html)

```
<html>
   <head>
      <title>3_Exemple-01_Write</title>
      <script type="text/javascript">
         function nouveauContenu()
         {
            alert("chargement du nouveau contenu");
            // écriture dans un nouveau document : l'ancien est vidé
            document.write("<h1>Assez de l'ancien contenu, passons au nouveau !</h1>");
         }
      </script>
   </head>
   <body>
      <p>Ceci est un test de document.write()</p>
      <br>
      <script type="text/javascript">
         // écriture dans le document déjà ouvert : body contient la balise h1
         document.write("<h1>Ceci est le contenu original du document.</h1>");

         alert("Appel de la fonction");

         nouveauContenu();
      </script>
   </body>
</html>
```

IV. SYNTAXE DU LANGAGE

A. LES COMMENTAIRES

Pour la compréhension du code, ne pas hésiter à inclure des commentaires :

- // pour une simple ligne de commentaires
- /* .....*/ pour les encadrer sur plusieurs lignes.

B. LA SEQUENCE OU BLOC D'INSTRUCTIONS

Une instruction se termine toujours par un point-virgule « ; ».
Un bloc d'instructions se trouve entouré par des accolades « { ..... } ».

```
{
Instruction 1;
Instruction 2;
.....
}
```

C. L'INSTRUCTION CONDITIONNELLE

La syntaxe générale est : Exemple : (Cours-JS-Ch04_Exemple-01.html)

```
if (condition) {
   séquence 1
} else {
   séquence 2
}
```

Les instructions conditionnelles peuvent être imbriquées :

```
if (condition) {
   séquence 1
} else {
   if (condition2) {
      séquence 2
   } else {
      séquence 3
   }
}
```

Il est toujours préférable de mettre les accolades ouvrantes et fermantes au niveau des séquences même si cette dernière ne comporte qu'une seule instruction.

D. L'ITERATION

1. L'instruction « for »

Permet de répéter une séquence d'instructions tant qu'une condition est vraie :

```
for (valeur initiale; condition; poursuite) {
   séquence d'instructions
}
```

Sur un tableau (voir plus loin) il est possible d'utiliser la syntaxe suivante :

```
for (indice in tableau) {
   séquence d'instructions
}
```

Exemple : (Cours-JS-Ch04_Exemple-02.html)

```
<script type="text/javascript">
  function hasard(N) {
            // renvoie une valeur entière au hasard entre 1 et N inclus
            return Math.floor(Math.random()*N)+1;
  }
  document.write("Affichage de 10 nombres au hasard<br>");
  max = prompt("Nombres au hasard de 1 à ", "100");
  for (var i  = 0; i < 10;  i++) {
            nombre = hasard(max);
            document.write("Nombre " + (i + 1) + "  = ",  nombre,  "<br>");
  }
</script>
```

2. L'itération « while »

L'instruction répétitive while permet de répéter une séquence d'instructions : cette séquence peut être exécutée de 0 à n fois (0 si la condition est tout suite fausse)

```
while (condition) {
  séquence d'instructions;
}
```

Exemple d'affichage de nombres aléatoires : (Cours-JS-Ch04_Exemple-03.html)

```
<script type="text/javascript">
  function hasard(N) {
            // renvoie une valeur entière au hasard entre 1 et N inclus
            return Math.floor(Math.random()*N)+1;
  }
  document.write("Affichage de 10 nombres au hasard<br/>");
  max = prompt("Nombres au hasard de 1 à ", "100");

  var i  = 0;

  while (i < 10) {
            nombre = hasard(max);
            document.write("Nombre " + (i + 1) + "  = ",  nombre,  "<br/>");
            i++ ;
  }
</script>
```

3. L'itération « do … while »

L'instruction répétitive do … while permet de répéter une séquence d'instructions : cette séquence sera exécutée de 1 à n fois

```
do {
   séquence d'instructions;
} while (condition)
```

V. LES DONNEES ET LES VARIABLES

A. LES TYPES

1. Les nombres : type « Number »

En JavaScript tous les nombres sont des flottants sur 8 octets (même les entiers !). Il faut être vigilant dans les calculs.
Les constantes numériques peuvent être écrites en notation décimale comme 2.718, ou en notation scientifique comme 2718E-3.
Des symboles existent pour désigner que la valeur n'est pas un nombre « NaN », ou est infinie « +/-Infinity », et des fonctions de test existent comme « isNaN() ».

2. Les booléens : type « Boolean »

Il existe 2 constantes booléennes : **true** ou **false** (Exemple : 5_Exemple-01_Boolean.html)

3. Les constantes spéciales

La constante spéciale « **null »** signifie "rien", "absence de valeur". Elle est attribuée à toute variable utilisée sans être définie (par exemple prompt() retourne null si on sélectionne le bouton Annuler)
A noter que dans un test logique null est considéré comme false.

La constante « **undefined »** signifie "de type indéterminé". Elle est attribuée aux variables non initialisées.

4. Les chaines de caractères

Les chaînes en JavaScript sont des séquences de caractères Unicode, entourées indifféremment par des guillemets « " " » ou des apostrophes « ' ' ».
JavaScript facilite beaucoup l'affichage des résultats en convertissant automatiquement les valeurs numériques en chaînes de caractères, ce qui permet de concaténer des nombres avec des chaînes de caractères (transtypage automatique).
Dans l'instruction d'écriture du document courant, « document.write( ) », les données à afficher sont séparées par l'opération de concaténation « + ».On peut insérer des codes HTML sous forme de chaines, qui seront bien interprétées à l'exécution comme véritables balises, et non pas affichées telles quelles.

Une chaîne de caractère est en fait un objet JS qui possède des propriétés et des méthodes. Voir le chapitre sur l'objet String.

B. DECLARATION ET AFFECTATION DE VARIABLES

Il n'y a pas de déclaration de type pour les variables en JS. Elles prennent le type de leur contenu.
Il est donc possible qu'une variable change de type en cours d'exécution.

Il existe deux types de variables en JS. Les variables **locales** et les variables **globales** (Voir ci-dessous).

Les syntaxes de déclaration de ces variables sont les suivantes :

- « nomVariable = valeur » ou « var nomVariable = valeur » :
  pour une variable globale déclarée en dehors de toutes fonctions
- « var nomVariable = valeur » :
  pour une variable locale déclarée à l'intérieur d'une fonction
- « let nomVariable = valeur » :
  pour une variable locale déclarée à l'intérieur d'un bloc d'instruction (entre deux accolades { … })

Une variable déclarée sans valeur est « undefined ».
Il est possible de connaître le type d'une variable par la fonction « typeof() » ou l'opérateur typeof.

Exemple : (Cours-JS-Ch05_Exemple-01.html)

```
var x ;
document.write("Voici la valeur de x : x = " + x + "<br />" );
document.write("et son type : typeof(x) = " + typeof(x) );
x = "une chaine" ;
document.write("Voici la valeur de x : x = " + x + "<br />" );
document.write("et son type : typeof(x) = " + typeof(x) );
```

Il est possible de convertir une variable de type « String » en « Number » par les fonctions natives parseInt() et parseFloat().

```
var a = "123";
var b = "456";
document.write(a + b,"<br />");            // affiche 123456
document.write(parseInt(a) + parseInt(b),"<br />");      // affiche 579
```

C. PORTEE DES VARIABLES

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

De façon générale les variables définies directement dans une séquence de script (entre <script> ....</script>) ont une portée globale sur toutes les parties de script.

Mais on peut rendre une variable locale en utilisant le mot clé « var » lors de la déclaration à l'intérieur d'une fonction.

Il existe un autre mot clé « let » qui permet de restreindre la portée à un bloc d'instruction (entre deux accolades).

Exemple : (Cours-JS-Ch05_Exemple-02.html)

```
<SCRIPT language="Javascript">
  var a = 12;                          // « a » est une variable globale
  b = 4;                               // « b » est une variable globale

  function multipliePar2( b ) {
    var a = b * 2;            // « a » est une variable locale à la fonction
    return a;
  }

  document.write("Le double de ", b, " est ", multipliePar2(b));          // affiche 8
  document.write("La valeur de a est ", a);       // affiche 12
</script>
```

D. CONSTRUCTION DES EXPRESSIONS

On peut distinguer plusieurs types d'expressions :

- expressions arithmétiques : construites par opérations sur les nombres :
  - les 4 opérations usuelles : « + », « - », « * », « / »
  - le modulo ou reste par une division entière : « % »
- expressions d'affectation (ou attribution) :
  - l'opérateur d'affectation : « = » variable = expression
  - les opérateurs d'attribution associatifs : « += », « -= », « *= », « /= », « %= »
    x += 4 *équivaut à* x = x + 4
    x *= 2 *équivaut à* x = x * 2
  - l'opérateur d'incrémentation : « ++ », ou de décrémentation : « -- »
    x++ *équivaut à* x = x + 1
- expressions chaînes de caractères :
  - l'opérateur de concaténation (mise bout à bout) de deux chaînes : « + »
  - ajout de la chaîne de droite à la chaîne de gauche : « += »
- expressions booléennes ou logiques :
  - opérateurs de comparaison entre 2 valeurs :
    l'égalité (même valeur) : « == » , la différence : « != »
    le supérieur à : « > », le supérieur ou égal à : « >= »
    le inférieur à : « < », le inférieur ou égal à : « <= »
    l'égalité strict (même valeur, même type) : « === »
    la différence strict : « !== »
  - opérateurs logiques :
    le ET : « && », le OU : « || », le NON : « ! », utilisés surtout dans les instructions conditionnelles.
- Les expressions conditionnelles
  variable = (condition) ? valeur1 : valeur2 : évalue la condition et renvoie valeur1 si vrai, ou valeur2 si faux.
  Exemple : message = (fin == true) ? "bonjour" : "au revoir"

E. TABLEAUX JS

Un tableau en JavaScript est en fait un « objet Array » (voir chapitre sur les objets JS). Plusieurs déclarations sont possibles : en faisant appel au constructeur de l'objet ou directement par extension.
Les indices des éléments d'un tableau ne peuvent être que numériques (premier élément ayant l'indice 0), on parle de « **tableaux indicés** ».

1. Déclaration

```
En utilisant le constructeur :

var nomTableau = new Array( dimension ) ;     // le paramètre dimension est optionnel
ou
var nomTableau = new Array(valeur1, valeur2, …) ;


En utilisant expression litéral

var nomTableau = [ ] ;

ou
var nomTableau = [valeur1, valeur2, …] ;
```

Le mot « new » commande la construction d'un objet de type « Array », c'est-à-dire, tableau. Le paramètre dimension, s'il est présent, est le nombre d'éléments.

Exemples :

```
var monTableau = new Array(10) ;
monTableau[0] = "lundi" ;
monTableau[1] = "mardi" ;
etc …
ou
var joursOuvres = new Array("lundi", "mardi", "mercredi", "jeudi", "vendredi") ;
```

On peut aussi créer un tableau directement "en extension", sans faire appel au constructeur Array(). On liste les valeurs des éléments dans [...].

```
var joursOuvres = ["lundi", "mardi", "mercredi", "jeudi", "vendredi"] ;
```

2. Utilisation

Les éléments d'un tableau de taille « dim », sont indicés à partir de 0 jusqu'à « dim – 1 ».

Exemple :

```
document.write("Les jours ouvrés sont : <ul>")
for (i = 0 ; i < 5 ; i++) {
  document.write("<li>", joursOuvres[i],"</li>") ;
}
document.write("</ul>");
```

L'itération sur un tableau peut se faire par la syntaxe suivante :

```
document.write("Les jours ouvrés sont : <ul>")
for (var idx in joursOuvres) {
  document.write("<li>", joursOuvre[idx],"</li>") ;
}
document.write("</ul>");
```

Exemple : (Cours-JS-Ch05_Exemple-03.html)

3. Propriétés

- « length » donne le nombre d'éléments.

```
document.write("il y a " + joursOuvres.length + " jours dans le tableau jour") ;
```

4. Méthodes

- tableau.reverse() change l'ordre des éléments
- tableau.sort(fonction_tri()) trie suivant l'ordre croissant, ou suivant la fonction indiquée en paramètre optionnel

VI. DECLARATION ET APPEL DES FONCTIONS EN JS

A. DECLARATION GENERALE ET APPEL

- JS lit les fonctions présentes dans la section « head » de la page web, lors de son ouverture, mais ne les exécute pas. Une fonction n'est exécutée qu'au moment de son appel.
- Dans l'écriture de l'appel de la fonction, il faut fournir une liste de valeurs correspondant exactement à la liste des paramètres présents dans la déclaration sinon les valeurs des paramètres sont « undefined ».
- Il est possible de donner des valeurs par défaut.
- Si la fonction retourne une valeur (par l'instruction return), elle doit être affectée à une variable.

```
<head>
  <script type="text/javascript" >
    function nomfonction(param1, param2 = valeurParDefaut, ...) {
          séquence d'instructions;
          return nom_variable ;          // retour de la fonction : optionnel
    }
</script>
</head>
<body>
  <script type="text/javascript" >
     nomfonction(valeur1, valeur2, ...) ; // appels
    // ou
    variable = nomfonction(valeur1, valeur2, ...);        // appels avec retour
  </script>
</body>
```

Exemple : (Cours-JS-Ch06_Exemple-01.html)

B. VISIBILITE DES PARAMETRES

De façon générale, les paramètres formels (param1 et param2 dans l'exemple) d'une fonction ne sont connus qu'à l'intérieur de la fonction, ce sont des variables locales. De même pour les variables locales (déclarées par le mot clé « var » ou « let »), variables qui sont explicitement déclarées à l'intérieur de la fonction.
Conséquences :
Si la valeur d'un paramètre ou d'une variable locale est modifiée dans la fonction, cette modification ne sera pas connue à l'extérieur de la fonction.

```
<script type="text/javascript">
  function bonjour(nom) {
    // nom est un paramètre local
    ch ="Bonjour ";        // ch est-elle une variable locale ou globale ?
    alert("à l'entrée de la fonction : \n"+ch + nom );
    nom = "Alain";         // on modifie le paramètre local
    alert("avant la sortie de la fonction :\n"+ch + nom );
  }
  var prenom = "Jacques";
  bonjour(prenom);
  // obtient-on le prénom modifié dans la fonction ?
  alert("après appel de la fonction :\n" + ch + prenom );
</script>
```

C. FONCTIONS

La déclaration d'une fonction peut se faire d'une façon littérale. Le mot clé « function » construit un objet pouvant être affecté à une variable. L'appel de cette fonction se fait par la variable.

```
<script type="text/javascript">
  var carre = function( x ){ return x * x };
  document.write("le carré de 12 est : " + carre(12) + "<br />") ;
</script>
```

Les fonctions JS sont en fait des objets et JS construit un tableau pour les arguments de cette fonction.

```
function calculerPrix(PrixUnitaire, NbArticles) {
  if (calculerPrix.arguments.length != 2)
     alert("impossible de calculer le prix !")
  else
     return PrixUnitaire * NbArticles;
}
```

- accès aux paramètres : calculerPrix.arguments[i]
- accès à la taille : calculerPrix.arguments.length

Il est possible de ne pas déclarer d'arguments à la définition de la fonction. Ceci permet d'avoir une fonction acceptant un nombre variable de paramètres.

D. L'OBJET « FUNCTION »

Il existe un constructeur « Function() », semblable au constructeur « Array() », permettant de définir une nouvelle.

La syntaxe en est la suivante :

```
var nom_fct = new Function("arg1", .. "argN", "code_fct"), où :
```

- arg1, .. argN sont des chaînes définissant la liste des paramètres de la fonction
- code_fct est une chaîne définissant le code

Exemple :

```
<script>
  var carre = new Function("x","return x*x")
  document.write("carré de 12 est : " + carre(12) + "<br />")
</script>
```

E. FONCTIONS JS « GLOBALES »

Il existe certaines fonctions dites **globales** indépendantes de la hiérarchie JavaScript (indépendant de tout objet).

- **encodeURI(), decodeURI()** : Tous les caractères spéciaux sont transformés en séquences de signes ASCII. Les caractères avec accents en français par

exemple, les caractères spéciaux ainsi que les espaces, les parenthèses, accolades etc. ... sont donc codés. Attention ceci ne prend pas en compte la partie « paramètre » ; utiliser pour cela les fonctions encodeURIComponent()et decodeURIComponent().

- **encodeURIComponent(), decodeURIComponent() :** permet d'encoder la partie paramètre de l'URI

- **isNaN(x) :** test si le paramètre x est un numérique. Renvoie true si Ok.

- **isFinite(x) :** test si le paramètre x est un numérique de valeur +infini, -infini ou NaN. Renvoie false si c'est le cas.

- **eval()** : Lance l'interpréteur JS sur l'expression fournie, et retourne le résultat. Peut servir à créer des variables JS à partir d'une chaine.

Exemple :

```
eval("8 + 5 + 3") ;
```

- **parseInt(x, [base]), parseFloat(x) :** Transforme la chaîne de caractères fournie, si possible, en nombre entier ou en flottant. Très utilisée pour transformer en nombre les chaines saisies dans les formulaires. Si la conversion n'est pas possible, renvoie NaN (Not a Number), éventuellement n'effectue qu'une conversion partielle du début de chaine.

- **toString(base) :** Convertit l'objet (généralement un nombre) en une chaine représentant le nombre écrit dans la base indiquée.

- **alert(), confirm()** et **prompt() :** ces trois méthodes font partie de l'objet « window » et sont souvent rattachées aux fonctions globales car le mot clé window est omis. La première affiche une boite d'alerte avec un message et le bouton Ok, la deuxième affiche une boite de confirmation avec les boutons Ok et Cancel, la troisième affiche une boite de saisie et renvoie une chaîne de caractères.

Exemple :

```
var reponse = window.confirm("Voulez vous quitter ?") ;
if( reponse ) {
  window.alert("Au revoir") ;
}
```

VII. CLASSES D'OBJETS PREDEFINIS

Il existe 8 classes d'objets à connaître : **Array, Boolean, Date, Math, Number, String, RegExp** et **Image**, avec leurs propriétés et leurs méthodes.

A. ARRAY

Voir le chapitre sur les tableaux.

B. BOOLEAN

L'objet « Boolean » est utilisé pour convenir un objet quelconque en un objet de type booléen.
La syntaxe générale est :

```
var myBoolean = new Boolean(object);
```

Exemple : (Cours-JS-Ch07_Exemple-01.html)

```
<script type="text/javascript">
   var b1 = new Boolean(0);
   var b2 = new Boolean(1);
   var b3 = new Boolean("");
   var b4 = new Boolean(null);
   var b5 = new Boolean(NaN);
   var b6 = new Boolean("false");
   var b7 = new Boolean("true");
   var b8 = new Boolean(new Date());

   document.write("<h3>0 est un boolean : " + b1 + "</h3>");
   document.write("<h3>1 est un boolean : " + b2 + "</h3>");
  document.write("<h3>Une chaine vide est un boolean : " + b3 + "</h3>");
  document.write("<h3>null est un boolean : "+ b4 + "</h3>");
  document.write("<h3>NaN est un boolean : " + b5 + "</h3>");
  document.write("<h3>La chaine 'false' est un boolean " + b6 + "</h3>");
  document.write("<h3>La chaine 'true' est un boolean " + b7 + "</h3>");
 document.write("<h3>L'objet Date est un boolean " + b8 + "</h3>");
</script>
```

0 est un boolean : false
1 est un boolean : true
Une chaine vide est un boolean : false
null est un boolean : false
NaN est un boolean : false
La chaine 'false' est un boolean true
La chaine 'true' est un boolean true
L'objet Date est un boolean true

Si la valeur initiale passée en paramètre est 0, -0, null, "", false, undefined, or NaN, l'objet booléen sera initialisé à « false ». Dans les autres cas il sera « true ».

C. DATE

L'objet « **Date** » permet de définir et gérer les dates et les heures.
L'origine des dates a été choisie le 1er janvier 1970 et est exprimée en millisecondes.

1. Construction d'un objet de type Date

Pour construire un objet de type **Date**, il faut utiliser le constructeur Date() avec le mot-clé **new.** Il est possible de passer des paramètres au constructeur.
La syntaxe générale est :

```
variable = new Date(liste de paramètres)
```

Attention : les secondes et les minutes sont notées de 0 à 59, les jours de la semaine de 0 (dimanche) à 6, les jours du mois de 1 à 31, les mois de 0 (janvier) à 11, et les années sont décomptées depuis 1970.

Exemples des différents paramètres passés au constructeur pour construire un objet date

- date = new Date(); pour obtenir la date et l'heure courante (connue du système)
- date = new Date(56000) ; en millisecondes depuis 01/01/1970
- date = new Date(2008, 4, 7); une suite convenable de 3 entiers (année, mois, jour)
- date = new Date(2008, 4, 7,10,15,30); une suite de 7 entiers (année, mois, jour, heures, minutes, secondes, millisecondes)

2. Méthodes

Elles permettent d'extraire diverses informations d'un objet date :

- set....() : pour transformer des entiers en Date
- get....() : pour transformer en date et heure des objets Date
- to...() : pour retourner une chaîne de caractères correspondant à l'objet Date

après les préfixes **set** et **get** , on peut mettre **Year, FullYear**, **Month**, **Date** , **Hours**, **Minutes**, **Seconds** pour obtenir respectivement : nombre d'années depuis 1900, le numéro du mois, le N° du jour dans le mois, et les heures, minutes et secondes.

- getDay() donne le N° du jour de la semaine (le 0 tombe le dimanche)
- getTime() donne le nombre de millisecondes écoulées depuis le 1/1/1970, très pratique pour calculer des intervalles entre 2 dates.

Exemple : (Cours-JS-Ch07_Exemple-02.html)

```
<script type="text/javascript">
    var date = new Date();
    //var annee = date.getYear() + 1900;
    var annee = date.getFullYear();
    var mois = date.getMonth() + 1;
    var jour = date.getDate();
    document.write("<h3>Date " + jour + " / " + mois + " / " + annee + "</h3>");
    document.write("<h3>Heure " + date.getHours() + " : " + date.getMinutes() +
            " : " + date.getSeconds() + "</h3>");
</script>
```

D. MATH

Les fonctions mathématiques usuelles doivent être préfixées par le nom de l'objet Math, desquelles elles dépendent. Ce sont les « méthodes » de calcul de l'objet Math.

Liste des principales méthodes :

- Math.log() , Math.exp() , Math.abs() ,Math.cos (), Math.sin() , Math.tan()
- Math.floor(), Math.ceil() entier immédiatement inférieur / supérieur.

- Math.pow(base, exposant), fonction puissance, où base et exposant sont des expressions numériques quelconques évaluables.
- Math.max() , Math.min()
- Math.random(), nombre "réel" choisi au hasard dans [0 , 1]
- Math.round() arrondit à l'entier le plus proche.

« Math.PI » désigne une propriété de l'objet Math : le nombre PI

E. STRING

La déclaration se fait par des guillemets ou des apostrophes.

```
var chaine = "<p>Bonjour !</p>" ou var chaine = '<p>Bonjour !</p>'
```

Crée une variable nommée chaine et lui attribue :
- le type : « string »
- la valeur : « Bonjour ! »
- la longueur : 15

1. Propriété

La seule propriété est **length** : chaine.length donne 15

2. Méthodes

- **chaine.toUpperCase(),** pour mettre chaine en majuscule.
- **chaine.toLowerCase(),** pour mettre chaine en minuscule.
- **chaine.substring(debut, fin)** extrait une partie de chaine, du caractère de position debut+1, jusqu'à fin.
- **chaine.slice(debut, fin)** idem que substring avec debut et fin qui peuvent être négatifs (comptage à partir de la fin).
- **chaine.substr(debut, long)** idem que slice mais le deuxième paramètre est le nombre de caractères que l'on veut extraire.
- **chaine.indexOf(ch2)** donne la 1ère position de la sous-chaine ch2 dans la chaine chaine. Retourne -1 si la sous-chaine n'est pas trouvée.
- **chaine.charAt(n)** donne le caractère en n^{ième} position (n de 0 à chaine.length-1).
- **chaine.charCodeAt(n),** retourne la valeur du code unicode.
- **chaine.split (séparateur)** Appliquée à un texte, elle fournit un tableau de chaines dont les éléments sont les sous-chaines extraites suivant le séparateur. Si le séparateur est "" on obtient un tableau de caractères.

F. NUMBER

L'objet Number est en fait un « wrapper » des type primitifs.
Deux utilisation sont possibles :

- **var n = new Number(valeur)**, créer une nouvelle instance de Number
- **Number("chaineNum")** fait appel à une fonction qui convertie la chaine représentant une valeur numérique en une valeur numérique.

1. Propriétés

L'objet **Number** possède plusieurs propriétés non modifiables, elles doivent être préfixées par le nom de l'objet Number :

- Number.MAX_VALUE, correspond au plus grand nombre enregistrable
- Number.MIN_VALUE, correspond au plus petit nombre enregistrable
- Number.NaN, nombre invalide
- Number.NEGATIVE_INFINITY, correspond à un nombre infiniment petit (inférieur à -MAX_VALUE)
- Number.POSITIVE_INFINITY, correspond à un nombre infiniment grand (supérieur à MAX_VALUE).

2. Méthodes

- La méthode *toString([base])*
  Retourne la valeur du nombre sous forme d'une chaine de caractères
- La méthode *toLocaleString([base])*
  Retourne la valeur du nombre sous forme d'une chaine de caractères dans la locale de l'ordinateur
- La méthode *valueOf()*
  Idem à toString() sans pouvoir imposer la base
- La méthode *toFixed([nbrdigit])*
  Retourne la valeur du nombre avec le nombre de décimales fixé par « nbrDigits »
- La méthode *toPrecision([nbrChiffre])*
  Retourne la valeur du nombre en fixant le nombre de chiffres (avant et après la virgule)
- La méthode *toSource()*
  Retourne l'expression à l'origine du nombre

Exemple : (Cours-JS-Ch07_Exemple-03.html)

G. REGEXP

Les expressions régulières agissent sur des chaînes de caractères pour permettre de les analyser, les filtrer ou de chercher des motifs contenus dans celles-ci.

Pour construire un objet de type **RegExp**, il faut utiliser un constructeur **RegExp()**) avec le mot-clé **new** et passer sa valeur au constructeur :

```
Variable ExpReg = new RegExp("valeur", modificateur)
```

On peut aussi utiliser la syntaxe simplifiée :

```
Variable ExpReg = /valeur/modificateur
```

- Le paramètre modificateur décrit le type d'expression régulière :

  "g" : l'expression sera analysée globalement
  "i" : l'expression sera analysée indifféremment sur les majuscules ou les minuscules.
  "m" : l'expression sera analysée en tant que expression multiligne
  Le modificateur peut prendre une combinaison de ces 3 caractères : (g, gi, gim, …)

Dans une expression régulière, outre les caractères devant se retrouver physiquement dans la chaîne qui seront traitées par celle-ci, on rencontre des caractères « spéciaux » dont il est important de bien saisir la sémantique pour pouvoir les utiliser de façon pertinente et réaliser ainsi des modèles efficaces et concis. Ces caractères sont de trois types :

- caractères définissant des *littéraux*

Ces caractères sont introduits par le caractère « \ » et la liste est donnée dans le tableau ci-dessous.

Caractère	*Signification*
\ /	*/*
**\ **	**
\ .	*.*
\ +	*+*
\ *	***
\ ?	*?*
\ |	*|*
\ }	*}*
\ {	*{*
\ (	*(*
\ )	*)*
\ [	*[*
\ ]	*]*
\t	*tabulation horizontale*
\r	*retour chariot*
\n	*saut de ligne*
\f	*saut de page*
\v	*tabulation vert.*
\xxx	*car. ASCII de code octal xxx*
\xhh	*car. ASCII de code hexa hh*

- les caractères *d'ensemble*

Les caractères d'ensemble permettent de construire à la demande une collection de caractères ou de désigner des collections prédéfinies.

La liste des caractères représentant cette collection est mise entre les caractères « [» et «] ». On peut aussi représenter cette suite par seulement le premier et le dernier caractère séparés par un tiret. Exemple : les caractères de l'alphabet majuscules « [A-Z] », minuscules « [a-z] », les chiffres décimaux « [0-9] », hexadécimaux « [0-9A-F] ».

Si, au contraire on désire représenter un caractère pouvant être quelconque hormis appartenir à un ensemble parfaitement cerné, on utilisera le caractère de complémentarité « ^ » en début d'ensemble. Exemple [^!?.,;] désigne tout caractère qui n'est pas un signe de ponctuation.

Certains ensembles peuvent être introduits directement par le caractère « \ » :

Caractère	Signification
[....]	Un qcq des caractères contenus
[^....]	Aucun des caractères contenus
.	Caractère quelconque sauf saut de ligne (équivaut à [^ \n])
\s	Tout caractère de césure (équivaut à [\t\r\n\f\v])
\S	Aucun caractère de césure (équivaut à [^ \t\r\n\f\v])
\w	Tout caractère alphanumérique (équivaut à [a-zA-Z0-9])
\W	Aucun caractère alphanumérique (équivaut à [^a-zA-z0-9])
\d	Tout chiffre (décimal) (équivaut à [0-9])
\D	Aucun chiffre (décimal) (équivaut à [^0-9])

- les caractères de *groupement*

Les caractères de groupement ont pour fonction de regrouper plusieurs éléments constituant un sous-motif du modèle en un seul élément. Le groupement s'opère simplement en l'encadrant d'une paire de parenthèses « (» et «) ».

Il y a plusieurs utilités à pouvoir regrouper des éléments :

- on peut ainsi faire porter les caractères de répétition que nous verrons plus loin sur l'ensemble des éléments du modèle ainsi regroupés. Ainsi, c'est cet ensemble qui sera dupliqué et non les éléments constitutifs individuellement ;
- on peut aussi référencer ce groupement de façon à prévoir une autre occurrence des éléments de la chaîne analysés avec lesquels il a été apparié, plus loin dans le modèle.

Le référencement dont il vient d'être question s'opère par l'apparition dans l'expression régulière du caractère \n où n désigne le n° de parenthésage auquel il fait référence. Pour illustrer d'un exemple simple, supposons que l'on veuille analyser la chaîne JavaScript encadrée de simples ou doubles guillemets (les deux devant être identiques bien entendu).
Si l'on analyse à l'aide de l'expression régulière /['"]JavaScript['"]/, les écritures 'JavaScript" et "JavaScript' seront reconnues correctes, ce que nous refusons. En fait, l'expression d'analyse devra être de la forme : /(['"])JavaScript\1/.
Grâce à cet essai, on constate que cette expression autorise en début de chaîne le caractère ' ou le caractère ", mais une fois le premier analysé, il contraint le second à lui être identique.

- les caractères de *répétition*

Ce sont des caractères dont la fonction est de prévoir dans le modèle un nombre d'occurrences successives du littéral, du groupement ou d'élément d'ensemble sur lequel ils portent.

Caractère	Signification
*	Un nombre indéfini de fois [0,x], x >= 0
+	Au moins une fois [1,x], x > 0
?	Eventuellement une fois [0,1]
{n}	Exactement n fois
{n,m}	Au moins n et au plus m fois [n,m]
{n,}	Au moins n fois [n,x], x >= n

- Le caractère de choix

Le caractère de choix permet de prévoir, dans l'expression régulière, le choix entre différents sous-motifs, séparés par le caractère « |». Comme pour les caractères de répétitions, celui-ci peut porter sur un ou plusieurs littéraux, ensembles ou groupements.

- Les caractères de *positionnement*

Nous avons seulement vu, jusqu'ici, des appariements de caractères. Il existe aussi la possibilité d'apparier et donc, de contraindre encore, selon la position dans la chaîne. Cela est d'un grand intérêt car on va pouvoir mettre en place des ancrages de motifs.

En fait pour toutes les expressions régulières que l'on a vues jusqu'ici, la vérification se contentait de tester si dans la chaîne fournie, une partie de celle-ci répondait au modèle décrit par l'expression régulière.

On peut donc contraindre l'analyse à appliquer le modèle à des endroits bien précis de la chaine à analyser, ceci grâce à quelques caractères spécifiques :

Caractère	Signification
^	Ancre en début de chaîne
$	Ancre en fin de chaîne
\b	Ancre sur limite de mot
\B	(entre \w et \W) Ancre sur non limite de mot

Ne pas confondre le caractère d'ancrage en début avec le caractère de complémentarité. Même s'ils ont une représentation similaire, il n'y a aucune ambiguïté car l'un se situe obligatoirement en début d'expression (après /) tandis que l'autre ne peut apparaître qu'en début de définition d'ensemble (après [).

1. Propriétés

- La propriété *input*
  La propriété javascript **input** de l'objet **RegExp** permet de connaître la dernière chaîne originale pour laquelle une correspondance a été trouvée.
- La propriété *source* (accessible en lecture)
  La propriété javascript **source** de l'objet **RegExp** permet de connaître de profil ou motif de l'expression régulière.

- La propriété *ingnoreCase*
  La propriété javascript **ignoreCase** de l'objet **RegExp** permet de connaître la valeur du modificateur **i**.
- La propriété *multiline* (accessible en lecture)
  La propriété javascript **multiline** de l'objet **RegExp** permet de connaître la valeur du modificateur **m**.
- La propriété *global* (accessible en lecture)
  La propriété javascript **global** de l'objet **RegExp** permet de connaître la valeur du modificateur **g**.
- La propriété *lastIndex* (accessible en lecture / écriture)
  La propriété javascript **lastIndex** de l'objet **RegExp** permet de connaître ou de modifier le point de départ de la recherche du profil dans la chaîne de caractères.

2. Méthodes

- La méthode *compile()*
  La méthode **regExp.compile(chaîne, modificateur)** permet de changer le motif d'une l'expression régulière déjà créée.
- La méthode *exec()*
  La méthode **regExp.exec(chaîne)** permet de récupérer sous la forme d'un tableau, la première correspondance trouvée entre une chaîne de caractères et une expression régulière et les groupes capturés dans les parenthèses capturantes. Sinon **null**.
- La méthode *test()*
  La méthode **regExp.test(chaine)** permet de vérifier s'il y a correspondance entre un texte et une expression régulière. Elle retourne **true** en cas de succès et **false** dans le cas contraire.
- La méthode *search()*
  La méthode **chaine.search(exp_reguliere)** renvoie la position de la première correspondance de l'expression rationnelle au sein de la chaîne, sinon elle renvoie `-1`.
- La méthode *match()*
  La méthode **chaine.match(exp_reguliere)** retourne un tableau (Array) qui contient les résultats des correspondances et les groupes capturés grâce aux parenthèse. S'il n'y a aucune correspondance, ce sera **null**.
- La méthode *replace()*
  La méthode **chaine.replace(exp_reguliere, remplacement)** permet de remplacer toutes ou certaines correspondances d'un motif dans une chaîne de caractère avec un outil de remplacement. La valeur renvoyée est la nouvelle chaîne ainsi créée. Cet outil de remplacement peut être une chaîne de caractère ou une fonction appelée pour chacune des correspondances.
- La méthode *split()*
  La méthode **chaine.split(exp_reguliere, nombreOcccurence)** permet de découper une chaîne de caractères suivant un séparateur (une expression régulière) Le résultat est un tableau (Array) dont les éléments sont les sous-chaînes de caractères issues de la découpe.

Exemple : Cours-JS-Ch7_Exemple-04
Pour tester : voir site https://regex101.com

VIII. LES OBJETS EN JAVASCRIPT

A. CREATION D'OBJETS

Il existe trois façons de créer des objets en javascript :

- Soit instancier directement un « Object »
- Soit de définir un constructeur
- Soit utiliser la notation littérale

1. Instanciation de l'Object

Object étant l'objet js le plus haut de la hiérarchie, il est possible de créer un nouvel objet en utilisant l'opérateur « new » :

```
var monObjet1 = new Object() ;
```

Maintenant il faut lui ajouter des propriétés :

```
monObjet1.propriete_1 = "prop1" ;
monObjet1.propriete_2 = "prop2" ;
```

On peut aussi lui ajouter des méthodes, exemple toString() :

```
monObjet1.toString = function() { return ('propriété n° 1 = ' + this.propriete_1 +
                                          ' propriété n° 2 = ' + this.propriete_2); }
```

L'utilisation : exemple

```
alert(monObjet1.toString()) ;
//affiche : propriété n° 1 = prop1 propriété n° 2 = prop2
```

2. Utilisation d'un constructeur

En JavaScript, un constructeur est réalisé grâce à une fonction. Il a pour effet de créer un objet vide et éventuellement de l'initialiser en tout ou partie par l'intermédiaire du mot « this » qui constitue une référence vers cet objet :

```
function monObjet(propriete_1, propriete_2) {
  this.propriete_1 = propriete_1 ;
  this.propriete_2 = propriete_2 ;

  this. toString = function() { return ('propriété n° 1 = ' + this.propriete_1 +
                                        ' propriété n° 2 = ' + this.propriete_2); }
}
```

Le constructeur est invoqué par l'opérateur « new » :

```
var monObjet1 = new monObjet('prop1', 'prop2') ;
```

L'utilisation : exemple

```
alert(monObjet1.toString()) ;
//affiche : propriété n° 1 = prop1 propriété n° 2 = prop2
```

3. La notation littérale

Un objet peut se définir par l'utilisation de la notation littérale. L'objet défini plus haut peut se définir comme suit :

```
var monObjet1 = {
  propriete_1 : 'prop1',
  propriete_2 : 'prop2',

  toString : function(){
          return ('propriété n° 1 = ' + this.propriete_1 +
                  ' propriété n° 2 = ' + this.propriete_2) }
}
```

B. LE « GARBAGE COLLECTOR »

Il arrive souvent qu'un objet ne soit plus référencé. Comme dans d'autres langages, le « garbage collector » récupèrera la mémoire occupé par cet objet.

Exemple :

```
var monObjet1 = new monObjet('prop1', 'prop2') ;
…….
monObjet1 = 'chaine' ;
```

C. LES PROPRIETES ET METHODES

En JavaScript, une méthode est appelée en tant que propriété de l'objet dans lequel elle a été définie.

A ce niveau, il convient de préciser que l'utilisation du mot « this » n'est pas réservée aux seuls constructeurs. Employé dans une méthode, il fait référence à l'objet auquel appartient ladite méthode.

Dans le constructeur, en utilisant « this », les propriétés et méthodes sont « publiques », sans ce mot clé les propriétés et méthodes deviennent « privées ».

```
function Personne (ident) {
```

```
  var nom = ident.nom ; // propriété privée
  var prenom = tronque(ident.prenom) ;          // propriété privée

  this.getNom = function(){ return nom ; } ;   // méthode publique
  this.getPrenom = function() { return prenom ; } ;   // méthode publique

  function tronque(longPrenom) {        // fonction privée
          return longPrenom.substring(0, 10) ;
  } ;
}
```

Utilisation :

```
var pers1 = new Personne( {nom : 'Dejour', prenom : 'Adan'}) ;
```

D. PROTOTYPAGE ET HERITAGE

Dans ce que l'on vient de voir dans la définition des objets, chaque objet créé possède ses propres propriétés et méthodes. Il y a recopie du code d'où une perte de place mémoire dans le cas de propriétés et méthodes communes à tous nos objets d'une même « classe ».

Javascript, propose la notion d'objet prototype. Tout objet possède un prototype dont il hérite des propriétés et méthodes.

Si on redéfinit une propriété du même nom qu'une des propriétés du prototype, cette dernière sera masquée par la propriété propre.

Les propriétés d'un objet masquent les propriétés de même nom de son prototype.

Du fait de l'utilisation du prototype, l'accès aux propriétés est plus délicat :

- en écriture, la modification, par l'utilisateur, d'une propriété ne pourra se faire que si celle-ci n'appartient pas au prototype (modifier une propriété prototype revient à modifier cette propriété pour l'ensemble des objets ayant le même prototype).
- en lecture, JavaScript vérifie si l'objet possède la propriété en propre. Si ce n'est pas le cas, la vérification se fait sur le prototype de l'objet. Si la recherche réussit (dans l'un ou l'autre cas), la valeur atteinte est fournie, sinon, la valeur rendue est « undefined ».

Pour créer une propriété dans le prototype d'un constructeur, la syntaxe sera la suivante :

```
<Nom constructeur>.prototype.<Nom propriété> = <expression> ;
```

Exemple :

```
Personne.prototype.age = 0 ;       // ajout de la propriété age au prototype
```

Il est possible de créer un nouvel objet à partir d'un autre (~héritage). Pour cela on peut employer la méthode « create() » de l'Object. L'appel au constructeur de l'objet père se fait par la méthode « call() » dont les paramètres sont l'objet lui-même (this) suivi des éventuels paramètres attendu par le constructeur père.

Exemple :

```
function Employe(ident, codeAgent) {

  Personne.call( this, ident ) ;
  var code = codeAgent ;

  this.toString() = function() {
          alert ('L'employé : ' + this.getNom() + ' a le code agent ' + code) ;
     }
}

Employe.prototype = Object.create(Personne.prototype) ;
```

```
var empl1 = new Employe( {nom : 'Dejour', prenom : 'Adan'}, 123) ;


alert(empl1) ;
```

Etablissement référent
Afpa de Créteil

Equipe de conception
Didier Bonneau

Date de mise à jour 23/01/2017