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>
```

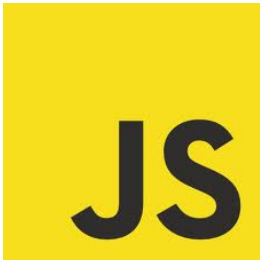
On peut alors définir un nouveau contenu :

```
    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.

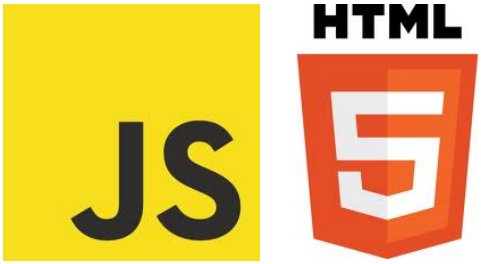
`nextSibling` et `previousSibling` permettent d'accéder à l'élément suivant, et au précédent :

```
<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();" />
```

FICHE 14.
Exemple d'un formulaire d'inscription

Head :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Inscription</title>
<style type="text/css">
body { padding-top: 50px; }
.form_col {
display: inline-block;
margin-right: 15px;
padding: 3px 0px;
width: 200px;
min-height: 1px;
text-align: right; }
input {
padding: 2px;
border: 1px solid #CCC;
-moz-border-radius: 2px;
-webkit-border-radius: 2px;
border-radius: 2px;
outline: none; }
input:focus {
border-color: rgba(82, 168, 236, 0.75);
-moz-box-shadow: 0 0 8px rgba(82, 168, 236, 0.5);
-webkit-box-shadow: 0 0 8px rgba(82, 168, 236, 0.5);
box-shadow: 0 0 8px rgba(82, 168, 236, 0.5); }
.correct {
border-color: rgba(68, 191, 68, 0.75); }
.correct:focus {
border-color: rgba(68, 191, 68, 0.75);
-moz-box-shadow: 0 0 8px rgba(68, 191, 68, 0.5);
-webkit-box-shadow: 0 0 8px rgba(68, 191, 68, 0.5);
box-shadow: 0 0 8px rgba(68, 191, 68, 0.5); }
.incorrect {
border-color: rgba(191, 68, 68, 0.75); }
.incorrect:focus {
border-color: rgba(191, 68, 68, 0.75);
-moz-box-shadow: 0 0 8px rgba(191, 68, 68, 0.5);
-webkit-box-shadow: 0 0 8px rgba(191, 68, 68, 0.5);
box-shadow: 0 0 8px rgba(191, 68, 68, 0.5); }
.tooltip {
display: inline-block;
margin-left: 20px;
padding: 2px 4px;
border: 1px solid #555;
background-color: #CCC;
-moz-border-radius: 4px;
-webkit-border-radius: 4px;
border-radius: 4px; }
</style>
</head>
```

Body :

```
<body>
<form id="myForm">
<span class="form_col">Sexe :</span>
<label><input name="sex" type="radio" value="H" />Homme</label>
<label><input name="sex" type="radio" value="F" />Femme</label>
<span class="tooltip">Vous devez sélectionnez votre sexe</span><br /><br />
<label class="form_col" for="lastName">Nom :</label>
<input name="lastName" id="lastName" type="text" />
<span class="tooltip">Un nom ne peut pas faire moins de 2 caractères</span><br /><br />
<label class="form_col" for="firstName">Prénom :</label>
<input name="firstName" id="firstName" type="text" />
<span class="tooltip">Un prénom ne peut pas faire moins de 2 caractères</span><br /><br />
<label class="form_col" for="age">Âge :</label>
<input name="age" id="age" type="text" />
<span class="tooltip">L'âge doit être compris entre 5 et 140</span><br /><br />
<label class="form_col" for="login">Pseudo :</label>
<input name="login" id="login" type="text" />
<span class="tooltip">Le pseudo ne peut pas faire moins de 4 caractères</span><br /><br />
<label class="form_col" for="pwd1">Mot de passe :</label>
<input name="pwd1" id="pwd1" type="password" />
<span class="tooltip">Le mot de passe ne doit pas faire moins de 6 caractères</span><br /><br />
<label class="form_col" for="pwd2">Mot de passe (confirmation) :</label>
<input name="pwd2" id="pwd2" type="password" />
<span class="tooltip">Le mot de passe de confirmation doit être identique à celui d'origine</span><br /><br />
<label class="form_col" for="country">Pays :</label>
<select name="country" id="country">
<option value="none">Sélectionnez votre pays de résidence</option>
<option value="en">Angleterre</option>
<option value="us">États-Unis</option>
<option value="fr">France</option></select>
<span class="tooltip">Vous devez sélectionner votre pays de résidence</span><br /><br />
<span class="form_col"></span>
<label><input name="news" type="checkbox" /> Je désire recevoir la newsletter chaque mois.</label><br /><br />
<span class="form_col"></span>
<input type="submit" value="M'inscrire" /> <input type="reset" value="Réinitialiser le formulaire" />
</form>
```

Script :

```
<script>
(function() { // On utilise une IEF pour ne pas polluer l'espace global
// Fonction de désactivation de l'aff des « tooltips »
function deactivateTooltips() {
var spans = document.getElementsByTagName('span'),
spansLength = spans.length;
for (var i = 0 ; i < spansLength ; i++) {
if (spans[i].className == 'tooltip') {
spans[i].style.display = 'none';
} } }
// La fonction ci-dessous permet de récupérer la « tooltip » qui correspond à notre input
function getTooltip(element) {
while (element = element.nextSibling) {
if (element.className === 'tooltip') {
```

```
      return element;
      } } return false; }
      // Fonctions de vérification du formulaire, elles
renvoient « true » si tout est OK
      var check = {}; // On met toutes nos fonctions dans un
objet littéral
      check['sex'] = function() {
      var sex = document.getElementsByName('sex'),
      tooltipStyle = getTooltip(sex[1].parentNode).style;
      if (sex[0].checked || sex[1].checked) {
      tooltipStyle.display = 'none';
      return true;
      } else {
      tooltipStyle.display = 'inline-block';
      return false; } };
      check['lastName'] = function(id) {
      var name = document.getElementById(id),
      tooltipStyle = getTooltip(name).style;
      if (name.value.length >= 2) {
      name.className = 'correct';
      tooltipStyle.display = 'none';
      return true;
      } else {
      name.className = 'incorrect';
      tooltipStyle.display = 'inline-block';
      return false; } };
      check['firstName'] = check['lastName']; // La fonction
pour le prénom est la même que celle du nom
      check['age'] = function() {
      var age = document.getElementById('age'),
      tooltipStyle = getTooltip(age).style,
      ageValue = parseInt(age.value);
      if (!isNaN(ageValue) && ageValue >= 5 && ageValue <=
140) {
      age.className = 'correct';
      tooltipStyle.display = 'none';
      return true;
      } else {
      age.className = 'incorrect';
      tooltipStyle.display = 'inline-block';
      return false; } };
      check['login'] = function() {
      var login = document.getElementById('login'),
      tooltipStyle = getTooltip(login).style;
      if (login.value.length >= 4) {
      login.className = 'correct';
      tooltipStyle.display = 'none';
      return true;
      } else {
      login.className = 'incorrect';
      tooltipStyle.display = 'inline-block';
      return false; } };
      check['pwd1'] = function() {
      var pwd1 = document.getElementById('pwd1'),
      tooltipStyle = getTooltip(pwd1).style;
      if (pwd1.value.length >= 6) {
      pwd1.className = 'correct';
      tooltipStyle.display = 'none';
      return true;
      } else {
      pwd1.className = 'incorrect';
      tooltipStyle.display = 'inline-block';
      return false; } };
      check['pwd2'] = function() {
      var pwd1 = document.getElementById('pwd1'),
      pwd2 = document.getElementById('pwd2'),
      tooltipStyle = getTooltip(pwd2).style;
      if (pwd1.value == pwd2.value && pwd2.value != '') {
      pwd2.className = 'correct';
      tooltipStyle.display = 'none';
      return true;
      } else {
      pwd2.className = 'incorrect';
      tooltipStyle.display = 'inline-block';
      return false; } };
      check['country'] = function() {
      var country = document.getElementById('country'),
      tooltipStyle = getTooltip(country).style;
      if (country.options[country.selectedIndex].value !=
'none')
      {
      tooltipStyle.display = 'none';
      return true;
      } else {
      tooltipStyle.display = 'inline-block';
      return false; } };
      // Mise en place des événements
      (function() { // Utilisation d'une fonction anonyme
pour éviter les variables globales.
      var myForm = document.getElementById('myForm'),
      inputs = document.getElementsByTagName('input'),
      inputsLength = inputs.length;
      for (var i = 0 ; i < inputsLength ; i++) {
      if (inputs[i].type == 'text' || inputs[i].type ==
      'password') {
      inputs[i].onkeyup = function() {
      check[this.id](this.id); // « this » représente l'input
actuellement modifié
      };
      }
      }
      myForm.onsubmit = function() {
      var result = true;
      for (var i in check) {
      result = check[i](i) && result;
      }
      if (result) {
      alert('Le formulaire est bien rempli.');
      }
      return false;
      };
      myForm.onreset = function() {
      for (var i = 0 ; i < inputsLength ; i++) {
      if (inputs[i].type == 'text' || inputs[i].type ==
      'password') {
      inputs[i].className = '';
      }
      }
      deactivateTooltips();
      };
      })();
      // Maintenant que tout est initialisé, on peut
désactiver les « tooltips »
      deactivateTooltips();
      })();
      </script>
      </body>
      </html>
```

FICHE 15.
Tableaux et Caractères (nouvelles notions)

15.1. Construire un tableau :

```
function Person(nick, age, sex, parent, work, friends) {
   this.nick = nick;
   this.age = age;
   this.sex = sex;
   this.classe = classe;
   this.work = work;
   this.friends = friends; }
```

On met la première lettre de la fonction du constructeur en majuscule. On utilise `this` pour construire, ce qui permet de définir les propriétés de la fonction `Person` dans l'exemple.

On peut ensuite ajouter des variables :

```
var seb = new Person('Rafael', 15, 'm', '3e 5', 'Javascripteur', []);
var lau = new Person('Mathilde', 12, 'f', '5e 6', 'Webmaster', []);
alert(seb.nick); // Affiche : « Rafael »
alert(lau.nick); // Affiche : « Mathilde »
```

On peut aussi changer des variables :

```
var seb = new Person('Rafael', 15, 'm', '3e 5', 'Javascripteur', []);
seb.nick = 'Bastien'; // On change le prénom
seb.age = 14; // On change l'âge
alert(seb.nick + ' a ' + seb.age + 'ans'); // Affiche : « Bastien a 14 ans »
```

On peut ajouter des méthodes, par exemple un « ami » dans le tableau :

```
var seb = new Person('Rafael', 15, 'm', '3e 5', 'Javascripteur', []);
// On ajoute un ami dans le tableau « friends »
seb.friends.push(new Person('Jérôme', 13, 'm', '3e 5',
'Javascripteur aussi', []));
alert(seb.friends[0].nick); // Affiche : « Jérôme »
```

Ou :

```
function Person(nick, age, sex, parent, work, friends) {
   this.nick = nick;     this.age = age;       this.sex = sex;
   this.parent = parent; this.work = work;     this.friends = friends;
   this.addFriend = function(nick, age, sex, parent, work, friends)
          {
          this.friends.push(new Person(nick, age, sex, parent, work, friends));
          };     }
```

15.2. Gérer un tableau :

On utilise l'objet `Array` :

```
var myArray = new Array('valeur1', 'valeur2', …, 'valeurX');
```

La méthode `concat()` pour concaténer :

```
var myArray = ['test1',
'test2'].concat(['test3', 'test4']);
alert(myArray); //retourne ['test1', 'test2',
'test3', 'test4']
```

La méthode `foreach()` pour parcourir :

```
var myArray = ["C'est", "un", "test"];
myArray.forEach(function(value, index, array)
{
    alert(
    'Index : ' + index
    + '\n' +
    'Valeur : ' + value
    );
});
```

La méthode `indexOf()` pour rechercher un élément :

```
var element2 = ['test'],
myArray = ['test', element2];
alert(myArray.indexOf(element2)); // Affiche : 1
```

La méthode `reverse()` pour inverser :

```
var myArray = [1, 2, 3, 4, 5];
myArray.reverse();
alert(myArray); // Affiche : 5,4,3,2,1
```

La méthode `sort()` pour ordonner alphabétiquement :

```
var myArray = [3, 1, 5, 10, 4, 2];
myArray.sort();
alert(myArray); // Affiche : 1,10,2,3,4,5
```

La même, mais pour respecter l'ordre numérique :

```
var myArray = [3, 1, 5, 10, 4, 2];
myArray.sort(function (a, b) {
if (a < b) {
return -1;
} else if (a > b) {
return 1;
} else {
return 0;
}
});
alert(myArray); // Affiche : 1,2,3,4,5,10
```

La méthode `slice()` pour extraire une partie d'un tableau (avec deux arguments, le premier, inclus, auquel on commence l'extrait, le deuxième, facultatif, non inclus, auquel on termine l'extrait) :

```
var myArray = [1, 2, 3, 4, 5];
alert(myArray.slice(1, 3)); // Affiche : 2,3
alert(myArray.slice(2)); // Affiche : 3,4,5
```

ou :

```
var myArray = [1, 2, 3, 4, 5];
alert(myArray.slice(1, -1)); // Affiche : 2,3,4
```

La méthode `splice()` pour remplacer une partie d'un tableau :

```
var myArray = [1, 2, 3, 4, 5];
var result = myArray.splice(1, 2);
// On retire 2 éléments à partir de l'index 1
alert(myArray); // Affiche : 1,4,5
alert(result); // Affiche : 2,3
```

Et aussi :

`push()` : ajoute un ou plusieurs éléments à la fin du tableau (un argument par élément ajouté) et retourne la nouvelle taille de ce dernier.

`pop()` : retire et retourne le dernier élément d'un tableau.

`unshift()` : ajoute un ou plusieurs éléments au début du tableau (un argument par élément ajouté) et retourne la nouvelle taille de ce dernier.

`shift()` : retire et retourne le premier élément d'un tableau.

15.3. Gérer les chaînes de caractères

Pour passer d'une chaîne de caractères primitive ou d'un tableau primitif à un objet :

```
var myString = "Chaîne de caractères primitive";
var myRealString = new String("Chaîne");

var myArray = []; // Tableau primitif
var myRealArray = new Array();

var myObject = {}; // Objet primitif
var myRealObject = new Object();

var myBoolean = true; // Booléen primitif
var myRealBoolean = new Boolean("true");

var myNumber = 42; // Nombre primitif
var myRealNumber = new Number("42");
```

On supprime les espaces d'une chaîne de caractères avec `trim()`

On fait des recherches dans une chaîne avec `indexOf()`

On extrait une chaîne avec `substring()`, `substr()` et `slice()`

On coupe une chaîne en un tableau avec `split()`

FICHE 16.
Expressions régulières (Regex)

Le regex :

```
var myRegex = /contenu_à_rechercher/;
var regex_2 = /contenu_\/_contenu/;
```

Cela permet de rechercher des mots :

```
if (/dragon/.test('Il a tué le dragon qui garde le trésor !')) {
        alert('Ça semble parler de dragon');
  } else {
        alert('Pas de dragon à l\'horizon');       }
```

On utilise `i` si on ne veut pas s'occuper de la casse :

```
if (/Dragon/i.test(' Il a tué le dragon qui garde le trésor !')) {
        alert('Ça semble parler de dragon');
  } else {
        alert('Pas de dragon à l\'horizon');       }
```

La barre verticale `|` permet le OU :

```
if (/Dragon|Licorne/i.test('Il a ensuite chevauché la licorne !')) {
        alert('Ça semble parler de trucs fantastiques');
  } else {
        alert('Pas de fantastique à l\'horizon');         }
```

Les symboles `^` et `$` permettent de dire respectivement si la chaîne recherchée commence la chaîne (au début) ou la termine (à la fin).

Pour dire que plusieurs caractères conviennent à un endroit de la recherche, les crochets `[]` :

```
if (/gr[ao]s/.test('Que ce soit gras ou gros')) {
        alert('L\'un des deux, oui');
  } else {
        alert('Ni gras ni gros');       }
```

On peut convenir d'un intervalle de lettre (de a à j : `[a-j]`), ou encore `[a-zA-Z]` pour ne pas tenir compte de la casse, ou `[0-9]` pour tous les chiffres, ou encore `[a-z0-9]`. On peut ignorer des lettres avec `[^aeiou]` ou `[^b-f]`. Mais il faut préciser les caractères accentuées, et utiliser `i` pour la casse (qui fonctionne aussi pour les caractères accentués) : `[^a-zâäàéèùêëîïôöçñ]/i`.

Le point désigne un caractère quelconque `/gr.s/`

On peut utiliser des symboles quantificateurs :
`?` : ce symbole indique que le caractère qui le précède peut apparaître 0 ou 1 fois ;
`+` : ce symbole indique que le caractère qui le précède peut apparaître 1 ou x fois ;
`*` : ce symbole indique que le caractère qui le précède peut apparaître 0, 1 ou x fois.

On peut préciser plutôt le nombre de fois :
`{n}` : le caractère est répété n fois ;
`{n,m}` : le caractère est répété de n à m fois. Par exemple, si on a `{0, 5}`, le caractère peut être présent de 0 à 5 fois ;
`{n,}` : le caractère est répété de n fois à l'infini.

Pour une adresse mail :

```
var email = prompt("Entrez votre adresse e-mail :","0780506b@ac-versailles.fr");
if (/^[a-z0-9._-]+@[a-z0-9._-]+\.[a-z]{2,6}$/.test(email)) {
          alert("Adresse e-mail valide !");
   } else {
          alert("Adresse e-mail invalide !"); }
```

On déclare une RegExp de manière primitive ou en créant un objet :

```
var myRegex1 = /^Dragon$/i;
var myRegex2 = new RegExp("^Dragon$", "i");
```

On peut récupérer un mois de naissance :

(avec `\S` : trouve un caractère qui n'est pas un caractère blanc, et + pour la répétition)

(par défaut on enregistre d'abord dans $1, puis $2, etc., jusqu'à $9)

```
var birth = 'Je suis né en mars';
/^Je suis né en (\S+)$/.exec(birth);
alert(RegExp.$1); // Affiche : « mars »
```

Pour les éléments d'une adresse mail :

```
var email = prompt("Entrez votre adresse e-mail :", "0780506b@ac-versailles.fr");
if (/^([a-z0-9._-]+)@([a-z0-9._-]+)\.([a-z]{2,6})$/.test(email)) {
   alert('Partie locale : ' + RegExp.$1 + '\nDomaine : ' +
   RegExp.$2 + '\nExtension : ' + RegExp.$3);
} else {
   alert('Adresse e-mail invalide !');        }
```

On peut remplacer une chaîne avec regex :

```
var sentence = 'Je m\'appelle Rafael';
var result = sentence.replace(/Rafael/, 'Paul');
alert(result); // Affiche : « Je m'appelle Paul »
```

On utilise `g` si une occurrence doit être remplacée à chaque fois :

```
var sentence = 'Il s\'appelle Rafael. Rafael écrit un tutoriel.';
var result = sentence.replace(/Rafael/g, 'Paul');
alert(result); // Il s'appelle Paul. Paul écrit un tutoriel.
```

On peut modifier le format d'une date :

```
var date = '01/11/2013';
date = date.replace(/^(\d{2})\/
(\d{2})\/(\d{4})$/, 'Le $2/$1/$3');
alert(date); // Le 11/01/2013
```

Ou encore :

```
   var total = 'J\'ai 25 euros en
liquide.';
   total = total.replace(/euros?/, '€');
   alert(total);// J'ai 25 € en liquide
```

`\d` trouve un caractère décimal (un chiffre)
`\D` trouve un caractère qui n'est pas décimal (pas un chiffre)
`\s` trouve un caractère blanc
`\S` trouve un caractère qui n'est pas blanc
`\w` trouve un caractère « de mot » (lettre ou `_`)
`\W` trouve un caractère qui n'est pas « de mot »
`\n` trouve un retour à la ligne
`\t` trouve une tabulations
`\b` trouve une limite de mot
`\B` ne trouve pas de limite de mot

On peut utiliser une fonction pour le remplacement :
- Le paramètre `str` contient la portion de texte trouvée par la regex ;
- Les paramètres `p*` contiennent les portions capturées par les parenthèses ;
- Le paramètre `offset` contient la position de la portion de texte trouvée ;
- Le paramètre `s` contient la totalité de la chaîne.

FICHE 17.
Les données numériques

L'objet `Number` possède des propriétés accessibles directement sans aucune instanciation (on appelle cela des propriétés propres à l'objet constructeur). Elles sont au nombre de cinq, et sont données ici à titre informatif, car leur usage est peu courant :

`NaN` : cette propriété signifie *Not A Number* et permet, généralement, d'identifier l'échec d'une conversion de chaîne de caractères en un nombre. À noter que cette propriété est aussi disponible dans l'espace global. Passer par l'objet Number pour y accéder n'a donc que peu d'intérêt, surtout qu'il est bien rare d'utiliser cette propriété, car on lui préfère la fonction `isNaN()`, plus fiable.

`MAX_VALUE` : cette propriété représente le nombre maximum pouvant être stocké dans une variable en Javascript. Cette constante peut changer selon la version du Javascript.

`MIN_VALUE` : identique à la constante `MAX_VALUE`, mais pour la valeur minimale.

`POSITIVE_INFINITY` : il s'agit ici d'une constante représentant l'infini positif. On peut l'obtenir en résultat d'un calcul si on divise une valeur positive par 0. Cependant, son utilisation est rare, car on lui préfère la fonction `isFinite()`, plus fiable.

`NEGATIVE_INFINITY` : identique à `POSITIVE_INFINITY`, pour l'infini négatif. On peut obtenir cette constante en résultat d'un calcul si on divise une valeur négative par 0.

Objet Math

```
alert(Math.PI); // Affiche la valeur du nombre pi
alert(Math.E); // Affiche la valeur du nombre d'Euler
```

On peut arrondir et tronquer :

```
Math.floor(33.15); // Retourne : 33
Math.floor(33.95); // Retourne : 33
Math.floor(34); // Retourne : 34
```

```
Math.ceil(33.15); // Retourne : 34
Math.ceil(33.95); // Retourne : 34
Math.ceil(34); // Retourne : 34
```

```
Math.round(33.15); // Retourne : 33
Math.round(33.95); // Retourne : 34
Math.round(34); // Retourne : 34
```

Calculer puissance et racine carrée

```
Math.pow(3, 2);
// Le premier paramètre est la base, le deuxième est l'exposant
// Ce calcul donne donc : 3 * 3 = 9

Math.sqrt(9); // Retourne : 3
```

Calculer cosinus, sinus :

```
Math.cos(Math.PI); // Retourne : -1
Math.sin(Math.PI); // Retourne : environ 0
```

Retrouver maximale et minimale :

```
Math.max(42, 4, 38, 1337, 105); // Retourne : 1337
Math.min(42, 4, 38, 1337, 105); // Retourne : 4
```

Choisir un nombre au hasard :

```
function rand(min, max, integer) {
if (!integer) {
return Math.random() * (max - min) + min;
} else {
return Math.floor(Math.random() * (max - min + 1) + min);
}
}
```

Convertir :

```
var myString = '08';
alert(parseInt(myString)); // Affiche : 0
alert(parseInt(myString, 10)); // Affiche : 8
```

FICHE 18.
La gestion du temps

On utilise l'objet `Date` :

```
new Date();
new Date(timestamp);
new Date(dateString);
new Date(année, mois, jour [, heure, minutes, secondes, millisecondes ]);
```

Huit méthodes :

`getFullYear()` : renvoie l'année sur 4 chiffres ;
`getMonth()` : renvoie le mois (0 à 11) ;
`getDate()` : renvoie le jour du mois (1 à 31) ;
`getDay()` : renvoie le jour de la semaine (0 à 6, la semaine commence le dimanche) ;
`getHours()` : renvoie l'heure (0 à 23) ;
`getMinutes()` : renvoie les minutes (0 à 59) ;
`getSeconds()` : renvoie les secondes (0 à 59) ;
`getMilliseconds()` : renvoie les millisecondes (0 à 999).
`getTime()` renvoie le timestamp de la date de votre objet ;
`setTime()` vous permet de modifier la date de votre objet en passant en unique paramètre un timestamp.

Avec par exemple :

```
var myDate = new Date('Sat, 04 May 1991 20:00:00 GMT+02:00');
alert(myDate.getMonth()); // Affiche : 4
alert(myDate.getDay()); // Affiche : 6
```

On choisit un intervalle avant l'exécution d'une fonction avec `setTimeout()` :

```
setTimeout(myFunction, 2000); // myFunction sera exécutée au bout de 2 secondes
```

On annule une fonction temporelle avec `clearTimeout()` ou `clearInterval()` :

```
<button id="myButton">Annuler le compte à rebours</button>
<script>
  (function() {
  var button = document.getElementById('myButton');
  var timerID = setTimeout(function() { // On crée notre compte à rebours
        alert("Vous n'êtes pas très réactif vous !");
  }, 5000);
  button.onclick = function() {
  clearTimeout(timerID); // Le compte à rebours est annulé
alert("Le compte à rebours a bien été annulé."); // Et on prévient l'utilisateur
}; })(); </script>
```

Ou encore :

```
<button id="myButton">Annuler le compte à rebours (5s)</button>
<script>
  (function() {
        var button = document.getElementById('myButton'),
        timeLeft = 5;
        var timerID = setTimeout(function() { // On crée notre compte à rebours
        clearInterval(intervalID);
        button.innerHTML = "Annuler le compte à rebours (0s)";
        alert("Vous n'êtes pas très réactif vous !");
        }, 5000);
  var intervalID = setInterval(function() { // On met en place l'intervalle
pour afficher la progression du temps
        button.innerHTML = "Annuler le compte à rebours (" + - -timeLeft + "s)";
        }, 1000);
  button.onclick = function() {
        clearTimeout(timerID); // On annule le compte à rebours
        clearInterval(intervalID); // Et l'intervalle
        alert("Le compte à rebours a bien été annulé.");
        };
  })();
</script>
```

On peut gérer la durée pour des animations :

```
<img id="myImg" src="images/circuit_html.jpg" style="opacity: 1" />
<script>
  var myImg = document.getElementById('myImg');
  function anim() {
        var s = myImg.style,
        result = s.opacity = parseFloat(s.opacity) - 0.1;
        if ( result > 0.2 ) {
              setTimeout(anim, 50); // La fonction anim() fait appel à elle-
même si elle n'a pas terminé son travail
              }
        }
  anim(); // Et on lance la première phase de l'animation
</script>
```

FICHE 19.
Les images

On utilise l'objet `Image` :

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

On peut créer une forme de **lightbox** :

Head avec le CSS3 :

```
<html>
<head>
  <style type="text/css">
  #overlay {
        display : none; <!-- Par défaut, on cache l'overlay -->
        position: absolute;
        top: 0; left: 0;
        width: 100%; height: 100%;
        text-align: center; <!-- Pour centrer l'image que l'overlay contiendra -->
        background-color: rgba(0,0,0,0.6); <!-- On applique un background de couleur noire et d'opacité 0.6. Il s'agit d'une propriété CSS3. -->
  }
  #overlay img {
        margin-top: 100px;
  }
  p {
        margin-top: 300px;
        text-align: center;
  }
</style>
</head>
```

Body :

```
<body>
  <a href="images/circuit_html.jpg" title="Afficher l'image originale"><img src="images/circuit_html_min.jpg" alt="Miniature" /></a> <!-- on pourrait avoir plusieurs images en portfolio -->
  <div id="overlay"></div>
```

Script :

```
    <script>
      var links = document.getElementsByTagName('a'),
      linksLen = links.length;
      for (var i = 0 ; i < linksLen ; i++) {
            links[i].onclick = function() { // Vous pouvez très bien utiliser le DOM-2
            displayImg(this); // On appelle notre fonction pour afficher les images et
on lui passe le lien concerné
            return false; // Et on bloque la redirection
            };
      }
      function displayImg(link) {
            var img = new Image(),
            overlay = document.getElementById('overlay');
            img.onload = function() {
                  overlay.innerHTML = '';
                  overlay.appendChild(img);
            };
            img.src = link.href;
            overlay.style.display = 'block';
            overlay.innerHTML = '<span>Chargement en cours...</span>';
      }
      document.getElementById('overlay').onclick = function() {
            this.style.display = 'none';
      };
    </script>
    </body>
    </html>
```

FICHE 20.
(nécessite un serveur)
AJAX & XMLHttpRequest

20.1. Qu'est-ce que l'AJAX ?

AJAX est l'acronyme d'*Asynchronous Javascript and XML*, ce qui, transcrit en français, signifie « Javascript et XML asynchrones ». Derrière ce nom se cache un ensemble de technologies destinées à réaliser de rapides mises à jour du contenu d'une page Web, sans qu'elles nécessitent le moindre rechargement visible par l'utilisateur de la page Web. Cela permet par exemple l'auto-complétion (mots possibles lors d'une recherche) ou encore la sauvegarde automatique.

Plusieurs formats existent pour le transfert de données : un fichier texte, HTML, XML pour de nombreuses données (eXtensible Markup Language), JSON pour de petits transferts (JavaScript Object Notation). Pour le texte et le HTML, il n'y a pas de traitement, alors que c'est différent avec XML et JSON.

XML :

```
<?xml version="1.0" encoding="utf-8"?>
<table>
<line>
<cel>Ligne 1 - Colonne 1</cel><cel>Ligne 1 - Colonne 2</cel><cel>Ligne 1 - Colonne 3</cel>
</line>
<line>
<cel>Ligne 2 - Colonne 1</cel><cel>Ligne 2 - Colonne 2</cel><cel>Ligne 2 - Colonne 3</cel>
</line>
<line>
<cel>Ligne 3 - Colonne 1</cel><cel>Ligne 3 - Colonne 2</cel><cel>Ligne 3 - Colonne 3</cel>
</line>
</table>
```

Avec en entête : `<?php header('Content-type: text/xml'); ?>`

JSON :

```
{
Membre1: {
   posts: 6230,
   inscription: '22/08/2003'
},
Membre2 {
   posts: 200,
   inscription: '04/06/2011'
}
}
```

Avec JSON, on peut utiliser la méthode `parse()`, qui prend en paramètre la chaîne de caractères à analyser et retourne le résultat sous forme d'objet JSON, la méthode `stringify()`, qui permet de faire l'inverse (elle prend en paramètre un objet JSON et retourne son équivalent sous forme de chaîne de caractères). Le JSON permet aussi d'envoyer des données (`json_encode()` et `json_decode()` depuis PHP 5.2).

20.2. XMLHttpRequest

On instancie un objet XMLHttpRequest :

```
var xhr = new XMLHttpRequest();
```

On prépare la requête avec la méthode `open()`, qui prend 5 arguments :

> Le premier argument contient la méthode d'envoi des données, les trois méthodes principales sont `GET`, `POST` et `HEAD`.
>
> Le deuxième argument est l'URL à laquelle vous souhaitez soumettre votre requête, par exemple : 'http://mon_site_web.com'.
>
> Le troisième argument est un booléen facultatif dont la valeur par défaut est `true`. À `true`, la requête sera de type asynchrone, à `false` elle sera synchrone (la différence est expliquée plus tard).
>
> Les deux derniers arguments sont à spécifier en cas d'identification nécessaire sur le site Web (à cause d'un .htaccess par exemple). Le premier contient le nom de l'utilisateur, tandis que le deuxième contient le mot de passe.

Avec :

```
xhr.open('GET', 'http://mon_site_web.com/ajax.php');
xhr.send(null);
```

Avec `GET` :

```
var value1 = encodeURIComponent(value1),
value2 = encodeURIComponent(value2);
xhr.open('GET', 'http://mon_site_web.com/ajax.php?param1=' + value1
+ '&param2=' + value2);
```

Avec `POST` :

```
xhr.open('POST', 'http://mon_site_web.com/ajax.php');
xhr.setRequestHeader("Content-Type", "application/x-www-formurlencoded");
xhr.send('param1=' + value1 + '&param2=' + value2);
```

Pour récupérer des données,

```
xhr.onreadystatechange = function() {
if (xhr.readyState == xhr.DONE) { // La constante DONE appartient à l'objet
XMLHttpRequest, elle n'est pas globale
// Votre code…
}
};
```

```
var xhr = new XMLHttpRequest();
xhr.open('HEAD', 'http://mon_site_web.com/', false);
xhr.send(null);
alert(xhr.getResponseHeader('Content-type')); // Affiche : « text/html;
charset=utf-8 »
```

FICHE 21.
(nécessite un serveur)
AJAX : Upload via une frame

Les iframes peuvent être utiles pour l'upload de fichiers.

On crée le frame :

```
var frame = document.getElementById('myFrame');
frame = frame.contentDocument || frame.document;
```

On peut charger une iframe en changeant l'URL :

```
document.getElementById('myFrame').src = 'request.php?nick=Thunderseb';
```

Avec target et un formulaire :

```
<form id="myForm" method="post" action="request.php" target="myFrame">
<div>
<!-- formulaire -->
<input type="submit" value="Envoyer" />      </div>        </form>
<iframe src="#" name="myFrame" id="myFrame"></iframe>
```

On détecte le chargement avec load :

```
<iframe src="file.html" name="myFrame" id="myFrame" onload="trigger()"></iframe>
<script>
   function trigger() {
          var frame = document.getElementById('myFrame');
          frame = frame.contentDocument || frame.document;
          alert(frame.body.textContent);
   } </script>
```

Avec une fonction de callback :

avec dans la page mère :

```
<iframe src="file.html" name="myFrame"
id="myFrame"></iframe>
<script>
   function trigger() {
   var frame =
document.getElementById('myFrame');
   frame = frame.contentDocument ||
frame.document;
   alert('Page chargée !');
   }
</script>
```

et dans l'iframe :

```
<script>
window.top.window.trigger(); // On
appelle ici notre fonction de callback
</script>
<p>Lorem ipsum dolor sit amet,
consectetuer adipiscing elit.<br />
Suspendisse molestie suscipit arcu.</p>
```

Avec du PHP (nécessite un serveur) :

dans la page HTML :

```
    <form id="myForm" method="post"
action="request.php" target="myFrame">
    <div>
    <label for="nick">Votre pseudo :</label>
    <input type="text" id="nick"
name="nick" />
    <input type="button" value="Envoyer"
onclick="sendForm();" />
    </div>
    </form>
    <iframe src="#" name="myFrame"
id="myFrame"></iframe>
    <script>
    function sendForm() {
    var nick =
document.getElementById("nick").value;
    if (nick) { // Si c'est OK
    document.getElementById("myForm").submit(
); // On envoie le formulaire
    }  }
    function receiveData(data) {
    alert('Votre pseudo est "' + data + '"');
    }  </script>
```

dans la page PHP (request.php) :

```
    <script>
    window.top.window.receiveData("<?php echo
htmlentities($_POST['nick']); ?>");
    </script>
```

Pour créer une interface d'upload :

dans la page HTML :

```
    <form id="uploadForm"
enctype="multipart/form-data"
    action="upload.php" target="uploadFrame"
method="post">
    <label for="uploadFile">Image :</label>
    <input id="uploadFile" name="uploadFile"
type="file" />
    <br /><br />
    <input id="uploadSubmit" type="submit"
value="Upload !" />
    </form>
    <div id="uploadInfos">
    <div id="uploadStatus">Aucun upload en
cours</div>
    <iframe id="uploadFrame"
name="uploadFrame"></iframe>
    </div>
    function uploadEnd(error, path) {
    if (error === 'OK') {
    document.getElementById('uploadStatus').i
nnerHTML = '<a
    href="' + path + '">Upload done !
</a><br /><br /><a href="' + path +
    '"><img src="' + path + '" /></a>';
    } else {
    document.getElementById('uploadStatus').i
nnerHTML = error;
    }
    }
    document.getElementById('uploadForm').add
EventListener('submit',
    function() {
    document.getElementById('uploadStatus').i
nnerHTML =
    'Loading...';
    }, true);
```

dans la page upload.php :

```
    <?php
    $error = NULL;
    $filename = NULL;
    if (isset($_FILES['uploadFile']) &&
    $_FILES['uploadFile']['error'] === 0) {
    $filename = $_FILES['uploadFile']
['name'];
    $targetpath = getcwd() . '/' . $filename;
// On stocke le chemin où enregistrer le
fichier
    // On déplace le fichier depuis le
répertoire temporaire vers $targetpath
    if
(@move_uploaded_file($_FILES['uploadFile']
['tmp_name'],
    $targetpath)) { // Si ça fonctionne
    $error = 'OK';
    } else { // Si ça ne fonctionne pas
    $error = "Échec de l'enregistrement !";
    }
    } else {
    $error = 'Aucun fichier réceptionné !';
    }
    // Et pour finir, on écrit l'appel vers
la fonction uploadEnd :
    ?>
    <script>
    window.top.window.uploadEnd("<?php echo
$error; ?>", "<?php echo
    $filename; ?>");
    </script>
```

FICHE 22.
(nécessite un serveur)
AJAX : Dynamix Script Loading (DSL)

Par exemple :

dans HTML :

```
<script>
  function sendDSL() {
        var scriptElement = document.createElement('script');
        scriptElement.src = 'dsl_script.js';
        document.body.appendChild(scriptElement);
  }
  function receiveMessage(message) {
        alert(message);
  }
</script>
<p><button type="button" onclick="sendDSL()">Exécuter le script</button></p>
```

et le fichier dsl_script.js :

```
receiveMessage('Ce message est envoyé par le serveur !');
```

Ou avec du PHP :

le fichier HTML :

```
<script>
  function sendDSL() {
        var scriptElement = document.createElement('script');
        scriptElement.src = 'dsl_script.php?nick=' + prompt('Quel est votre
pseudo ?');
        document.body.appendChild(scriptElement);
  }
  function receiveMessage(message) {
        alert(message);
  }
</script>
<p><button type="button" onclick="sendDSL()">Exécuter le script</button></p>
```

le fichier dsl_script.php :

```
<?php header("Content-type: text/javascript"); ?>
var string = 'Bonjour <?php echo $_GET['nick'] ?> !';
receiveMessage(string);
```

Ou encore avec JSON :

dans HTML :

```
<script>
  function sendDSL() {
        var scriptElement = document.createElement('script');
        scriptElement.src = 'dsl_script_json.php';
        document.body.appendChild(scriptElement);
  }
  function receiveMessage(json) {
        var tree = '', nbItems, i;
        for (node in json) {
              tree += node + "\n";
              nbItems = json[node].length;
              for (i=0; i<nbItems; i++) {
                    tree += '\t' + json[node][i] + '\n';
              }
        }
        alert(tree);
  }
</script>
<p><button type="button" onclick="sendDSL()">Charger le JSON</button></p>
```

dans dsl_script_json.php :

```
<?php
header("Content-type: text/javascript");
echo 'var softwares = {
  "Adobe": [
  "Acrobat",
  "Dreamweaver",
  "Photoshop",
  "Flash"
  ],
  "Mozilla": [
  "Firefox",
  "Thunderbird",
  "Lightning"
  ],
  "Microsoft": [
  "Office",
  "Visual C# Express",
  "Azure"
  ]
};';
?>
receiveMessage(softwares);
```

FICHE 23.
(nécessite un serveur)
Exemple d'un système d'auto-complétion

(avec towns.txt)

Le fichier PHP (search.php) :

```
<?php
$data = unserialize(file_get_contents('towns.txt')); // Récupération de la liste complète des villes
$dataLen = count($data);
sort($data); // On trie les villes dans l'ordre alphabétique
$results = array(); // Le tableau où seront stockés les résultats de la recherche
// La boucle ci-dessous parcourt tout le tableau $data, jusqu'à un maximum de 10 résultats
for ($i = 0 ; $i < $dataLen && count($results) < 10 ; $i++)
{
if (stripos($data[$i], $_GET['s']) === 0) { // Si la valeur commence par les mêmes caractères que la recherche
array_push($results, $data[$i]); // On ajoute alors le résultat à la liste à retourner
}
}
echo implode('|', $results); // Et on affiche les résultats séparés par une barre verticale |
?>
```

Le fichier HTML :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>TP : Un système d'auto-complétion</title>
<style>
body {
margin: auto;
padding-top: 80px;
width: 300px;
font-family: sans-serif;
font-size: 0.8em;
background-color: #F5F5F5;
}
input[type="submit"] {
margin-left: 10px;
width: 50px; height: 26px;
}
#search {
padding: 2px 4px;
width: 220px; height: 22px;
border: 1px solid #AAA;
}
#search:hover, #search:focus {
border-color: #777;
}
#results {
display: none;
width: 228px;
border: 1px solid #AAA;
border-top-width: 0;
background-color: #FFF;
}
#results div {
width: 220px;
padding: 2px 4px;
text-align: left;
border: 0;
background-color: #FFF;
}
#results div:hover, .result_focus {
background-color: #DDD !important;
}
</style>
</head>
<body>
<input id="search" type="text" autocomplete="off" />
<div id="results"></div>
<script>
(function() {
var searchElement = document.getElementById('search'),
results = document.getElementById('results'),
selectedResult = -1, // Permet de savoir quel résultat est sélectionné : -1 signifie « aucune sélection »
previousRequest, // On stocke notre précédente requête dans cette variable
```

```
    previousValue = searchElement.value;
    // On fait de même avec la précédente
valeur
    function getResults(keywords) { //
Effectue une requête et récupère les résultats
    var xhr = new XMLHttpRequest();
    xhr.open('GET', './search.php?s='+
encodeURIComponent(keywords));
    xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && xhr.status ==
200) {
    displayResults(xhr.responseText);
    }
    };
    xhr.send(null);
    return xhr;
    }
    function displayResults(response) {
    // Affiche les résultats d'une requête
    results.style.display = response.length ?
'block' : 'none';
    // On cache le conteneur si on n'a pas de
résultats
    if (response.length) { // On ne modifie
les résultats que si on en a obtenu
    response = response.split('|');
    var responseLen = response.length;
    results.innerHTML = ''; // On vide les
résultats
    for (var i = 0, div ; i < responseLen ;
i++) {
    div =
    results.appendChild(document.createElemen
t('div'));
    div.innerHTML = response[i];
    div.onclick = function() {
    chooseResult(this);
    };
    }
    }
    }
    function chooseResult(result) { //
Choisit un des résultats d'une requête et gère
tout ce qui y est attaché
    searchElement.value = previousValue =
result.innerHTML; // On change le contenu du
champ de recherche et on enregistre en tant
que précédente valeur
    results.style.display = 'none'; // On
cache les résultats
    result.className = ''; // On supprime
l'effet de focus
    selectedResult = -1; // On remet la
sélection à zéro
    searchElement.focus(); // Si le résultat
a été choisi par le biais d'un clic, alors le
focus est perdu, donc on le réattribue
    }

    searchElement.onkeyup = function(e) {
    e = e || window.event; // On n'oublie pas
la compatibilité pour IE
    var divs =
results.getElementsByTagName('div');
    if (e.keyCode == 38 && selectedResult >
-1) { // Si la touche pressée est la flèche «
haut »
    divs[selectedResult--].className = '';
    if (selectedResult > -1) { // Cette
condition évite une modification de
childNodes[-1], qui n'existe pas, bien entendu
    divs[selectedResult].className =
'result_focus';
    }
    }
    else if (e.keyCode == 40 &&
selectedResult < divs.length -
    1) { // Si la touche pressée est la
flèche « bas »
    results.style.display = 'block'; // On
affiche les résultats
    if (selectedResult > -1) { // Cette
condition évite une modification de
childNodes[-1], qui n'existe pas, bien entendu
    divs[selectedResult].className = '';
    }
    divs[++selectedResult].className =
'result_focus';
    }
    else if (e.keyCode == 13 &&
selectedResult > -1) { // Si la touche pressée
est « Entrée »
    chooseResult(divs[selectedResult]);
    }
    else if (searchElement.value !=
previousValue) { // Si le contenu du champ de
recherche a changé
    previousValue = searchElement.value;
    if (previousRequest &&
previousRequest.readyState < 4) {
    previousRequest.abort(); // Si on a
toujours une requête en cours, on l'arrête
    }
    previousRequest =
getResults(previousValue); // On stocke la
nouvelle requête
    selectedResult = -1; // On remet la
sélection à zéro à chaque caractère écrit
    }
    };
    })();
    </script>
    </body>
    </html>
```

FICHE 24.
L'élément Canvas (1/2)

Avec `<canvas>`

Par exemple un rectangle doré et un bleu, avec un effacement :

```
    <!DOCTYPE html>
    <html>
    <head>
    <title>CANVAS</title>
    <meta charset="utf-8" />
    </head>
    <body>
    <canvas id="canvas" width="150" height="150">
    <p>Désolé, votre navigateur ne supporte pas Canvas. Mettez-vous à jour</p>
    </canvas>
    <script>
    var canvas = document.querySelector('#canvas');
    var context = canvas.getContext('2d');
    context.lineWidth = "3";
    context.strockStyle = "gold";
    context.strockRect(50, 35, 50, 80);
    context.fillStyle = "rgba(23, 145, 167, 0.5)";
    context.fillRect(40, 25, 40, 40);
    context.clearRect(45, 40, 30, 10);
    </script>
    </body>
    </html>
```

On peut créer des rectangles, des arcs, mais aussi des chemins et courbes de Béziers :

```
    <script>
    var canvas = document.querySelector('#canvas');
    var context = canvas.getContext('2d');
    context.strokeStyle = "rgb(23, 145, 167)";
    context.beginPath();
    context.moveTo(20, 20); // 1er point
    context.lineTo(130, 20); // 2e point
    context.lineTo(130, 50); // 3e
    context.lineTo(75, 130); // 4e
    context.lineTo(20, 50); // 5e
    context.closePath(); // On relie le 5e au 1er
    context.stroke();
    </script>
```

Un smiley avec la méthode arc (x, y, rayon, angleDepart, angleFin, sensInverse) :

```
    <script>
    var canvas = document.querySelector('#canvas');
    var context = canvas.getContext('2d');
    context.beginPath(); // Le cercle extérieur
    context.arc(75, 75, 50, 0, Math.PI * 2); // Ici le calcul est simplifié
    context.stroke();
    context.beginPath(); // La bouche, un arc de cercle
    context.arc(75, 75, 40, 0, Math.PI); // Ici aussi
    context.fill();
    context.beginPath(); // L'oeil gauche
    context.arc(55, 70, 20, (Math.PI / 180) * 220, (Math.PI / 180) * 320);
    context.stroke();
    context.beginPath(); // L'oeil droit
    context.arc(95, 70, 20, (Math.PI / 180) * 220, (Math.PI / 180) * 320);
    context.stroke();
    </script>
```

Un triangle rempli de rouge :

```
<script>
var canvas = document.querySelector('#canvas');
var context = canvas.getContext('2d');
context.fillStyle = "#990000";
context.beginPath();
context.moveTo(20, 20); // 1er point
context.lineTo(500, 100); // 2e point
context.lineTo(250, 300); // 3e
context.closePath(); // On relie le 3e au 1er
context.fill();
</script>
```

Le logo JavaScript avec des courbes de Béziers :

```
<script>
var canvas = document.querySelector('#canvas');
var context = canvas.getContext('2d');
context.beginPath();
context.moveTo(131, 119);
context.bezierCurveTo(131, 126, 126, 131, 119, 131);
context.lineTo(30, 131);
context.bezierCurveTo(23, 131, 18, 126, 18, 119);
context.lineTo(18, 30);
context.bezierCurveTo(18, 23, 23, 18, 30, 18);
context.lineTo(119, 18);
context.bezierCurveTo(126, 18, 131, 23, 131, 30);
context.lineTo(131, 119);
context.closePath();
context.fillStyle = "rgb(23, 145, 167)";
context.fill();
context.font = "68px Calibri,Geneva,Arial";
context.fillStyle = "white";
context.fillText("js", 84, 115);
</script>
```

On peut intégrer une image :

```
<script>
var canvas = document.querySelector('#canvas');
var context = canvas.getContext('2d');
var circuit = new Image();
circuit.src = 'images/circuit_html.jpg';
circuit.onload = function() {
context.drawImage(circuit, 35, 35);
}
</script>
```

On peut intégrer du texte :

```
<script>
var canvas = document.querySelector('#canvas');
var context = canvas.getContext('2d');
context.fillStyle = "rgba(23, 145, 167, 1)";
context.fillRect(50, 50, 50, 50);
context.font = "bold 22pt Calibri,Geneva,Arial";
context.fillStyle = "#fff";
context.fillText("js", 78, 92);
</script>
```

FICHE 25.
L'élément Canvas (2/2)

Les extrémités des chemins peuvent être `butt` (par défaut, droites), `round` (arrondies) ou `square` (avec un carré qui dépasse de chaque côté).

`lineJoin` permet de gérer les intersections, avec `round`, `bevel` ou `mitter`.

Pour un dégradé, on crée un objet `canvasGradient` (avec `createLinearGradient()` ou `createRadialGradient()`), auquel on affile `fillStyle`.

```
var linear = context.createLinearGradient(0, 0, 0, 150);
linear.addColorStop(0, 'white');
linear.addColorStop(1, '#1791a7');
context.fillStyle = linear;
context.fillRect(20, 20, 110, 110);
```

Avec un exemple :

```
<script>
var canvas = document.querySelector('#canvas');
var context = canvas.getContext('2d');
var linear = context.createLinearGradient(0, 0, 0, 150);
linear.addColorStop(0, 'white');
linear.addColorStop(0.5, '#1791a7');
linear.addColorStop(0.5, 'orange');
linear.addColorStop(1, 'white');
context.fillStyle = linear;
context.fillRect(10, 10, 130, 130); </script>
```

Ou :

```
var radial = context.createRadialGradient(75, 75, 0, 130, 130, 150);
radial.addColorStop(0, '#1791a7');
radial.addColorStop(1, 'white');
context.fillStyle = radial;
context.fillRect(10, 10, 130, 130);
```

Ou (avec des bulles) :

```
<script>
var canvas = document.querySelector('#canvas');
var context = canvas.getContext('2d');
var radial1 = context.createRadialGradient(0, 0, 10, 100, 20, 150); // fond
radial1.addColorStop(0, '#ddf5f9');
radial1.addColorStop(1, '#ffffff');
var radial2 = context.createRadialGradient(75, 75, 10, 82, 70, 30); // bulle orange
radial2.addColorStop(0, '#ffc55c');
radial2.addColorStop(0.9, '#ffa500');
radial2.addColorStop(1, 'rgba(245,160,6,0)');
var radial3 = context.createRadialGradient(105, 105, 20, 112, 120, 50); // bulle turquoise
radial3.addColorStop(0, '#86cad2');
radial3.addColorStop(0.9, '#61aeb6');
radial3.addColorStop(1, 'rgba(159,209,216,0)');
context.fillStyle = radial1;
context.fillRect(10, 10, 130, 130);
context.fillStyle = radial2;
context.fillRect(10, 10, 130, 130);
context.fillStyle = radial3;
context.fillRect(10, 10, 130, 130);
</script>
```

On restaure l'état pour revenir aux coordonnées 0 du plan :

```
    <script>
    var canvas =
document.querySelector('#canvas');
    var context =
canvas.getContext('2d');
    context.save();
    context.translate(40, 40);
    context.fillStyle = "teal";
    context.fillRect(0, 0, 50, 50);
    context.restore();
    context.fillStyle = "orange";
    context.fillRect(0, 0, 50, 50);
    </script>
```

On exerce une rotation :

```
    <script>
    var canvas =
document.querySelector('#canvas');
    var context =
canvas.getContext('2d');
    context.translate(75,75);
    context.fillStyle = "teal";
    context.rotate((Math.PI / 180) * 45);
    context.fillRect(0, 0, 50, 50);
    context.fillStyle = "orange";
    context.rotate(Math.PI / 2);
    context.fillRect(0, 0, 50, 50);
    context.fillStyle = "teal";
    context.rotate(Math.PI / 2);
    context.fillRect(0, 0, 50, 50);
    context.fillStyle = "orange";
    context.rotate(Math.PI / 2);
    context.fillRect(0, 0, 50, 50);
    </script>
```

On crée une animation :

```
    <script>
     window.requestAnimFrame =
(function(){
     return window.requestAnimationFrame
|| // La forme standardisée
    window.webkitRequestAnimationFrame ||
// Pour Chrome et Safari
    window.mozRequestAnimationFrame    ||
// Pour Firefox
    window.oRequestAnimationFrame      ||
// Pour Opera
    window.msRequestAnimationFrame     ||
// Pour Internet Explorer
    function(callback){
// Pour les mauvais
    window.setTimeout(callback, 1000 /
60);
                    };
          })();
    window.onload = function() {
    var canvas =
document.querySelector('#canvas');
    var context =
canvas.getContext('2d');
    function draw(angle) {
    context.save();
    context.clearRect(0, 0, 150, 150);
    context.translate(75,75);
    context.fillStyle = "teal";
    context.rotate((Math.PI / 180) * (45
+ angle)); // ne pas oublier le décalage
    context.fillRect(0, 0, 50, 50);
    context.fillStyle = "orange";
    context.rotate(Math.PI / 2);
    context.fillRect(0, 0, 50, 50);
    context.fillStyle = "teal";
    context.rotate(Math.PI / 2);
    context.fillRect(0, 0, 50, 50);
    context.fillStyle = "orange";
    context.rotate(Math.PI / 2);
    context.fillRect(0, 0, 50, 50);
    context.restore();
    angle = angle + 2; // on augmente le
décalage
    if (angle >= 360) angle = 0; // on
remet le décalage à 0, puisqu'on a fait
le tour du cercle
    window.requestAnimFrame(function()
{ draw(angle) });
    }
    draw(0); // premier appel
    };
    </script>
```

Avec l'élément Canvas, on peut créer des jeux, comme des RPG (à voir sur le site http://rpgjs.com/wiki/)

FICHE 26.
L'API File

L'API File permet de récupérer le nom d'un fichier sur l'ordinateur de l'usager :

```
<input id="file" type="file" multiple />
<script>
  document.querySelector('#file').onchange = function() {
        alert(this.files[0].name);
  };
</script>
```

L'API File permet de lire le contenu d'un fichier .txt :

```
<input id="file" type="file" />
<script>
  var fileInput = document.querySelector('#file');
  fileInput.onchange = function() {
        var reader = new FileReader();
        reader.onload = function() {
              alert('Contenu du fichier "' + fileInput.files[0].name +
'":\n\n' + reader.result);
              };
        reader.readAsText(fileInput.files[0]);
  };
</script>
```

L'API File permet de sélectionner et prévisualiser des images (avant upload) :

```
<input id="file" type="file" multiple />
<div id="prev"></div>
<script>
  (function() {
        function createThumbnail(file) {
              var reader = new FileReader();
              reader.onload = function() {
                    var imgElement = document.createElement('img');
                    imgElement.style.maxWidth = '150px';
                    imgElement.style.maxHeight = '150px';
                    imgElement.src = this.result;
                    prev.appendChild(imgElement);
              };
              reader.readAsDataURL(file);
        }
        var allowedTypes = ['png', 'jpg', 'jpeg', 'gif'],
        fileInput = document.querySelector('#file'),
        prev = document.querySelector('#prev');
        fileInput.onchange = function() {
              var files = this.files,
              filesLen = files.length,
              imgType;
              for (var i = 0 ; i < filesLen ; i++) {
                    imgType = files[i].name.split('.');
                    imgType = imgType[imgType.length - 1];
                    if(allowedTypes.indexOf(imgType) != -1) {
                          createThumbnail(files[i]);
                    }
              }
        };
  })();
</script>
```

FICHE 27.
Drag & Drop

Cet outil permet un déplacement d'objets dans la page HTML :

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Drag & Drop</title>
<style>
   .dropper {
   margin: 50px 10px 10px 50px;
   width: 400px;
   height: 250px;
   background-color: #555;
   border: 1px solid #111;
   -moz-border-radius: 10px;
   border-radius: 10px;
   -moz-transition: all 200ms linear;
   -webkit-transition: all 200ms linear;
   -o-transition: all 200ms linear;
   transition: all 200ms linear;
    }
   .drop_hover {
   -moz-box-shadow: 0 0 30px rgba(0, 0,
0, 0.8) inset;
    box-shadow: 0 0 30px rgba(0, 0, 0,
0.8) inset;
    }
```

```
.draggable {
   display: inline-block;
   margin: 20px 10px 10px 20px;
   padding-top: 20px;
   width: 80px;
   height: 60px;
   color: #3D110F;
   background-color: #822520;
   border: 4px solid #3D110F;
   text-align: center;
   font-size: 2em;
   cursor: move;
   -moz-transition: all 200ms linear;
   -webkit-transition: all 200ms linear;
   -o-transition: all 200ms linear;
   transition: all 200ms linear;
   -moz-user-select: none;
   -khtml-user-select: none;
   -webkit-user-select: none;
   user-select: none;
   }
</style>
```

```
</head>
<body>
   <div class="dropper">
   <div class="draggable">#1</div>
   <div class="draggable">#2</div>
   </div>
   <div class="dropper">
   <div class="draggable">#3</div>
   <div class="draggable">#4</div>
   </div>
<script>
(function() {
   var dndHandler = {
   draggedElement: null, // Propriété pointant vers l'élément en cours de déplacement
           applyDragEvents: function(element) {
           element.draggable = true;
           var dndHandler = this; // Cette variable est nécessaire pour que l'événement "dragstart" ci-
dessous accède facilement au namespace "dndHandler"
           element.addEventListener('dragstart', function(e) {
                   dndHandler.draggedElement = e.target; // On sauvegarde l'élément en déplacement
                   e.dataTransfer.setData('text/plain', ''); // Nécessaire pour Firefox
           }, false);      },
```

```
		applyDropEvents: function(dropper) {
			dropper.addEventListener('dragover', function(e) {
				e.preventDefault(); // On autorise le drop d'éléments
				this.className = 'dropper drop_hover'; // Et on applique le design adéquat à notre zone de drop quand un élément la survole
			}, false);
		dropper.addEventListener('dragleave', function() {
			this.className = 'dropper'; // On revient au design de base lorsque l'élément quitte la zone de drop
		});
		var dndHandler = this; // Cette variable est nécessaire pour que l'événement "drop" ci-dessous accède facilement au namespace "dndHandler"
		dropper.addEventListener('drop', function(e) {
			var target = e.target,
			draggedElement = dndHandler.draggedElement, // Récupération de l'élément concerné
			clonedElement = draggedElement.cloneNode(true); // On créé immédiatement le clone de cet élément
			while(target.className.indexOf('dropper') == -1) { // Cette boucle permet de remonter jusqu'à la zone de drop parente
				target = target.parentNode;
			}
			target.className = 'dropper'; // Application du design par défaut
			clonedElement = target.appendChild(clonedElement); // Ajout de l'élément cloné à la zone de drop actuelle
			dndHandler.applyDragEvents(clonedElement); // Nouvelle application des événements qui ont été perdus lors du cloneNode()
			draggedElement.parentNode.removeChild(draggedElement); // Suppression de l'élément d'origine
			});
		}
		};
		var elements = document.querySelectorAll('.draggable'),
		elementsLen = elements.length;
		for(var i = 0 ; i < elementsLen ; i++) {
			dndHandler.applyDragEvents(elements[i]); // Application des paramètres nécessaires aux élément déplaçables
		}
		var droppers = document.querySelectorAll('.dropper'),
		droppersLen = droppers.length;
		for(var i = 0 ; i < droppersLen ; i++) {
			dndHandler.applyDropEvents(droppers[i]); // Application des événements nécessaires aux zones de drop
		}
	})();
</script>
</body>
</html>
```