Fiches Javascript : les fondamentaux

FICHE 1. Les bases du JavaScript.....p1

FICHE 2. Les variables.....p3

FICHE 3. Les conditions.....p5

FICHE 4. Les conditions.....p7

FICHE 5. Les boucles.....p9

FICHE 6. Les fonctions.....p11

FICHE 7. Les objets et les tableaux.....p13

FICHE 8. Les objets et les tableaux.....p15

FICHE 9. Le DOM.....p17

FICHE 10. Le DOM.....p19

FICHE 11. Le DOM.....p21

FICHE 12. Les évènements.....p23

FICHE 13. Les formulaires.....p25

FICHE 1.
Les bases du JavaScript

Le JavaScript, créé en 1995 par Brendan Eich (pour la Netscape Communication Corporation), est un langage de programmation de scripts orienté objet. Si le terme Java est commun au langage du même nom, le JavaScript est radicalement différent.

La version ES5 date de 2009.

On crée un instruction Javascript à l'intérieur des balises `<script>` `</script>`

1.1. La boîte de dialogue alert()

`alert()` est une instruction simple, appelée fonction, qui permet d'afficher une boîte de dialogue contenant un message. Ce message est placé entre apostrophes, elles-mêmes placées entre les parenthèses de la fonction `alert()`.

1.2. La syntaxe Javascript

La syntaxe du Javascript n'est pas compliquée. De manière générale, les instructions doivent être séparées par un point-virgule que l'on place à la fin de chaque instruction :

```
<script>
  instruction_1;
  instruction_2;
  instruction_3;
</script>
```

La syntaxe d'une fonction se compose de deux choses : son nom, suivi d'un couple de parenthèses (une ouvrante et une fermante). Entre les parenthèses se trouvent les arguments, que l'on appelle aussi paramètres.

```
<script>
  myFunction();
</script>
```

Par exemple :

```
<script>
  alert('Bonjour');
// la fonction affiche une boîte de dialogue avec "Bonjour"
</script>
```

1.3. Des fichiers .js

Il est possible, et même conseillé, d'écrire le code Javascript dans un fichier externe, portant l'extension .js. Ce fichier est ensuite appelé depuis la page Web au moyen de l'élément `<script>` et de son attribut `src` qui contient l'URL du fichier .js.

Par exemple dans le fichier hello.js, on écrit :

```
alert('Hello world!');
```

Et dans le body de la page html, on trouve :

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

Pour éviter des problèmes de chargement sur les pages, il est conseillé de placer les éléments `<script>` juste avant la fermeture de l'élément `<body>`.

1.4. Indentations et commentaires

Pour s'y retrouver dans l'écriture du code, on peut l'indenter, c'est-à-dire hiérarchiser les lignes de code avec des tabulations.

Par ailleurs, on peut intégrer des commentaires, qui ne sont pas interprétés comme du code, afin d'expliquer son code ou de mieux s'y retrouver :

```
<script>
  instruction_1; // Ceci est ma première instruction
  instruction_2;
        /* La troisième instruction ci-dessous,
        avec un commentaire sur deux lignes */
  instruction_3;
</script>
```

1.5. Un site pour tester le Javascript

Pour tester le code Javascript sans créer systématiquement des pages web :

http://jsfiddle.net/

FICHE 2.
Les variables

2.1. Bases des variables en JavaScript

Une variable consiste en un espace de stockage, qui permet de garder en mémoire tout type de données. La variable est ensuite utilisée dans les scripts. Une variable contient seulement des caractères alphanumériques, le `$` (dollar) et le `_` (underscore) ; elle ne peut pas commencer par un chiffre ni prendre le nom d'une fonction existante de Javascript. On crée la variable et on lui affecte (ou attribue) une valeur :

```
<script>
  var myVariable;
  myVariable = 2;
</script>
```

ou :

```
<script>
var myVariable = 2;
</script>
```

2.2. Les types de variables

Une variable peut être de type numérique, mais aussi une chaîne de caractères :

```
<script>
var text = 'J\'écris mon texte ici'; /* Avec des apostrophes, le \ sert à échapper une apostrophe intégrée dans le texte, pour ne pas que Javascript pense que le texte s'arrête là.*/
</script>
```

Une variable peut enfin être de type booléen (boolean), avec deux états possibles : vrai ou faux (`true` ou `false`).

2.3. Les opérateurs arithmétiques

On peut utiliser 5 opérateurs arithmétiques : l'addition (+), la soustraction (-), la multiplication (*), la division (/) et le modulo (%). Le modulo est le reste d'une division.

Par exemple :

```
<script>
var number1 = 3,
number2 = 2, result;
result = number1 *
number2;
alert(result); //
Affiche : « 6 »
</script>
```

ou :

```
<script>
var number = 3;
number = number + 5;
alert(number); //
Affiche : « 8 »
</script>
```

qui équivaut à :

```
<script>
var number = 3;
number += 5;
alert(number); //
Affiche : « 8 »
</script>
```

2.4. La concaténation

Une concaténation consiste à ajouter une chaîne de caractères à la fin d'une autre, comme dans cet exemple :

```
<script>
  var hi = 'Bonjour ', name = 'toi', result;
  result = hi + name;
  alert(result);
// Affiche : « Bonjour toi »
</script>
```

ou :

```
<script>
  var text = 'Bonjour ';
  text += 'toi';
  alert(text);
// Affiche « Bonjour toi ».
  </script>
```

On peut convertir un nombre en chaîne de caractères avec l'astuce suivante :

```
<script>
  var text, number1 = 4, number2 = 2;
  text = number1 + '' + number2; /* on ajoute une chaîne de caractères vide entre les deux nombres pour permettre une concaténation sans calcul */
  alert(text); // Affiche : « 42 »
</script>
```

2.5. La fonction prompt(), avec concaténation et calcul

Voici la base de cette fonction :

```
<script>
  var userName = prompt('Entrez votre prénom :');
  alert(userName); // Affiche le prénom entré par l'utilisateur
</script>
```

On peut demander le prénom et afficher un message avec concaténation :

```
<script>
  var start = 'Bonjour ', name, end = ' !', result;
  name = prompt('Quel est votre prénom ?');
  result = start + name + end;
  alert(result);
</script>
```

On peut aussi se servir de la fonction `prompt()` pour un calcul :

```
<script>
var first, second, result;
first = prompt('Entrez le premier chiffre :');
second = prompt('Entrez le second chiffre :');
result = parseInt(first) + parseInt(second); /* la fonction parseInt() transforme la chaîne de caractères en nombre */
alert(result);
</script>
```

FICHE 3.
Les conditions (1/2)

Une condition (`true` ou `false`) est un test qui permet de vérifier qu'une variable contient bien une certaine valeur.

3.1. Les huit opérateurs de comparaison

Il y en a 8 :

`==` : égal à	`===` : contenu et type de variable égal à	`>` supérieur à	`<` : inférieur à
`!=` : différent de	`!==` : contenu ou type de variable différent de	`>=` supérieur ou égal à	`<=` : inférieur ou égal à

Il suffit d'écrire deux valeurs avec l'opérateur de comparaison souhaité entre les deux et un booléen est retourné. Si celui-ci est `true` alors la condition est vérifiée, si c'est `false` alors elle ne l'est pas :

```
<script>
var number1 = 2, number2 = 2, number3 = 4, result;
  result = number1 == number2; // Au lieu d'une seule valeur, on en écrit deux avec l'opérateur de comparaison entre elles
  alert(result); // la condition est donc vérifiée car les deux variables contiennent bien la même valeur
  result = number1 == number3;
  alert(result); // la condition n'est pas vérifiée car 2 est différent de 4
  result = number1 < number3;
  alert(result); // la condition est vérifiée car 2 est bien inférieur à 4
</script>
```

3.2. Les opérateurs logiques

Il y en a 3 :

`&&` qui signifie ET avec par exemple : `valeur1 && valeur2`

Cet opérateur vérifie la condition lorsque toutes les valeurs qui lui sont passées valent `true`.

`||` qui signifie OU avec par exemple : `valeur1 || valeur2`

Cet opérateur est plus souple car il renvoie `true` si une des valeurs qui lui est soumise contient `true`, qu'importent les autres valeurs.

`!` qui signifie NON avec par exemple : `!valeur`

Cet opérateur se différencie des deux autres car il ne prend qu'une seule valeur à la fois. S'il se nomme « NON » c'est parce que sa fonction est d'inverser la valeur qui lui est passée, ainsi `true` deviendra `false` et inversement.

3.3. La condition if else

La condition est composé :

- de la structure conditionnelle `if` ;

- de parenthèses qui contiennent la condition à analyser, ou plus précisément le booléen retourné par les opérateurs conditionnels ;

– d'accolades qui permettent de définir la portion de code qui sera exécutée si la condition se vérifie.

La fonction `confirm()` permet une interaction de l'utilisateur à l'exécution du code (`true` si OK, `false` si Annuler) :

```
<script>
if (confirm('Voulez-vous exécuter le code Javascript de cette page ?')) {
  alert('Le code a bien été exécuté !');
  }
</script>
```

La structure `else` permet de simplifier l'alternative :

```
<script>
if (confirm('Pour accéder à ce site vous devez être une fille, cliquez sur
"OK" si c\'est le cas.')) {
  alert('Vous allez être redirigé vers le site.');
}
  else {
  alert("Désolé, vous n'avez pas accès à ce site.");
  }
</script>
```

On peut ajouter des conditions intermédiaires avec la structure `else if` :

```
<script>
var floor = parseInt(prompt("Entrez l'étage où l'ascenseur doit se rendre (de
-2 à 30) :"));
  if (floor == 0) {
  alert('Vous vous trouvez déjà au rez-de-chaussée.');
  } else if (-2 <= floor && floor <= 30) {
  alert("Direction l'étage n°" + floor + ' !');
  } else {
        alert("L'étage spécifié n'existe pas.");
        }
</script>
```

FICHE 4.
Les conditions (2/2)

4.1. La condition switch

Cette structure permet de gérer une courte liste de possibilités :

```
    <script>
    var drawer = parseInt(prompt('Choisissez le tiroir à ouvrir (1 à 4) :')); //on précise bien le type de la valeur, ici un nombre avec la fonction parseInt()
    switch (drawer) {
    case 1: // on pose chaque cas l'un après l'autre ; on met des apostrophes si l'on vérifie des chaînes de caractères au lieu de nombres
    alert('Contient divers outils pour dessiner : du papier, des crayons, etc.');
    break; // on arrête la fonction pour passer à un autre cas
    case 2:
    alert('Contient du matériel informatique : des câbles, des composants, etc.');
    break;
    case 3:
    alert('Ah ? Ce tiroir est fermé à clé ! Dommage !');
    break;
    case 4:
    alert('Contient des vêtements : des chemises, des pantalons, etc.');
    break;
    default: // on pose une autre possibilité, pour gérer une erreur de l'utilisateur
    alert("Info du jour : le meuble ne contient que 4 tiroirs et, jusqu'à preuve du contraire, les tiroirs négatifs n'existent pas.");
    }
    </script>
```

4.2. Les ternaires

Cette structure permet de simplifier certaines conditions :

```
    <script>
      var startMessage = 'Votre genre : ',
            endMessage,
            adult = confirm('Êtes-vous une fille ?');
      endMessage = adult ? 'Fille' : 'Garçon';
      alert(startMessage + endMessage);
    </script>
```

4.3. Exercice sur les conditions.

Fournir un commentaire selon l'âge de la personne.
Vous devez fournir un commentaire sur 4 tranches d'âge qui sont les suivantes :

Tranche d'âge	**Exemple de commentaire**
1 à 6 ans	« Vous êtes un jeune enfant. »
7 à 11 ans	« Vous êtes un enfant qui a atteint l'âge de raison. »
12 à 17 ans	« Vous êtes un adolescent. »
18 à 120 ans	« Vous êtes un adulte. »

Correction.

```
<script>
var age = parseInt(prompt('Quel est votre âge ?'));
if (1 <= age && age <= 6) {
alert('Vous êtes un jeune enfant.');
} else if (7 <= age && age <= 11) {
alert ('Vous êtes un enfant qui a atteint l\'âge de raison.');
} else if (12 <= age && age <= 17) {
alert ('Vous êtes un adolescent.');
} else if (18 <= age && age <= 120) {
alert ('Vous êtes un adulte.');
} else {
alert ('Erreur !!');
}
</script>
```

FICHE 5.
Les boucles

5.1. Incrémentation et décrémentation

L'incrémentation permet d'ajouter une unité à un nombre au moyen d'une syntaxe courte. À l'inverse, la décrémentation permet de soustraire une unité.

```
<script>
var number = 0;
number++;
alert(number); // Affiche : « 1 »
number--;
alert(number); // Affiche : « 0 »
</script>
```

5.2. La boucle while

Une boucle sert à répéter une série d'instructions. La répétition (ou itération) se fait jusqu'à ce qu'on dise à la boucle de s'arrêter. Pour une boucle, on pose une condition, et la boucle se répète tant que la condition est vérifiée (`true`), selon la structure suivante :

```
<script>
  while (condition) {
        instruction_1; instruction_2; instruction_3;
  } </script>
```

Quand la boucle s'arrête, les instructions qui suivent la boucle sont executées :

```
<script>
  var number = 1;
  while (number < 10) {
  number++; // Tant que le nombre est inférieur à 10, on l'incrémente de 1
  }
  alert(number); // Affiche : « 10 » </script>
```

Un exemple avec prompt() et break

```
<script>
var prenoms = '', prenom; // On crée une variable prenoms pour mémoriser
while (true) {
prenom = prompt('Entrez un prénom :'); // L'utilisateur entre chaque prenom
if (prenom) {
prenoms += prenom + ' '; // Ajoute le nouveau prénom ainsi qu'une espace
} else {
break; // On quitte la boucle
}
} alert(prenoms); // Affiche les prénoms à la suite </script>
```

5.3. La boucle do while (peu utile)

Dans ce cas, la boucle est executée au moins une fois, après quoi on teste la condition, selon la structure suivante :

```
<script>
  do {
        instruction_1; instruction_2; instruction_3;
  } while (condition);
</script>
```

5.4. La boucle for (très utile)

Cette boucle est très utile pour l'incrémentation automatique :

```
<script>
  for (initialisation; condition; incrémentation) {
        instruction_1;
        instruction_2;
        instruction_3;
  } </script>
```

Par exemple :

```
<script>
for (var iter = 1; iter <= 5; iter++) { // On initialise une variable, et tant qu'elle est inférieure ou égale à 5 on l'incrémente de 1.
alert('Itération n°' + iter); // A chaque fois on affiche une boîte de dialogue (5 fois)
} </script>
```

Et avec les prénoms :

```
<script>
for (var prenoms = '', prenom; true;) { // ici sans incrémentation nécessaire, mais avec un point-virgule obligatoire après la condition true
prenom = prompt('Entrez un prénom :');
if (prenom) { prenoms += prenom + ' '; }
else { break; } }
alert(prenoms);
</script>
```

Mais on peut se servir de l'incrémentation pour compter le nombre de prénoms :

```
<script>
for (var i = 0, prenoms = '', prenom; true; i++) {
prenom = prompt('Entrez un prénom :');
if (prenom) { prenoms += prenom + ' '; }
else { break; } }
alert('Il y a ' + i + ' prénoms :\n\n' + prenoms); // Les \n servent à faire des sauts de ligne
</script>
```

FICHE 6.
Les fonctions

Il y a les fonctions ou variables natives (déjà existantes), mais on peut aussi en créer de nouvelles, selon la structure suivante :

```
<script>
function myFunction(arguments) { // Le terme "function" est obligatoire pour déclarer une fonction
// Le code que la fonction va devoir exécuter
} </script>
```

Par exemple :

```
<script>
function byTwo() {
var result = parseInt(prompt('Donnez le nombre à multiplier par 2 :'));
alert(result * 2); }
byTwo(); // On appelle la fonction créée
alert('Vous en êtes à la moitié !'); // Puis un message intermédiaire
byTwo(); // Et appelle de nouveau la fonction </script>
```

6.1. Les variables locales et globales

Attention : toute variable déclarée dans une fonction n'est utilisable que dans cette même fonction. Ces variables spécifiques à une seule fonction ont un nom : les variables locales. Déclarées en dehors des fonction, on parle de variables globales.

```
<script>
var message = 'Ici la variable globale !';
function showMsg() {
var message = 'Ici la variable locale !';
alert(message); }
showMsg(); // On affiche la variable locale
alert(message); // Puis la variable globale
</script>
```

Mais on évite de créer des variables locales et globales qui portent le même nom. En règle générale, on préfère utiliser des variables locales (pour éviter les confusions).

6.2. Les arguments

Pas obligatoire, l'argument peut être ainsi utilisé :

```
<script>
function myFunction(arg) { // Notre argument est la variable « arg »
alert('Votre argument : ' + arg); }
myFunction('En voilà un beau test !'); </script>
```

Ou :

```
<script>
function myFunction(arg) {
alert('Votre argument : ' + arg); }
myFunction(prompt('Que souhaitez-vous passer en argument à la fonction ?'));
</script>
```

Ou encore avec des arguments multiples :

```
<script>
function moar(first, second) {
// On peut maintenant utiliser les variables « first » et « second » comme on le souhaite :
alert('Votre premier argument : ' + first);
alert('Votre deuxième argument : ' + second);
}
moar(
  prompt('Entrez votre premier argument :'),
  prompt('Entrez votre deuxième argument :')
); </script>
```

6.3. Les valeurs de retour

Une fonction peut retourner une seule valeur, stockée dans ue variable :

```
<script>
function sayHello() {
return 'Bonjour !'; // L'instruction « return » suivie d'une valeur, cette dernière est donc renvoyée par la fonction (il ne peut pas y en avoir d'autres)
}
alert(sayHello());
</script>
```

6.4. Les fonctions anonymes (bases)

Elles supposent la structure suivante, sans nom :

```
<script>
function (arguments) {
// Le code de votre fonction anonyme
} </script>
```

Une fonction anonyme peut être utilisée, entre autres, par le biais d'une variable :

```
<script>
var sayHello = function() {
alert('Bonjour !');
};
sayHello(); </script>
```

FICHE 7.
Les objets et les tableaux (1/2)

7.1. Les objets

Les variables contiennent des objets, qui peuvent être des nombres, des chaînes de caractères ou des booléens. Mais le Javascript n'est pas un langage orienté objet (C++, C# ou Java), mais un langage orienté objet par prototype.

Les objets contiennent trois choses :
- un constructeur
- des propriétés
- des méthodes.

Par exemple :

```
<script>
var myString = 'Ceci est une chaîne de caractères'; // On crée un objet String
alert(myString.length); // On affiche le nombre de caractères, au moyen de la propriété « length »
alert(myString.toUpperCase()); // On récupère la chaîne en majuscules, avec la méthode toUpperCase(), l'inverse étant la méthode toLowerCase()
</script>
```

7.2. Les tableaux

Après `Number`, `String` et `Boolean`, `Array` est un 4e objet natif de Javascript.

Un tableau, ou plutôt un array en anglais, est une variable qui contient plusieurs valeurs, appelées items. Chaque item est accessible au moyen d'un indice (index en anglais) et dont la numérotation commence à partir de 0.

```
<script>
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume'];
// Le contenu se définit entre crochets, avec une virgule entre chaque valeur.
// La chaîne 'Rafael' correspond à l'indice 0, 'Mathilde' à l'indice 1...
alert(myArray[1]); // Affiche : « Laurence »
</script>
```

On peut modifier une valeur :

```
<script>
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume'];
myArray[1] = 'Paul';
alert(myArray[1]); // Affiche : « Paul »
</script>
```

7.3. Opérations sur les tableaux

On peut ajouter des items avec la méthode `push()` :

```
<script>
var myArray = ['Rafael', 'Mathilde'];
myArray.push('Ahmed'); // Ajoute « Ahmed » à la fin du tableau
myArray.push('Jérôme', 'Guillaume'); // Ajoute « Jérôme » et « Guillaume » à la fin du tableau
</script>
```

La méthode `unshift()` fonctionne comme `push()`, excepté que les items sont ajoutés au début du tableau. Les méthodes `shift()` et `pop()` retirent respectivement le premier et le dernier élément du tableau.

```
<script>
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume'];
myArray.shift(); // Retire « Rafael »
myArray.pop(); // Retire « Guillaume »
alert(myArray); // Affiche « Mathilde,Ahmed,Jérôme »
</script>
```

On peut découper une chaîne de caractères en tableau avec `split()` :

```
<script>
var cousinsString = 'Jérôme Guillaume Paul',
cousinsArray = cousinsString.split(' '); // Avec les espaces, on a trois items
alert(cousinsString);
alert(cousinsArray);
</script>
```

On fait l'inverse avec `join()` :

```
<script>
var cousinsString_2 = cousinsArray.join('-');
alert(cousinsString_2); </script>
```

7.4. Parcourir un tableau

On peut parcourir un tableau avec `for` :

```
<script>
var myArray = ['Rafael', 'Mathilde', 'Ahmed', 'Jérôme', 'Guillaume']; // La length est de 5
for (var i = 0, c = myArray.length; i < c; i++) { // On crée la variable c pour que la condition ne soit pas trop lourde en caractères
alert(myArray[i]); // On affiche chaque item, l'un après l'autre, jusqu'à la longueur totale du tableau
}
</script>
```

FICHE 8.
Les objets et les tableaux (2/2)

8.1. Les objets littéraux

On peut remplacer l'indice par un identifiant. Dans ce cas on crée un objet (dans l'exemple family). Les identifiants créés (self, sister...) sont des propriétés, avec plusieurs possibilités d'affichage (ce qui convient à toutes les propriétés, également pour `length` par exemple). On peut ajouter des données (avec une méthode différente que pour un tableau).

```
<script>
var family = {
self: 'Rafael',
sister: 'Mathilde',
brother: 'Ahmed',
cousin_1: 'Jérôme',
cousin_2: 'Guillaume'
};
var id = 'sister';
alert(family[id]); // Affiche : « Mathilde »
alert(family.brother); // Affiche : « Ahmed »
alert(family['self']); // Affiche : « Rafael »
family['uncle'] = 'Pauline'; // On ajoute une donnée, avec un identifiant.
family.aunt = 'Karim'; // On peut ajouter aussi de cette manière.
alert(family.uncle); </script>
```

8.2. Parcourir un objet avec for in

On ne peut pas parcourir l'objet avec `for`, parce `for` s'occupe d'incrémenter des variables numériques. Là on fournit une variable-clé pour le parcours

```
<script>
var family = {
self: 'Rafael',
sister: 'Mathilde',
brother: 'Ahmed',
cousin_1: 'Jérôme',
cousin_2: 'Guillaume'
};
for (var id in family) { // On stocke l'identifiant dans « id » pour parcourir
l'objet « family »
alert(family[id]);
} </script>
```

8.3. Exercice

Demandez les prénoms aux utilisateurs et stockez-les dans un tableau. Pensez à la méthode `push()`. À la fin, il faudra afficher le contenu du tableau, avec `alert()`, seulement si le tableau contient des prénoms ; en effet, ça ne sert à rien de l'afficher s'il ne contient rien. Pour l'affichage, séparez chaque prénom par un espace. Si le tableau ne contient rien, faites-le savoir à l'utilisateur, toujours avec `alert()`.

Code utilisé précédemment :

```
<script>
var nicks = '', nick;
while (true) {
nick = prompt('Entrez un prénom :');
if (nick) {
nicks += nick + ' '; // Ajoute le nouveau prénom ainsi qu'un espace
} else {
break; // On quitte la boucle
}
}
alert(nicks); // Affiche les prénoms à la suite
</script>
```

Correction

```
<script>
var prenoms = [],
  prenom;
while (prenom = prompt('Entrez un prénom :')) {
  prenoms.push(prenom); // Ajoute le nouveau prénom ainsi qu'un espace
}
if (prenoms.length > 0) {
  alert(prenoms.join(' '));
} else {
  alert('Il n\'y a aucun prénom en mémoire');
}
</script>
```

FICHE 9.
Modélisation de pages DHTML (bases)

Le DOM (*Document Object Model*) est une interface de programmation (ou API, *Application Programming Interface*) pour les documents XML et HTML. Via le Javascript, le DOM permet d'accéder au code du document ; on va alors pouvoir modifier des éléments du code HTML.

Contrairement à ce qui a été vu avant, `alert()` n'est pas vraiment une fonction, mais une méthode qui appartient à l'objet `window`, qui est implicite (il y a en fait très peu de variables globales). Les deux lignes suivantes signifient la même chose :

```
<script>
   alert('Hello world !');
   window.alert('Hello world !');
</script>
```

L'objet document est un sous-objet de window. L'objet document possède trois méthodes principales : `getElementById()`, `getElementsByTagName()` et `getElementsByName()`.

Avec `getElementById()` :

```
<div id="myDiv"><p>Un peu de texte <a>et un lien</a></p></div>
<script>
var div = document.getElementById('myDiv');
   alert(div); </script>
```

On nous dit alors que `div` est un objet de type `HTMLDivElement`. Cela fonctionne.

Avec `getElementsByTagName()`, on récupère les éléments sous forme de tableau :

```
<script>
   var divs = document.getElementsByTagName('div');
   for (var i = 0, c = divs.length ; i < c ; i++) {
          alert('Element n° ' + (i + 1) + ' : ' + divs[i]);
   } </script>
```

On parcourt le tableau avec une boucle pour récupérer les éléments.

Avec `getElementsByName()`, on récupère les éléments par `name`, dans les formulaires.

On peut aussi utiliser `querySelector()`, qui renvoie le premier élément trouvé correspondant au sélecteur CSS spécifié, ou `querySelectorAll()`, qui renvoie tous les éléments (sous forme de tableau) correspondant au sélecteur CSS spécifié entre parenthèses :

```
<div id="menu"><div class="item"><span>Élément 1</span><span>Élément 2</span></div>
   <div class="publicite"><span>Élément 3</span><span>Élément 4</span></div></div>
<div id="contenu"><span>Introduction au contenu de la page...</span></div>
<script>
   var query = document.querySelector('#menu .item span'),
   queryAll = document.querySelectorAll('#menu .item span');
   alert(query.innerHTML); // Affiche : "Élément 1"
   alert(queryAll.length); // Affiche : "2"
   alert(queryAll[0].innerHTML + ' - ' + queryAll[1].innerHTML); </script>
```

On suit le schéma suivant :

Node > Element >HTMLElement >HTMLDivElement

On peut jouer sur les attributs d'une balise HTML avec l'objet `Element` et `getAttribute()` et `setAttribute()`, permettant par exemple de modifier un lien :

```
    <a id="myLink" href="http://www.un_lien_quelconque.com">Un lien modifié
dynamiquement</a>
    <script>
       var link = document.getElementById('myLink');
       var href = link.getAttribute('href'); // On récupère l'attribut « href »
       alert(href);
       link.setAttribute('href', 'http://blog.crdp-versailles.fr/rimbaud/'); // on édite
    </script>
```

Cela fonctionne aussi avec :

```
    <a id="myLink" href="http://www.un_lien_quelconque.com">Un lien modifié
dynamiquement</a>
    <script>
       var link = document.getElementById('myLink');
       var href = link.href;
       alert(href);
       link.href = 'http://www.clg-rimbaud-aubergenville.ac-versailles.fr/';
    </script>
```

Par contre on ne peut pas utiliser `class`, à remplacer par `className` en Javascript. Comme `for`, à remplacer par `htmlFor` (le `for` du Javascript servant aux boucles utiles aux fonctions).

`innerHTML` permet de récupérer le code HTML enfant d'un élément en texte :

```
    <div id="myDiv">
       <p>Un peu de texte <a>et un lien</a></p>
    </div>
    <script>
       var div = document.getElementById('myDiv');
       alert(div.innerHTML);
    </script>
```

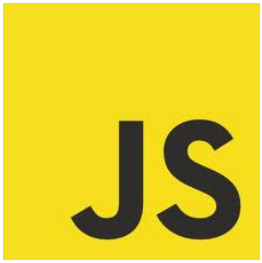
On peut alors définir un nouveau contenu :

```
    document.getElementById('myDiv').innerHTML = '<blockquote>Je mets une citation à la
place du paragraphe</blockquote>';
```

Ou encore ajouter un contenu à celui qui est en place (à éviter dans une boucle) :

```
    document.getElementById('myDiv').innerHTML += ' et <strong>une portion mise en
emphase</strong>.';
```

Dans Internet Explorer (sauf IE9), `innerText` récupère le texte, pas les balises, et `textContent` est sa version standardisée pour tous les autres navigateurs.



FICHE 10.
Modélisation de pages DHTML (récupération d'éléments HTML)

La propriété `parentNode` permet d'accéder à l'élément parent d'un élément :

```
<blockquote>
  <p id="myP">Ceci est un paragraphe !</p>
</blockquote>
<script>
  var paragraph = document.getElementById('myP');
  var blockquote = paragraph.parentNode;
</script>
```

`nodeType` et `nodeName` permettent de vérifier le type et le nom d'un nœud :

```
var paragraph = document.getElementById('myP');
alert(paragraph.nodeType + '\n\n' +
paragraph.nodeName.toLowerCase());
```

`firstChild` et `lastChild` permettent d'accéder au premier et au dernier élément d'un nœud :

```
<div>
  <p id="myP">Un peu de texte, <a>un lien</a> et <strong>une portion en emphase</strong></p>
</div>
<script>
  var paragraph = document.getElementById('myP');
  var first = paragraph.firstChild;
  var last = paragraph.lastChild;
  alert(first.nodeName.toLowerCase());
  alert(last.nodeName.toLowerCase());
</script>
```

Ou encore avec :

```
var paragraph = document.getElementById('myP');
var first = paragraph.firstChild;
var last = paragraph.lastChild;
alert(first.nodeValue);
alert(last.firstChild.data);
```

`childNodes` retourne un tableau contenant la liste des enfants d'un élément.

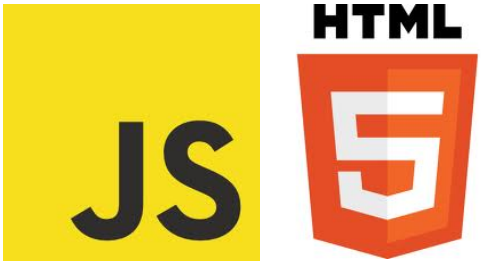
`nextSibling` et `previousSibling` permettent d'accéder à l'élément suivant, et au précédent :

```
<div>
  <p id="myP">Un peu de texte, <a>un lien</a> et <strong>une portion en emphase</strong></p>
</div>
<script>
  var paragraph = document.getElementById('myP');
  var first = paragraph.firstChild;
  var next = first.nextSibling;
  alert(next.firstChild.data); // Affiche « un lien »
</script>
```

ou :

```
<div>
  <p id="myP">Un peu de texte <a>et un lien</a></p>
</div>
<script>
  var paragraph = document.getElementById('myP');
  var child = paragraph.lastChild; // On prend le dernier enfant
  while (child) {
        if (child.nodeType === 1) { // C'est un élément HTML
        alert(child.firstChild.data);
  } else { // C'est certainement un noeud textuel
        alert(child.data);
        }
  child = child.previousSibling; // À chaque tour de boucle, on prend l'enfant précédent
  }
</script>
```

Attention ! Les espaces et retours à la ligne effectués dans le code HTML sont généralement considérés comme des nœuds textuels par les navigateurs. On utilise alors `firstElementChild`, `lastElementChild`, `nextElementSibling` et `previousElementSibling`, non supportés par IE8 et inférieur.



FICHE 11.
Modélisation de pages DHTML (création d'éléments HTML)

11.1. Créer et insérer des éléments

Avec le DOM, l'ajout d'un élément se fait en trois temps : on crée l'élément, on lui affecte des attributs, on l'insère dans le document.

Par exemple, on crée <a> :

```
var newLink = document.createElement('a');
```

On lui affecte des attributs :

```
newLink.id = 'sdz_link';
newLink.href = 'http://blog.crdp-versailles.fr/rimbaud/';
newLink.title = 'Découvrez le blog de la Classe Actu !';
newLink.setAttribute('tabindex', '10');
```

On l'insère dans le document :

```
<div><p id="myP">Un peu de texte <a>et un lien</a></p></div>
<script>
  var newLink = document.createElement('a');
  newLink.id = 'sdz_link';
  newLink.href = 'http://blog.crdp-versailles.fr/rimbaud/';
  newLink.title = 'Découvrez le blog de la Classe Actu !';
  newLink.setAttribute('tabindex', '10');
  document.getElementById('myP').appendChild(newLink); // le nouvel élément
est le dernier enfant dans le paragraphe avec id 'myP'
  var newLinkText = document.createTextNode("Le Tonnerre de Rimbaud");
  newLink.appendChild(newLinkText); // ces deux lignes pour ajouter le texte
</script>
```

11.2. Cloner, remplacer, supprimer

Pour cloner un élément, on utilise `cloneNode()`, et on choisit avec (`true`) ou sans (`false`) ses enfants et ses attributs.

Pour remplacer un élément par un autre, on utilise `replaceChild()`, avec deux paramètres, le nouvel élément et l'élément qu'on veut remplacer :

```
<div><p id="myP">Un peu de texte <a>et un lien</a></p></div>
<script>
  var link = document.getElementsByTagName('a')[0];
  var newLabel= document.createTextNode('et un hyperlien');
  link.replaceChild(newLabel, link.firstChild);
</script>
```

Pour supprimer un élément, on utilise `removeChild()`, avec le nœud enfant à retirer :

```
var link = document.getElementsByTagName('a')[0];
link.parentNode.removeChild(link);
```

Pour vérifier la présence d'éléments enfant, on utilise `hasChildNodes()` :

```
<div><p id="myP">Un peu de texte <a>et un lien</a></p></div>
<script>
   var paragraph = document.getElementsByTagName('p')[0];
   alert(paragraph.hasChildNodes()); // Affiche true
</script>
```

Pour insérer un élément avant un autre, on utilise `insertBefore()` :

```
<p id="myP">Un peu de texte <a>et un lien</a></p>
<script>
   var paragraph = document.getElementsByTagName('p')[0];
   var emphasis = document.createElement('em'),
   emphasisText = document.createTextNode(' en emphase légère ');
   emphasis.appendChild(emphasisText);
   paragraph.insertBefore(emphasis, paragraph.lastChild);
</script>
```

Exercice 1.

Passez ce code HTML en script :

```
<div id="divTP1">
Le <strong>World Wide Web Consortium</strong>, abrégé par le sigle <strong>W3C</strong>, est un <a href="http://fr.wikipedia.org/wiki/Organisme_de_normalisation" title="Organisme de normalisation">organisme de standardisation</a> à but non-lucratif chargé de promouvoir la compatibilité des technologies du <a href="http://fr.wikipedia.org/wiki/World_Wide_Web" title="World Wide Web">World Wide Web</a>.
</div>
```

Exercice 2.

Passez le code HTML en script, en utilisant une boucle for :

```
<div id="divTP2">
<p>Langages basés sur ECMAScript :</p>
<ul>       <li>JavaScript</li>
           <li>JScript</li>
           <li>ActionScript</li>
           <li>EX4</li>            </ul>          </div>
```

Exercice 3.

Passez le code HTML en script :

```
<div id="divTP4">
<form enctype="multipart/form-data" method="post" action="upload.php">
<fieldset>
<legend>Uploader une image</legend>
<div style="text-align: center">
<label for="inputUpload">Image à uploader :</label>
<input type="file" name="inputUpload" id="inputUpload" />
<br /><br />
<input type="submit" value="Envoyer" />
</div>
</fieldset></form></div>
```

FICHE 12.
Les événements

Plusieurs exemples d'événements :

`click`	Cliquer (appuyer puis relâcher) sur l'élément
`dblclick`	Double-cliquer sur l'élément
`mouseover`	Faire entrer le curseur sur l'élément
`mouseout`	Faire sortir le curseur de l'élément
`mousedown`	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
`mouseup`	Relâcher le bouton gauche de la souris sur l'élément
`mousemove`	Faire déplacer le curseur sur l'élément
`keydown`	Appuyer (sans relâcher) sur une touche de clavier sur l'élément
`keyup`	Relâcher une touche de clavier sur l'élément
`keypress`	Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
`focus`	« Cibler » l'élément
`blur`	Annuler le « ciblage » de l'élément
`change`	Changer la valeur d'un élément spécifique aux formulaires (`input`, `checkbox`, etc.)
`select`	Sélectionner le contenu d'un champ de texte (`input`, `textarea`, etc.)

Deux événements spécifiques à <form> :

`submit`	Envoyer le formulaire
`reset`	Réinitialiser le formulaire

Quelques exemples :

```
<span onclick="alert('Hello !');">Cliquez-moi !</span>
```

ou :

```
<span onclick="alert('Voici le contenu de l\'élément que vous avez
cliqué :\n\n' + this.innerHTML);">Cliquez-moi !</span>
```

ou :

```
<input id="input" type="text" size="50" value="Cliquez ici !"
onfocus="this.value='Appuyez maintenant sur votre touche de tabulation.';"
onblur="this.value='Cliquez ici !';"/>
<br /><br/>
<a href="#" onfocus="document.getElementById('input').value = 'Vous avez
maintenant le focus sur le lien, bravo !';">Un lien bidon</a>
```

ou (sans `return false`) :

```
<a href="http://blog.crdp-versailles.fr/rimbaud/" onclick="alert('Vous avez
cliqué!');">Cliquez-moi !</a>
```

ou (avec `return false`) :

```
<a href="http://blog.crdp-versailles.fr/rimbaud/" onclick="alert('Vous avez
cliqué !'); return false;">Cliquez-moi !</a>
```

ou (lien créé seulement pour l'événement `onclick`, sans `href`) :

```
<a href="#" onclick="alert('Vous avez cliqué !'); return false;">
Cliquez-moi !
</a>
```

FICHE 13.
Les formulaires

Dans `<input>`, on utilise la propriété `value`.

```
<input id="text" type="text" size="60" value="Vous n'avez pas le
focus !" />
<script>
  var text = document.getElementById('text');
  text.addEventListener('focus', function(e) {
        e.target.value = "Vous avez le focus !";
  }, true);
  text.addEventListener('blur', function(e) {
        e.target.value = "Vous n'avez pas le focus !";
  }, true);
</script>
```

Pour désactiver un champ de texte :

```
<input id="text" type="text" />
<script>
  var text = document.getElementById('text');
  text.disabled = true;
</script>
```

On peut utiliser `checked` pour des boutons radio :

```
<label><input type="radio" name="check" value="1" />Case n°1</label><br />
<label><input type="radio" name="check" value="2" />Case n°2</label><br />
<label><input type="radio" name="check" value="3" />Case n°3</label><br />
<label><input type="radio" name="check" value="4" />Case n°4</label>
<br /><br />
<input type="button" value="Afficher la case cochée" onclick="check();" />
<script>
  function check() {
  var inputs = document.getElementsByTagName('input'),
  inputsLength = inputs.length;
  for (var i = 0 ; i < inputsLength ; i++) {
        if (inputs[i].type == 'radio' && inputs[i].checked) {
        alert('La case cochée est la n°'+ inputs[i].value);
        }
  }
  }
</script>
```

Ou `selectedIndex` pour des listes déroulantes :

```
<select id="list">
  <option>Sélectionnez votre sexe</option>
  <option>Homme</option>
  <option>Femme</option>
</select>
<script>
  var list = document.getElementById('list');
  list.addEventListener('change', function() {
  // On affiche le contenu de l'élément <option> ciblé par la propriété selectedIndex
        alert(list.options[list.selectedIndex].innerHTML);
  }, true);
</script>
```

L'élément `<form>` possède deux méthodes intéressantes : `submit()` pour effectuer l'envoi du formulaire, `reset()` pour réinitialiser le formulaire :

```
<form id="myForm">
  <input type="text" value="Entrez un texte" /><br /><br />
  <input type="submit" value="Submit !" />
  <input type="reset" value="Reset !" />
</form>
<script>
  var myForm = document.getElementById('myForm');
  myForm.addEventListener('submit', function(e) {
        alert('Vous avez envoyé le formulaire !\n\nMais celui-ci a été bloqué pour que vous ne changiez pas de page.');
        e.preventDefault();
  }, true);
  myForm.addEventListener('reset', function(e) {
        alert('Vous avez réinitialisé le formulaire !');
  }, true);
</script>
```

Les méthodes `focus()` et `blur()` pour donner ou retirer le focus sur un événement.

```
<input id="text" type="text" value="Entrez un texte" /><br /><br />
<input type="button" value="Donner le focus"
  onclick="document.getElementById('text').focus();" /><br />
<input type="button" value="Retirer le focus"
  onclick="document.getElementById('text').blur();" />
```

Avec `select()`, on sélectionne le texte :

```
<input id="text" type="text" value="Entrez un texte" /><br /><br />
<input type="button" value="Sélectionner le texte"
  onclick="document.getElementById('text').select();" />
```