Langage Javascript

Table des matières

JavaScript est un langage de programmation orienté objet principalement employé dans les pages web interactives mais aussi pour les serveurs.

1. Premiers pas

1.1. Introduction

Javascript (JS) est un langage orienté objet qui est utilisé à l'intérieur des pages web pour les rendre interactives.

Voici quelques exemples de ce que l'on peut en faire :

- ouvrir des pop-up (les petites fenêtres qui s'ouvrent de manière intempestive)
- faire défiler un texte
- insérer un menu dynamique (qui se développe au passage de la souris)
- …

Javascript est exécuté côté client. Pour cela :

- votre ordinateur récupère le code source d'une page web sur un serveur.
- votre navigateur interprète la page et les scripts qu'elle contient.
- la page formatée s'affiche sur votre écran. Les scripts, quant à eux, sont mis en mémoire et seront lancés dès que l'événement attendu se produira.

1.2. Un premier script

Pour coder un script en javascript, il faut quelques outils :

- un éditeur de texte : Notepad++.
- un navigateur : FireFox.
- le greffon Web Developer.

Pour insérer du JavaScript, la seconde solution consiste à écrire le script entre deux balises spécifiques, <script> et </script> :

```
<script type="text/javascript">

   /* votre code javascript se trouve ici
   c'est deja plus pratique pour un script de plusieurs lignes */

</script>
```

On peut les placer soit dans l'en-tête (<head> ... </head>;), soit dans le corps (<body> ... </body>;) de la page (x)HTML :

- dans le corps de la page, sont à placer les scripts à exécuter au chargement de cette dernière (lorsque le navigateur "lira" le code, il l'exécutera en même temps).
- en revanche, sont à placer dans l'en-tête les éléments qui doivent être exécutés plus tard (lors d'un événement particulier, par exemple).

Exemple : on veut :

- une boîte de dialogue indiquant le début du chargement de la page (donc, le code est à placer au début du corps de la page),
- une autre indiquant la fin du chargement de celle-ci (donc, à la fin du corps).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>

<!-- en-tete du document -->

  <title>Un exemple</title>

  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
</head>

<body>

  <!-- script pour le debut du chargement -->
  <script type="text/javascript">
  <!--
    alert('Debut du chargement de la page');
  //-->
  </script>

  <!-- ici se trouve le contenu de la page web -->
  <p>
    Vous testez un script...<br />
    Enjoy ;)
  </p>

  <!-- script pour la fin du chargement -->
  <script type="text/javascript">
  <!--
    alert('Fin du chargement de la page');
  //-->
  </script>

</body>

</html>
```

Tant que la première boîte de dialogue est ouverte, la page n'est pas chargée. En effet, le navigateur exécute le JS au fur et à mesure du chargement de la page : il attend donc que le script soit terminé avant de charger la suite.

Il y a deux manières d'écrire un commentaire.

- Les commentaires sur une seule ligne se placent sur la fin de la ligne, après // (deux slashs). Les commentaires sur plusieurs lignes se placent entre /* et */.
- La fonction alert() ouvre une boîte de dialogue : c'est une petite fenêtre qui sert à communiquer avec l'utilisateur (lui afficher ou lui demander quelque chose).
- JS utilise le point-virgule (;) ou le retour à la ligne pour séparer des instructions.

On peut très bien placer le code JS dans un fichier indépendant. On dit que le code est importé depuis un fichier externe. L'extension du fichier externe contenant du code JavaScript est .js.

On va indiquer aux balises le nom et l'emplacement du fichier contenant le(s) script(s), grâce à la propriété src :

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

Ce fichier contiendra par exemple : alert('Bonjour');

Remarque : on peut appeler directement le code JS à l'intérieur de la balise qui va être concernée par le script grâce au gestionnaire d'événements[1].

- Cette propriété est caractéristique du JS : elle suffit à dire au navigateur : "attention, ce qui suit est du JS".
- Elle porte le nom de l'événement qui doit déclencher le script (c'est pour cela qu'on parle de "gestionnaire d'événements").
- Elle contient comme valeur le script à exécuter lors de cet événement.

Sa syntaxe est la suivante : <nom eventHandler="script" />

Exemple :

```
<a href="#" onclick="alert('Bonjour !');">lien</a>
```

Résultat : la boîte de dialogue apparaît lorsque vous cliquez sur le lien.

Pour un lien, il est possible d'écrire du JavaScript directement à la place de l'adresse d'un lien, en le faisant précéder de javascript:, comme dans cet exemple :

```
<a href="javascript:alert('Coucou');"> Cliquez ici </a>
```

Il existe d'autres événements que le clic de souris, en voici quelques-uns :

- onclick : au clic (gauche) de la souris
- ondblclick : lors d'un double clic
- onmouseover : au passage de la souris
- onmouseout : lorsque la souris "sort" de cet élément (le contraire de onmouseover).

Gestionnaires d'événements de la balise <body> :

- onload : au chargement de la page
- onunload : lorsqu'on quitte la page (ou qu'on change de page).

Exemple : pour créer une page disant "Bonjour" et "Au revoir" :

```
<body onload="alert('Bonjour !');" onunload="alert('Au revoir !');">
  <!-- corps de la page -->
```

1 event handler

```
</body>
```

1.3. Les variables

Pour déclarer une variable, on utilise la syntaxe : var nom;

nom est le nom de la variable et il peut contenir les 26 lettres de l'alphabet, en majuscules et en minuscules, les 10 chiffres ainsi que le underscore (le tiret du bas, touche 8 sur les claviers français). Attention JS est sensible à la casse.

Remarque : le nom ne doit pas commencer par un chiffre.

Il y a également des mots réservés : break, case, catch, continue, debugger, default, delete, do, else, finally, for, function, if, in, instanceof, new, return, switch, this, throw, try, typeof, var, void, while, with.

Il est possible, lorsque l'on crée une variable, de lui affecter immédiatement une valeur.

```
var annee = 2006;
var message = "Bonjour, visiteur";
```

1.3.1. Les chaînes de caractères

On utilise indifféremment les guillemets " (dits "double quotes") ou les apostrophes ' (dites "simple quotes") pour délimiter une chaîne de caractères.

Pour inclure des guillemets " ou des apostrophes dans une chaînes, il faut utiliser le caractère d'échappement \ (antislash).

Exemple :

```
// deux chaines de caracteres
var message1 = "Ceci est un \"petit\" test (pas besoin d'antislash \).";
var message2 = 'Un autre "petit" test (attention à l\'antislash)';

// maintenant, on les affiche
alert(message1);
alert(message2);
```

On peut également insérer des retours à la ligne ainsi que des tabulations avec des caractères spéciaux :

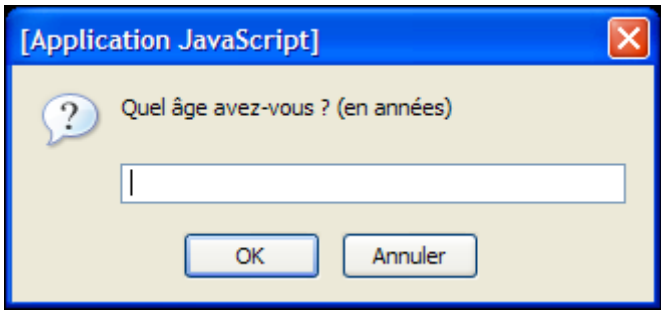
- \n : retour à la ligne.
- \t : une tabulation (ne marche pas dans tous les cas)
- \b : pour insérer un backspace (touche "retour arrière")
- \uXXXX : pour insérer le caractère donc la valeur unicode est XXXX (en hexadécimales).

Pour concaténer plusieurs chaînes de caractères, on utilise simplement le symbole + entre chaque chaîne :

```
var age = 18;                          // on cree la variable pour pouvoir tester
alert("Vous avez " + age + " ans"); // on affiche les chaines mises bout-à-bout
```

Il existe un moyen pour demander à l'internaute de saisir du texte :

```
var age = prompt("Texte d'invite");
```




```
var age = prompt("Quel âge avez-vous ? (en années)"); // on demande l'age
alert("Vous avez " + age + " ans");                   // on affiche la phrase
```

1.3.2. Les nombres

Les nombres, au même titre que les chaînes de caractères, sont un type de variable. Ils se classent en deux catégories :

- les nombres entiers
- les nombres à virgule

Voici les opérateurs de base :

- + (addition), exemple : 52 + 19 = 71
- - (soustraction), exemple : 52 - 19 = 33
- * (multiplication), exemple : 5 * 19 = 95
- / (division), exemple : 5 / 3 = 1,666...667
- % (modulo) : 52 % 19 = 14 (car 52 = 19*2 + 14)

Pour convertir une chaîne en nombre, il faut utiliser parseInt (convertir en un nombre entier) ou parseFloat (convertir en un nombre décimal).

Si la conversion échoue, on obtient un "NaN" (Not a Number).

Quelques exemples vous montrant la souplesse de ces fonctions :

- parseInt("12.9 Euros") vaudra 12 (on convertit en entiers, les chiffres après la virgule sont ignorés)
- parseInt(" 12 Frs ") vaudra également 12 (l'espace en début de chaîne est ignoré)
- parseInt("J'ai 12 Euros") vaudra Nan (la chaîne commence par une lettre)
- parseFloat(" 12.9 Frs ") vaudra 12.9
- parseFloat("3,14") vaudra... 3 : il faut utiliser le point et non la virgule. La conversion va donc s'arrêter après le "3".

Exemple :

```
var nombre1 = prompt('Premier nombre ?');
var nombre2 = prompt('Deuxieme nombre ?');

// on convertit en nombres decimaux et on calcule
var resultat = parseFloat(nombre1) + parseFloat(nombre2);
alert("La somme de ces deux nombres est " + resultat);
```

Remarque : la conversion Nombre → chaîne se fait automatiquement lorsque l'on affiche un

nombre au milieu de texte.

Il existe en JS des techniques permettant de raccourcir l'écriture des opérations.

Au lieu d'écrire : nombre = nombre + 1;

on écrira : nombre++;

De même pour : nombre = nombre - 1;

on pourra écrire : nombre--;

Il existe d'autres raccourcis qui fonctionnent sur le même principe qui fonctionnent pour toutes les opérations de base :

```
var nombre = 2;
nombre += 4;          // nombre vaut 6...
nombre -= 3;          // ... nombre vaut maintenant 3
nombre *= 5;          // ... nombre vaut 15
nombre /= 3;          // ... nombre vaut 5
nombre %= 3;          // ... nombre vaut 2 (car 5 = 1 * 3 + 2)
```

1.4. Les fonctions

Une fonction est une série d'instructions qui effectue des actions qui sont répétées plusieurs fois dans le code sans avoir à les réécrire à chaque fois.

Lorsqu'on appelle une fonction, il y a trois étapes.

1. L'entrée: on donne des informations à la fonction en lui passant des paramètres avec lesquelles travailler.
2. Le traitement : grâce aux informations qu'elle a reçues en entrée.
3. La sortie : une fois qu'elle a fini son traitement, la fonction renvoie un résultat.

La syntaxe pour coder un fonction est donc :

```
function nomFonction(parametres)
{
        // Insérez vos instructions ici
}
```

- function : mot réservé qui définit une fonction
- nomFonction : c'est le nom de la fonction (pas d'accents, pas d'espaces, etc.)
- parametres (correspond à l'entrée) : valeurs avec lesquelles la fonction va travailler. Il peut ne pas y avoir de paramètres.

Remarque : Les parenthèses sont obligatoires, quand bien même votre fonction n'attendrait aucun paramètre.

Exemple : conversion euros → francs

```
function conversion()
{
   var somme      = prompt("Entrez la valeur en Euros :");
   var resultat   = somme * 6.55957;
   alert(somme + " E\n" + resultat + " Frs");
}
```

On appellera cette fonction lorsque l'utilisateur cliquera sur l'image représentant un symbole monétaire :

```
<img src="EuroFrancs.png" alt="" onclick="conversion();" />
```

Pour une fonction à plusieurs arguments, il faut juste les séparer par des virgules. Lors de l'appel de la fonction, il faudra donner les paramètres dans le même ordre.

Exemple :

```
function conversion(unite1, taux, unite2)
{
   var valeur = prompt("Entrez la valeur à convertir, en " + unite1);
   var resultat = valeur * taux;
   alert(valeur + ' ' + unite1 + '\n' + resultat + ' ' + unite2);
}
```

On l'appelle ensuite de cette manière :

```
<p>
 <img src="EuroFrancs.png" alt="" onclick="conversion('Euros', 6.55957, 'Frs')" /> <br />
 <a href="#" onclick="conversion('m',100, 'cm')">Conversion mètres &gt; centimètres</a>
</p>
```

Pour qu'une fonction renvoie un résultat, on fait précéder la valeur à renvoyer du mot-clé return.

Exemple :

```
function carre(x)
{
   return x*x;
}
function cube(x)
{
   return x*x*x;
}

var resultat = 4*cube(5) - 7*3 / carre(6);
```

1.4.1. Les arguments facultatifs : nombre fixé d'arguments

On peut créer des fonctions pour lesquelles certains arguments sont facultatifs. Au début de la fonction, il faudra vérifier si les arguments facultatifs ont été précisés à l'aide du mot clé undefined. Si ce n'est pas le cas, il va falloir leur donner une valeur par défaut.

Exemple : calcul de la distance

```
function dist(a,b)
{
      // on initialise b a 0 si besoin
      if ( b == undefined )
```

```
            b = 0;
      // on continue ensuite normalement
      if(a > b)
            return a-b;
      else
            return b-a;
}
```

1.4.2. Les arguments facultatifs : nombre illimité d'arguments

Il est possible de récupérer un tableau qui contient tous les arguments d'une fonction à l'aide de la fonction arguments.

Exemple :

```
function maxi(m)
{
   var nb = maxi.arguments;    // on a donc m = nb[0]

   for (var i=1; i < nb.length; i++)
      if (nb[i] > m)           // si l'argument est plus grand que le maximum...
         m = nb[i];            // ... alors c'est le maximum

   return m;
}

var n = maxi(7, 3);
alert(n);

var p = maxi(2, 8, 4, 1, 9, 4);
alert(p);
```

1.5. Les conditions

Il est possible de créer des variables qui ne prennent que deux valeurs : true (vrai) et false (faux).

Exemple :

```
var age = prompt('Quel âge avez-vous ? (en années)');

// si l'age est superieur ou egal à 18 ans, alors l'internaute est majeur
var majeur = (age >= 18);

alert('Vous êtes majeurs : ' + majeur);      // on verifie que ça marche ^^
```

Une condition peut être écrite en JS sous différentes formes. On parle de structures conditionnelles.

1.5.1. if… else

Les symboles suivant permettent de faire des comparaisons simples ou multiples pour les conditions.

Symbole	Signification
==	Est égal à
>	Est supérieur à
<	Est inférieur à

Symbole	Signification
&&	ET logique
||	OU logique
!	NON logique

>=	Est supérieur ou égal à
<=	Est inférieur ou égal à
!=	Est différent de

Pour introduire une condition :

1. on utilise le mot if, qui en anglais signifie « si ».
2. on ajoute à la suite entre parenthèses la condition en elle-même.
3. on ouvre des accolades à l'intérieur desquelles on placera les instructions à exécuter si la condition est **vraie**.
4. **Facultativement** : on utilise le mot else, qui en anglais signifie « sinon ».
5. puis on ouvre des accolades à l'intérieur desquelles on placera les instructions à exécuter si la condition est **fausse**.

```
var age     = prompt('Quel âge avez-vous ? (en années)');
var majeur  = (age >= 18);

if ( majeur )                       // on effectue le test : majeur == true
{
      alert("Salut gamin !");       // action si condition vraie
}
else    // SINON
{
      // action si condition fausse
}
```

Dans le cas des conditions multiples, on utilise les opérateurs logiques.

```
var age     = prompt('Quel âge avez-vous ? (en années)');
var genre   = prompt('Etes-vous un garçon ? (O/N)');
var garcon  = (genre == 'O');

if ( age < 12 && garcon )           // si j'ai moins de 18 ans et que je suis un garçon
{
      alert("Bonjour Jeune homme");       // action si condition vraie
}
else
{
      if ( age < 12 && !garcon )    // si j'ai moins de 12 ans et que je suis pas un garçon
      {
            alert("Bonjour Mademoiselle");
      }

}
```

1.5.2. switch... case

Les structures à base de if… else suffisent pour traiter n'importe quelle condition.

Cependant, pour une imbrication de if... else trop importante, il existe une autre structure plus souple : switch... case.

```
var texte;
```

```
var jour = prompt("Quel jour est-il ?");

switch ( jour )   // on indique sur quelle variable on travaille
{
      case "lundi" :    // dans le cas où c'est le début de la semaine
      case "mardi" :    // on peut tester deux valeurs à la suite
            texte = "courage !!!";
      break;            // ne pas oublier le mot-clef break sinon le JS continue

      case "mercredi" :  // dans le cas où c'est le début de la semaine
            texte = "c'est le jour des enfants";
      break;

      case "jeudi" :    // dans le cas où c'est la fin de la semaine
      case "vendredi" :
            texte = "bientôt le we !";
      break;

      default:    // il faut traiter les autres jours (cas par défaut)
            texte = "vive le week end !";
      break;
}

alert(texte);
```

Le mot-clé default est le traitement par défaut quelle que soit la valeur de la variable.

JS traite les instructions des case à la suite. Pour interrompre le traitement, il faut utiliser break.

1.6. Les boucles

1.6.1. Tant que : while

Une boucle permet de répéter des instructions plusieurs fois.

- les instructions sont d'abord exécutées dans l'ordre, de haut en bas
- à la fin des instructions, on retourne à la première
- et ainsi de suite...

Tant que la condition est remplie, les instructions sont réexécutées. Dès que la condition n'est plus remplie, on sort de la boucle.

```
var i = 0;  // initialisation
while(i < 10)   // condition
{
      alert(i);       // action

i++;    // incrémentation
}
```

Il faut TOUJOURS s'assurer que la condition sera fausse au moins une fois. Si elle ne l'est jamais, alors la boucle s'exécutera à l'infini !

1.6.2. Faire tant que : do... while

Les boucles do-while ressemblent beaucoup aux boucles while, mais l'expression est testée à la fin de chaque itération plutôt qu'au début. La principale différence par rapport à la boucle while est que la première itération de la boucle do-while est toujours exécutée au moins une fois (l'expression

n'est testée qu'à la fin de l'itération).

```
var saisie;
do
      saisie = prompt("Entrez un nom, ou cliquez sur Annuler pour quitter");
while(saisie);  // ca revient au meme que dans l'exemple ci-dessus
```

1.6.3. Boucle : for

La boucle for est un autre type de boucle, dans une forme un peu plus condensée et plus commode à écrire, ce qui fait que for est assez fréquemment utilisé.

for (initialisation ; condition ; incrémentation)

1. initialisation. C'est la valeur que l'on donne au départ à la variable.
2. condition. Comme pour le while, tant que la condition est remplie, la boucle est réexécutée. Dès que la condition ne l'est plus, on en sort.
3. incrémentation, qui vous met à jour la variable à chaque tour de boucle.

```
for(var i=0; i<10; i++)
      alert(i);
```

Cette boucle est utilisée lorsque l'on connait à l'avance le nombre d'instructions à répéter.

1.7. Les tableaux

Les tableaux sont une suite de variables de même type, situées dans un espace contigu en mémoire. Un tableau a une dimension bien précise.

1.7.1. Déclarer un tableau

Pour déclarer un tableau, il faut respecter la syntaxe suivante :

var <nom du tableau> = new Array();

Ex : `var table = new Array();`

Un tableau commence à l'indice n° 0 !

On peut initialiser un tableau de deux façon :

Élément par élément

```
var noms = new Array();
noms[0] = "Pierre";
noms[1] = "Paul";
noms[2] = "Jacques";
```

À la déclaration

```
var noms = new Array("Pierre", "Paul", "Jacques");
var best_scores = new Array(2.5, 4.7, 3.14);
```

Pour un tableau associatif, on crée un tableau vide, et on associe "manuellement" (une par une) toutes les valeurs, en utilisant une chaîne de caractères en tant qu'indice :

```
var scores = new Array();
scores["Toto"]    = 142;
scores["Pierre"]  = 137;
```

1.7.2. Parcourir un tableau

Pour accéder à un élément du tableau, il suffit de donner son indice dans le tableau dans une boucle for classique.

```
var noms = new Array("Pierre", "Paul", "Jacques");
```

```
noms.sort();      // on tri le tableau par ordre alphabétique

for (var i=0; i < noms.length; i++)
{
      chaine = "element : " + i + " -> " + noms[i];
      alert(chaine);
}
```

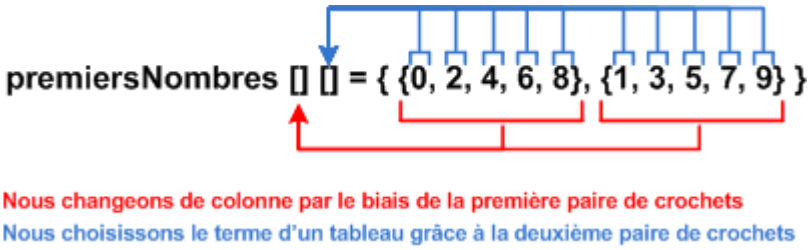
Ou d'utiliser le mot clé « in ».

```
for(var i in tableau)
{
      chaine = "element : " + i + " -> " + noms[i];
      alert(chaine);
}
```

1.7.4. Tableaux multi dimensionnels

Un tableau multidimensionnel n'est rien d'autre qu'un tableau contenant au minimum deux tableaux.

Chaque dimension est désignée par des crochets [].




```
// on cree le tableau, contenant les lignes
var grille = new Array();

// on cree les lignes  les unes après les autres
for(var i=0; i<9; i++)
      grille[i] = new Array();

// on parcourt les lignes...
for(var i=0; i<9; i++)
      // ... et dans chaque ligne, on parcourt les cellules
      for(var j=0; j<9; j++)
            grille[i][j] = 0;
```

2. Les objets

Les objets sont des variables plus complexes qui peuvent être constitués d'une ou plusieurs autres variables (attributs) et d'une ou de plusieurs fonctions (méthodes). Ainsi, un objet peut lui-même contenir un objet ! Un objet peut représenter absolument ce que l'on veut : une chaise, une voiture, un concept philosophique, une formule mathématique, etc.

Pour cela, il faut déclarer ce qu'on appelle une classe et lui donner un nom. On construit ensuite un objet en faisant référence à cette classe. Construire un objet s'appelle l'instanciation.

Exemple :

```
tableau = new Array();     // instanciation de l'objet Array

tableau.length;     // attribut length de l'objet "Array" : renvoie la taille du tableau
tableau.sort();     // méthode sort() de l'objet "Array" : tri le tableau
```

2.1. Les objets HTML

Pour JS, tous les éléments HTML sont des objets (images, liens, etc.) : on va donc pouvoir s'en servir pour en connaître ou en modifier les caractéristiques (comme l'adresse de l'image ou ses dimensions).

2.1.1. L'objet window

L'objet window "représente" la fenêtre du navigateur. Ça va être un objet très utilisé, car il possède de nombreux sous-objets.

Less trois fonctions qui font apparaître des boîtes de dialogues, alert(), prompt() et confirm() sont des objets windows. En fait, si on voulait tout écrire, il faudrait employer window.alert(), window.prompt() et window.confirm()...

2.1.2. L'objet document

L'objet document est un sous-objet de window. Cet objet représente la page HTML affichée dans le navigateur.

Il est possible d'atteindre (comprenez accéder à) tous les éléments de la page HTML avec deux méthodes de l'objet document :

- document.getElementById("id") : permet d'accéder très facilement à l'élément dont l'id est id.
- document.getElementsByTagName("balise") : retourne sous forme de tableau tous les éléments HTML dont on donne la balise en argument.

Exemple : `document.getElementsByTagName('img');`

Cet exemple va retourner un tableau contenant tous les éléments <textarea> de la page.

Puisque la méthode retourne un tableau, il est possible d'accéder à chacun des éléments de la façon suivante :

```
monImage = document.getElementsByTagName('img')[0];
```

Pour modifier l'adresse d'une image et ses dimensions :

```
monImage.src      = "banniere.jpg";
monImage.width    = "800";
monImage.height   = "200";
monImage.onclick = function()
{
      // Affiche la destination du lien sur l'image
      alert( monImage.href );
}
```

Ce code JS doit être placé dans la page, après l'image car s'il se trouve dans l'en-tête de la page, l'image n'aura pas encore été créée lors de son exécution (qui se fait au fil du chargement de la page), et JS ne pourra donc pas lui associer un évènement.

On peut également le faire de cette manière :

```
function lien()
{
      alert( monImage.href );
};
monImage.onclick = lien;   // on parle de la fonction elle-meme : pas de parentheses
```

ou bien :

```
function lien(link)
{
     alert("Ce lien mene ici : " + link.href);
}
```

```
<a href="page.html" onclick="lien(this);">Cliquez ici</a>
```

On utilise dans ce cas le mot-clé this, qui désigne cet élément (d'où son nom).

2.1.3. Les formulaires

En partant d'un formulaire (balise form), il est possible d'accéder à chacun des champs en utilisant leur attribut name.

```
<form id="monFormulaire" method="post" action="demo.php">
   <p>
     <label for="pseudo">Pseudo :</label>
     <input id="pseudo" name="pseudo" type="text" />
   </p>
   <p>
     <label for="modepasse">Mot de passe :</label>
     <input id="modepasse" name="motdepasse" type="password" />
  </p>
  <p>
     <textarea name="contenu" cols="100" rows="15"></textarea><br />
     <input type="submit" value="Envoyer" />
   </p>
 </form>
```

On commencer accéder au formulaire par son id :

```
var monForm = document.getElementById("monFormulaire");
```

Pour accéder au champ nommé "pseudo" à partir du formulaire, sans utiliser son id, on peut utiliser :

- directement pseudo, qui est en fait un sous-objet du formulaire :

  ```
  var champPseudo = monForm.pseudo;
  ```

  Dans le cas des éléments de formulaires, on accède donc aux sous-objets directement par leur nom (l'attribut name).
- un tableau associatif : on utilise pour les indices non pas les numéros, mais les noms.

  ```
  monForm.elements["pseudo"];
  ```

2.2. L'objet Form

L'objet form représente le formulaire lui-même : tous les éléments du formulaire en sont des sous-objets.

Deux évènements qui lui sont associés : onReset et onSubmit.

- onSubmit : cet évènement se produit lorsque le formulaire est envoyé. Il va permettre de vérifier si les champs sont bien remplis avant d'envoyer le formulaire ; si ce n'est pas le cas, on pourra annuler l'envoi.

  ```
  <form method="post" action="page.php" onsubmit="return verifier(this);">
       <!-- ici, le contenu du formulaire -->
  ```

```
        <input type="submit" value="Envoyer" />
    </form>

    function verifier(f) {
        if ( /* le formulaire est bien rempli */ )
            return true;
        else
        {
            alert('Le formulaire est mal rempli');
            return false;
        }
    }
```

- onReset : cet évènement se produit lorsque le formulaire est remis à zéro par un bouton de type reset.

```
<form method="post" action="page.php" onreset="return confirm('Vraiment ?');">
```

Dans ce cas, si l'utilisateur veut réinitialiser le formulaire, il devra le confirmer. Si il confirme son action, true est renvoyé, et le formulaire est alors réinitialisé ; s'il annule, c'est false qui est retourné, et rien ne se passe.

2.2.1. Les éléments, de A à Z

Cette partie a pour but de lister les attributs, méthodes et évènements typiques des différents éléments de formulaire. Certains éléments ont été regroupés dans une même catégorie (signalée par une astérisque) :

- Bouton* : regroupe les balises <button> ainsi que les <input /> de types button, reset et submit
- Case* : regroupe les <input /> de types checkbox et radio
- Select : balise <select>
- Textarea : balise <textarea>
- Texte* : regroupe les <input /> de types text, password et file

Attributs spécifiques aux formulaires. Tous les éléments de formulaire possèdent ces attributs :

- form : le formulaire auquel appartient l'élément
- name : nom de l'objet (nom qui nous sert à désigner cet objet en JS)
- type : type de l'objet (button, select, textarea, ou l'attribut HTML "type" dans le cas d'un input)

Certains éléments possèdent ces attributs :

- checked et defaultchecked : valeur booléenne, true si la case est cochée (cochée par défaut dans le cas de defaultchecked), false sinon
- disabled : si cet attribut vaut true, l'objet est grisé (impossible à modifier / cliquer)
- maxlength : nombre de caractères maximal que peut contenir ce champ
- readonly : si cet attribut vaut true, l'utilisateur ne peut pas modifier le contenu du champ
- size : nombre de caractères / d'options qui sont affichés simultanément
- value et defaultvalue : valeur (valeur par défaut dans le cas de defaultvalue) de l'élément

Quelques éléments particuliers :

- textarea : rows et cols spécifient respectivement le nombre de lignes et de colonnes affichées
- select : multiple indique si on peut sélectionner plusieurs choix ou non ; options est un tableau contenant les option, qui sont au nombre de length ; et selectedindex est l'indice (dans le tableau d'options) du choix sélectionné.

Méthodes spécifiques aux formulaires :

- focus : donne le focus à cet élément (pour une zone de texte, place le curseur à l'intérieur)
- blur : enlève le focus de cet élément (en quelque sorte le contraire de focus)
- click : simule un clic de souris sur cet élément
- select : sélectionne ("surligne") le texte de ce champ

Quelles méthodes pour quels éléments :

Élément	**focus**	**blur**	**click**	**select**
Bouton*	oui	oui	oui	non
Case*	oui	oui	oui	non
Select	oui	oui	non	non
Textarea	oui	oui	non	oui
Texte*	oui	oui	non	oui

Évènements spécifiques aux formulaires :

- onFocus : lorsque l'élément reçoit le focus (pour une zone de texte, quand on place le curseur à l'intérieur)
- onBlur : lorsque l'élément perd le focus (en quelque sorte le contraire de onFocus)
- onChange : lorsque la valeur / l'état de l'élément change (quand on coche la case, qu'on modifie le texte, etc.)
- onSelect : lorsqu'on sélectionne (quand on "surligne") le texte de ce champ

Quels évènements pour quels éléments :

Élément	**onFocus**	**onBlur**	**onChange**	**onSelect**
Bouton*	oui	oui	non	non
Case*	oui	oui	oui	non
Select	oui	oui	oui	non
Textarea	oui	oui	oui	oui
Texte*	oui	oui	oui	oui

2.2.2. Les éléments et leur fonctionnement

On accède au contenu des champs textes (input de type text ou password, ainsi que textarea) à l'aide de l'attribut value, qu'on peut lire, mais aussi modifier.

Afficher la valeur du champ pseudo du formulaire monForm :

```
alert("Vous avez saisi le pseudo : " +
document.getElementById("idForm").elements["pseudo"].value);
```

Mmodifier le contenu d'un textarea dont l'attribut name vaut infos :

```
document.getElementById("idForm").elements["infos"].value = "Vous testez un
script JS";
```

On peut ainsi récupérer et afficher des informations dans des champs de texte.

Les trois balises HTML citées ci-dessus possèdent l'attribut booléen readonly, qui la rend non modifiable par l'utilisateur : on peut ainsi les utiliser pour afficher des messages.

L'objet select représente une liste déroulante, et comme dans toute liste, il peut être intéressant de savoir quel choix est sélectionné :

```
Vous êtes :
     <select name="genre" onchange="voirSelection(this)">
          <option value="rien">Choisissez...</option>
          <option value="garcon">Un garçon</option>
          <option value="fille">Une fille</option>
          <option value="saispas">Je ne sais pas</option>
     </select>
```

- On va récupérer le numéro de l'option sélectionnée, grâce à l'attribut selectedIndex.
- Ensuite, il ne reste plus qu'à lire la valeur value de l'option correspondante, qui se trouve dans le tableau options.

```
function voirSelection(liste)
{
     var numero = liste.selectedIndex;
     var valeur = liste.options[numero].value;
     alert("Vous avez choisi : " + valeur);
}
```

Bien entendu, il y a moyen de réduire ce code en faisant comme ceci :

```
var valeur = liste.options[liste.selectedIndex].value;
```

Il reste à analyser le contenu de la variable valeur pour agir en conséquence :

```
function voirSelection(liste)

{
     var valeur = liste.options[liste.selectedIndex].value;
     if ( valeur != 'rien' )
     {
          if ( valeur == 'saispas' )
               alert('Noob...');
          else
               alert(valeur);
     }
}
```

La première utilisation des cases à cocher (checkbox) est d'en utiliser une seule (par exemple pour activer / désactiver un aperçu automatique).

Dans ce cas, on peut savoir si la case est cochée grâce à son attribut checked, qui est une valeur booléenne (il vaut soit true, soit false). On peut également la (dé)cocher, soit en modifiant la valeur de cet attribut, soit en appelant la méthode click() qui simulera un clic de souris sur la case.

Par exemple, si on a une checkbox dont le nom est "maCase", dans un formulaire dont l'id est "idForm" :

```
if(document.getElementById("idForm").maCase.checked)
     alert("La case est cochee");
else
{
     document.getElementById("idForm").maCase.checked = true;
     alert("La case n'etait pas cochee, mais maintenant elle l'est");
}
```

Il est également possible d'utiliser les cases à cocher par groupe : dans ce cas, toutes les cases du groupe auront le même nom (c'est toujours le cas avec les boutons radio).

```
Vous aimez : <br />
   <input type="checkbox" name="mesCases" value="pizza" /> la pizza <br />
   <input type="checkbox" name="mesCases" value="tartiflette" /> la tartiflette <br />
   <input type="checkbox" name="mesCases" value="ratatouille" /> la ratatouille <br />
   <input type="button" value="Ok" onclick="afficherMessage()" />
```

Toutes ces cases possèdent le même nom ; document.getElementById("idForm").mesCases n'est donc pas un objet, mais un tableau d'objets. On va donc pouvoir accéder à la case numéro i avec document.getElementById("idForm").mesCases[i].

```
function afficherMessage()
{
     var monForm        = document.getElementById("idForm");
     var pizza          = monForm.mesCases[0].checked;
     var tartiflette    = monForm.mesCases[1].checked;
     var ratatouille    = monForm.mesCases[2].checked;

     if ( tartiflette )
          alert("Vous venez de la montagne, non ?");
     if ( ratatouille && !pizza )
          alert("Vous preferez la ratatouille a la pizza ?!");
}
```

Avec ce script, on récupère l'état de chaque case (cochée ou non), et on affiche ensuite des messages selon les cases cochées.

Bien entendu, on n'est pas obligé d'utiliser des variables pour enregistrer l'état de chaque case, on aurait pu écrire directement les document.monForm.mesCases[i] dans les if().

On peut également utiliser une boucle pour parcourir notre tableau de cases et savoir lesquelles sont cochées.

Et c'est d'ailleurs très intéressant dans le cas des boutons radio, car il ne peut y en avoir qu'un seul de coché parmi ceux ayant le même attribut name :

```
function radio()
{
     var cases = document.getElementById("idForm").mesCases;
     var platFavori;

     // on recherche le bouton coche (s'il y en a un)
     for (var i=0; i < cases.length && !platFavori; i++)
          if(cases[i].checked)
               platFavori = cases[i].value;

     // s'il y en a un, on affiche la valeur correspondante
     if ( platFavori )
```

```
        alert("Votre plat favori est : " + platFavori);
}
```

L'intérêt de se baser sur l'attribut value, c'est qu'on peut rajouter des choix à notre formulaire sans avoir à toucher au code JS.

3. Les objets JS

3.1. L' objet "Array"

- length : Cet attribut contient la longueur du tableau.
- join(separateur) : Renvoie tous les éléments du tableaux séparés par séparateur, sous forme de chaîne de caractères. Si aucun séparateur n'est précisé, c'est la virgule qui est utilisée.
- sort() : Trie le tableau par ordre croissant (par ordre alphabétique pour des chaînes de caractères). Le tableau avec lequel on appelle cette méthode est modifié.
- concat(t) : Renvoie un tableau contenant les éléments du tableau, à la fin desquels sont ajoutés les éléments du tableau t. Il est possible de mettre plusieurs arguments : monTableau.concat(t1, t2, t3, t4).
- reverse() : Inverse l'ordre des éléments (le tableau d'origine est modifié).
- slice(debut, fin) : retourne le tableau composé des éléments de t entre debut (inclus) et fin (exclus). Si les valeurs sont négatives, les éléments sont comptés à partir de la fin.

  Il est possible d'appeler t.slice(debut) (sans indiquer la fin), auquel cas le tableau retourné contient tous les éléments à partir de debut, et jusqu'à la fin du tableau.

  Le tableau d'origine n'est pas modifié.
- pop() : retire et renvoie le dernier élément du tableau
- push(x) : ajoute l'élément x à la fin du tableau
- shift() : retire et renvoie le premier élément du tableau. Les autres éléments sont décalés vers le haut pour "combler le trou".
- unshift(x) : ajoute l'élément x au début du tableau. Les éléments sont décalés vers le bas.

3.2. L'objet "Math"

- abs(x) : valeur absolue
- min(x,y) : plus petite valeur
- max(x,y) : plus grande valeur.
- round(x) : arrondi à l'entier le plus proche
- floor(x) : arrondi à l'entier inférieur
- ceil(x) : arrondi à l'entier supérieur.
- sin(x), cos(x), tan(x)
- asin(x), acos(x), atan(x).
- exp(x)

- log(x) : logarithme népérien
- pow(x,y) : x puissance y
- sqrt(x) : racine carrée.
- PI : constante pi
- random() : renvoie un nombre aléatoire de l'intervalle [0;1[

Ces fonctions ne sont pas des méthodes de Math.

- isNaN(x) : renvoie true si x n'est pas un nombre, false sinon
- isFinite(x) : renvoie true si x est un nombre que JS peut traiter, false sinon
- parseInt(str) : convertit la chaîne str en entier
- parseFloat(str) : convertit la chaîne str en flottant.

3.3. L'objet "Date"

Pour récupérer l'heure et la date actuelle, rien de compliqué : il suffit de créer une instance de cet objet.

```
// initialisation avec l'heure courante
var date = new Date();
// initialisation à une autre date (si les derniers paramètres ne sont pas précisés, ils sont mis à 0)
var uneDate = new Date(annee, mois, jour, heure, minute, seconde);
// initialisation avec le nombre de millisecondes depuis le 01/01/1970, 0h00
var derniereDate = new Date(millisecondes);
```

- getTime() : retourne le nombre de millisecondes écoulées depuis le 01/01/1970
- setTime(x) : modifie la date en précisant le nombre x de millisecondes écoulées depuis le 01/01/1970
- getTimezoneOffset() : retourne, en minutes, le décalage horaire avec le méridien de Greenwich.

Exemple :

```
function dateFr()
{
     // les noms de jours / mois
     var jours = new Array("dimanche", "lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi");
     var mois = new Array("janvier", "fevrier", "mars", "avril", "mai", "juin", "juillet", "aout", "septembre", "octobre", "novembre", "decembre");

     // on recupere la date
     var date = new Date();

     // on construit le message
     var message = jours[date.getDay()] + " ";  // nom du jour
     message += date.getDate() + " ";           // numero du jour
     message += mois[date.getMonth()] + " ";    // mois
     message += date.getFullYear();

     return message;
}
```

3.4. L'objet "String"

- length : longueur de la chaîne (nombre de caractères)
- charAt(i) : retourne le iième caractère
- indexOf(str) : retourne la position de str dans la chaîne (-1 si elle n'est pas trouvée)
- lastIndexOf(str) : idem, mais renvoie la position de la dernière occurrence de str
- toLowerCase() : retourne la chaîne en minuscules
- toUpperCase() : retourne la chaîne en majuscules
- concat(str) : retourne la chaîne concaténée avec str
- split(str) : retourne, sous forme de tableau, les portions de la chaînes délimitées par str
- substring(debut,fin) : extrait une sous-chaîne, depuis la position debut (incluse) à fin (exclue).
- substr(debut,i) : extrait une sous-chaîne, depuis la position debut, en prenant i caractères

3.5. L'objet "Image"

L'objet image s'utilise en trois temps :

- On crée une instance de l'objet Image :

```
var monImage = new Image();
```

- On crée des fonctions qu'on associe aux différents évènements :

```
// quand le chargement est terminé
monImage.onload = function()
{
    alert("Chargement fini !");
}
```

- On indique l'adresse de l'image à l'aide de l'attribut src, ce qui aura pour effet de lancer son chargement :

```
monImage.src = "http://www.mon-site.com/une-image.png";
```

Évènements :

- onLoad : une fois le chargement de l'image terminé
- onError : lorsqu'une erreur se produit lors du chargement
- onAbort : lors de l'abandon du chargement

Attributs :

- src : adresse de l'image
- width et height : taille de l'image
- complete : indique si l'image est totalement chargée (true) ou non (false)

Exemple :

```
function changerImage(img, src, maxWidth, maxHeight)
{
   var image = new Image();
```

```
   image.onerror = function()
   {
      alert("Erreur lors du chargement de l'image");
   }

   image.onabort = function()
   {
      alert("Chargement interrompu");
   }

   // une fois l'image chargée :
   image.onload = function()
   {
      // si l'image est désignée par son id
      if ( typeof img == "string" )
         img = document.getElementById(img);

      // si l'image doit être redimensionnée
      var reduction = 1;
      if ( maxWidth && maxHeight )
         if ( image.width > maxWidth || image.height > maxHeight )
            reduction = Math.max(image.width/maxWidth, image.height/maxHeight);

      // on affiche l'image
      img.src     = image.src;
      img.width   = Math.round(image.width / reduction);
      img.height  = Math.round(image.height / reduction);
   }

   // lancement du chargement
   image.src = src;
}
```

4. Le CSS via JS

En JS, il est possible de modifier le style de la page : modifier la couleur d'un bloc, sa position, ses dimensions, ses bordures, le masquer, etc.

Les attributs id et className permettent de lire ou de modifier l'identifiant et la classe d'un élément :

```
document.getElementById("ancien_id").id = "nouvel_id";
document.getElementById("nouvel_id").className = "Une_classe";
```

On peut modifier toutes les propriétés CSS d'un élément de manière très simple :

```
document.getElementById("id").style.uneProprieteCss = "Nouvelle valeur";
```

Attributs des éléments de la page :

- id : valeur de l'attribut id de l'objet
- className : valeur de l'attribut class de l'objet
- style : sous-objet contenant les attributs CSS de l'objet

Accéder aux éléments :

- document.getElementById(id) : renvoie l'élément dont l'id est id

- document.getElementsByTagName(nom) : renvoie un tableau contenant tous les éléments dont on a indiqué le nom de la balise (exemple : tous les div)
- document.getElementsByClassName(classe), qui renvoie un tableau contenant tous les éléments dont on a indiqué la classe
- document.getElementsByClassName(classe, elmt), qui renvoie les éléments contenus dans elmt dont on a indiqué la classe

Afficher ou masquer des éléments :

- toggleVisibility(elmt) : pour rendre visible ou invisible l'élément elmt (qu'on peut également désigner par son id)
- toggleDisplay(elmt) : pour afficher ou masquer l'élément
- getClasses(elmt), qui renvoie un tableau contenant toutes les classes de l'élément elmt (qu'on peut également désigner par son id)
- hasClassName(elmt, className), qui renvoie true si l'élément possède la classe className, false sinon.

Exemple :

Commençons par définir un style par défaut ainsi qu'une classe :

```
div
{
   background-color: blue;
}
.rouge
{
   background-color: red;
}
```

Ajoutons sur la page un div pour faire le test :

```
<div id="test">Je suis un test...</div>
```

Faisons-lui subir quelques changements de couleur...

```
var divTest = document.getElementById("test");
divTest.className = "rouge";
divTest.style.backgroundColor = "green";
divTest.style.backgroundColor = "";
divTest.className = "";
```

- Au début, ni la classe ni le style ne sont définis : la couleur d'arrière-plan est donc celle définie par défaut : le bleu.
- Ensuite, on définit la classe pour le test : celui-ci prend alors la couleur spécifiée dans cette dernière(le rouge).
- On indique une couleur directement grâce à style : elle est donc prioritaire sur la classe. C'est pourquoi notre test devient vert.
- On supprime cet attribut. Le test reprend alors la couleur définie dans la classe : il vire au rouge.
- On supprime finalement sa classe. Il termine donc avec sa couleur par défaut, le bleu.