Manipuler le code HTML (Partie 2/2)

Informations sur le tutoriel

Auteurs :

- Johann Pardanaud (Nesquik69)
- Sébastien de la Marck (Thunderseb)

Difficulté :
Licence :

Télécharger en PDF (0.51M)

Historique des mises à jour

Rester informé grâce au flux RSS

- *Le 12/12/2011 à 18:57:05*
  Correction orthographique suite au report #5764
- *Le 07/12/2011 à 18:01:13*
  Correction d'une erreur de syntaxe suite au report #5752
- *Le 07/12/2011 à 14:43:39*
  Correction d'une erreur de syntaxe suite au report #5751

innerHTML a comme principale qualité d'être facile et rapide à utiliser et c'est la raison pour laquelle il est généralement privilégié par les débutants et même certains développeurs expérimentés. **innerHTML** a longtemps été une propriété non-standardisée, mais depuis HTML 5, elle est reconnue par le W3C et peut donc être utilisée.

Dans ce deuxième chapitre dédié à la manipulation du contenu, nous allons aborder la modification via DOM. On l'a déjà fait dans le premier chapitre, avec notamment **setAttribute()**, mais ici, il va s'agir de créer, de supprimer, de déplacer des éléments HTML. C'est un gros morceau du Javascript, pas toujours facile à assimiler. Vous allez me dire, si **innerHTML** suffit, pourquoi devoir s'embêter avec le DOM ? Tout simplement car le DOM est plus puissant et est nécessaire pour traiter du XML (ce que nous ferons par la suite).

Sommaire du chapitre :

- Naviguer entre les nœuds
- Créer et insérer des éléments
- Notions sur les références
- Cloner, remplacer, supprimer...
- Autres actions
- Minis-TD : recréer une structure DOM
- *Q.C.M.*

Naviguer entre les nœuds

Nous avons vu précédemment qu'on utilisait les méthodes **getElementById()** et **getElementsByTagName()** pour accéder aux éléments HTML. Une fois qu'on a atteint un élément, il est possible de se déplacer de façon un peu plus précise, avec toute une série de méthodes et de propriétés que nous allons étudier ici.

La propriété parentNode

La propriété **parentNode** permet d'accéder à l'élément parent d'un élément. Regardez le code ci-dessous :

Code : HTML - Sélectionner

```
<blockquote>
  <p id="myP">Ceci est un paragraphe !</p>
</blockquote>
```

Admettons qu'on doive accéder à **myP**, et que pour une autre raison on doive accéder à l'élément **<blockquote>**, qui est le parent de **myP**, il suffit d'accéder à myP puis à son parent, avec **parentNode** :

Code : JavaScript - Sélectionner

```
var paragraph = document.getElementById('myP');
var blockquote = paragraph.parentNode;
```

nodeType et nodeName

nodeType et **nodeName** servent respectivement à vérifier le type d'un nœud et le nom d'un nœud.

nodeType retourne un nombre, qui correspond à un type de nœud. Voici un tableau qui liste les types possibles, ainsi que leurs numéros (les types courants sont mis en gras) :

Numéro	Type de nœud
1	**Nœud élément**
2	**Nœud attribut**
3	**Nœud texte**
4	**Nœud pour passage CDATA** (relatif au XML)
5	Nœud pour référence d'entité

Numéro	Type de nœud
6	Nœud pour entité
7	Nœud pour instruction de traitement
8	Nœud pour commentaire
9	Nœud document
10	Nœud type de document
11	Nœud de fragment de document
12	Nœud pour notation

nodeName quant à lui, retourne simplement le nom de l'élément, en majuscule. Il est toutefois conseillé d'utiliser **toLowerCase()** (ou **toUpperCase()**) pour forcer un format de casse.

Code : JavaScript - Sélectionner

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

Essayer !

Lister et parcourir des nœuds enfants

firstChild et lastChild

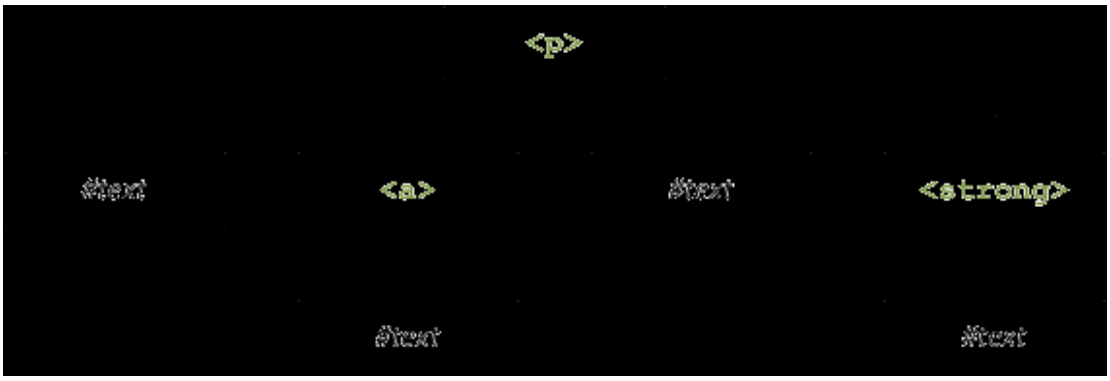
Comme leur nom le laisse présager, **firstChild** et **lastChild** servent respectivement à accéder au premier et au dernier enfant d'un nœud.

Code : HTML - Sélectionner

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
  </head>

  <body>
    <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>
  </body>
</html>
```

Essayer !

En schématisant l'élément **myP** ci-dessus, on obtient ceci :



Le premier enfant de **<p>** est un nœud textuel, alors que le dernier enfant est un élément ****.

nodeValue et data

Changeons de problème : il faut récupérer le texte du premier enfant, et le texte contenu dans l'élément ****, mais comment faire ?

Il faut soit utiliser la propriété **nodeValue** soit la propriété **data**. Si on recode le script ci-dessus, ça donne ceci :

Code : JavaScript - Sélectionner

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

alert(first.nodeValue);
alert(last.firstChild.data);
```

Essayer !

first contient le premier nœud, un nœud textuel. Il suffit de lui appliquer la propriété **nodeValue** (ou **data**) pour récupérer son contenu ; pas de difficulté ici. En revanche il y a une petite différence avec notre élément **** : vu que les attributs **nodeValue** et **data** ne s'appliquent que sur des nœuds textuels, il nous faut d'abord accéder au nœud textuel que contient notre élément , c'est-à-dire son nœud enfant. Pour cela, on utilise **firstChild**, et ensuite on récupère le contenu avec **nodeValue** ou **data**.

childNodes

La propriété **childNodes** retourne un tableau contenant la liste des enfants d'un élément. L'exemple suivant illustre le fonctionnement de cette propriété, de manière à récupérer le contenu des éléments enfants :

Code : HTML - Sélectionner

```
<body>
  <div>
    <p id="myP">Un peu de texte <a>et un lien</a></p>
```

```
  </div>

  <script>
    var paragraph = document.getElementById('myP');
    var children  = paragraph.childNodes;

    for (var i = 0, c = children.length; i < c; i++) {

      if (children[i].nodeType === 1) { // C'est un élément HTML
        alert(children[i].firstChild.data);
      } else { // C'est certainement un noeud textuel
        alert(children[i].data);
      }

    }
  </script>
</body>
```

Essayer !

nextSibling et previousSibling

nextSibling et **previousSibling** sont deux attributs qui permettent d'accéder respectivement au nœud suivant et au nœud précédent.

Code : HTML - Sélectionner

```
<body>
  <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>
</body>
```

Essayer !

Dans l'exemple ci-dessus, on récupère le premier enfant de **myP**, et sur ce premier enfant, on utilise **nextSibling**, qui permet de récupérer l'élément **<a>**. Avec ça, il est même possible de parcourir les enfants d'un élément, en utilisant une boucle **while** :

Code : HTML - Sélectionner

```
<body>
  <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; // A chaque tour de boucle, on
prend l'enfant précédent

    }
  </script>
</body>
```

Essayer !

Pour changer un peu, la boucle tourne "à l'envers", car on commence par récupérer le dernier enfant et on chemine à reculons.

Attention aux nœuds vides

En considérant le code HTML suivant, on peut penser que l'élément `<div>` ne contient que 3 enfants `<p>` :

Code : HTML - Sélectionner

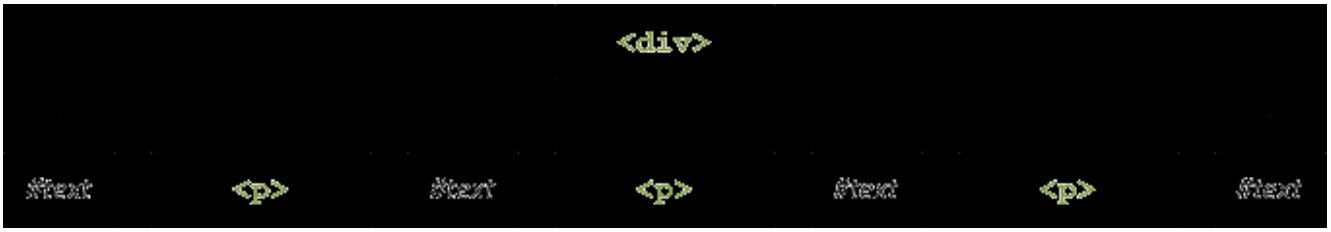
```
<div>
  <p>Paragraphe 1</p>
  <p>Paragraphe 2</p>
  <p>Paragraphe 3</p>
</div>
```

Mais attention, car le code ci-dessus est radicalement différent de celui-ci :

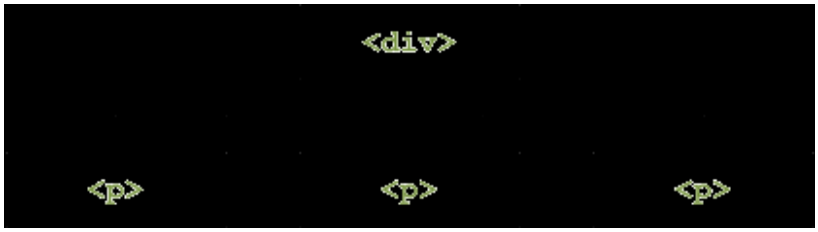
Code : HTML - Sélectionner

```
<div><p>Paragraphe 1</p><p>Paragraphe 2</p><p>Paragraphe 3</p></div>
```

En fait, les espaces blancs entre les éléments, tout comme les retours à la ligne sont considérés comme des nœuds textuels (enfin, cela dépend des navigateurs) ! Ainsi donc, si l'on schématise le premier code, on obtient ceci :



Alors que le deuxième code peut être schématisé comme ça :



Heureusement, il existe une solution à ce problème ! Les attributs **firstElementChild**, **lastElementChild**, **nextElementSibling** et **previousElementSibling** ne retournent que des éléments HTML et permettent donc d'ignorer les nœuds textuels, ils s'utilisent exactement de la même manière que les attributs de base (**firstChild**, etc...). Cependant, ces attributs ne sont pas supportés par Internet Explorer 8 et ses versions antérieures et il n'existe pas d'alternative.

Créer et insérer des éléments

Ajouter des éléments HTML

Avec DOM, l'ajout d'un élément HTML se fait en 3 temps :

1. on crée l'élément ;
2. on lui affecte des attributs ;
3. on l'insère dans le document, et ce n'est qu'à ce moment-là qu'il sera "ajouté".

Création de l'élément

La création d'un élément se fait avec la méthode **createElement()**, un sous objet de l'objet racine, c'est-à-dire `document` (dans la majorité des cas) :

Code : JavaScript - Sélectionner

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

On crée ici un nouvel élément **<a>**. Cet élément est créé, mais n'est PAS inséré dans le document, il n'est donc pas visible. Cela dit, on peut déjà "travailler" dessus, en lui ajoutant des attributs ou même des événements (que nous verrons dans le chapitre suivant).

 Si vous travaillez dans une page Web, l'élément racine sera toujours document, sauf dans le cas de frames. La création d'éléments au sein de fichiers XML sera abordée plus tard.

Affectation des attributs

Ici, c'est comme nous avons vu précédemment : on définit les attributs, soit avec **setAttribute()**, soit directement

avec les propriétés adéquates.

Code : JavaScript - Sélectionner

```
newLink.id    = 'sdz_link';
newLink.href  = 'http://www.siteduzero.com';
newLink.title = 'Découvrez le Site du Zéro !';
newLink.setAttribute('tabindex', '10');
```

Insertion de l'élément

On utilise la méthode **appendChild()** pour insérer l'élément. "append child" signifie "ajouter un enfant", ce qui signifie qu'il nous faut connaître l'élément auquel on va ajouter l'élément créé. Considérons donc le code suivant :

Code : HTML - Sélectionner

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
  </head>

  <body>
    <div>
      <p id="myP">Un peu de texte <a>et un lien</a></p>
    </div>
  </body>
</html>
```

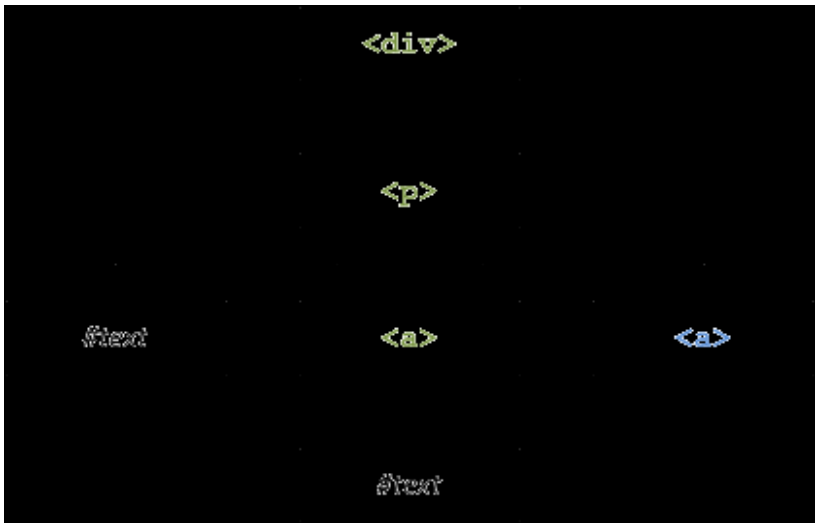
On va ajouter notre <a> dans l'élément <p> portant l'ID myP. Pour ce faire, il suffit de récupérer cet élément, et d'ajouter notre élément <a> via **appendChild()** :

Code : JavaScript - Sélectionner

```
document.getElementById('myP').appendChild(newLink);
```

Une petite explication s'impose ! Avant d'insérer notre élément <a>, la structure DOM du document ressemble à ceci :

On voit que l'élément **<a>** existe, mais n'est pas relié. C'est comme s'il était "libre" dans le document : il n'est pas encore placé. Le but est de le placer comme enfant de l'élément **myP**. La méthode **appendChild()** va alors déplacer notre **<a>** pour le placer en tant que dernier enfant de **myP** :



Cela veut dire qu'**appendChild()** insérera toujours l'élément en tant que dernier enfant, ce qui n'est pas toujours très pratique. Nous verrons après comment insérer un élément avant ou après un enfant donné.

Ajouter des nœuds textuels

L'élément a été inséré, seulement, il manque quelque chose : le contenu textuel ! La méthode **createTextNode()** sert à créer un nœud textuel (de type #text), qu'il nous suffira d'ajouter à notre élément fraîchement inséré, comme ceci :

Code : JavaScript - Sélectionner

```
var newLinkText = document.createTextNode("Le Site du Zéro");

newLink.appendChild(newLinkText);
```

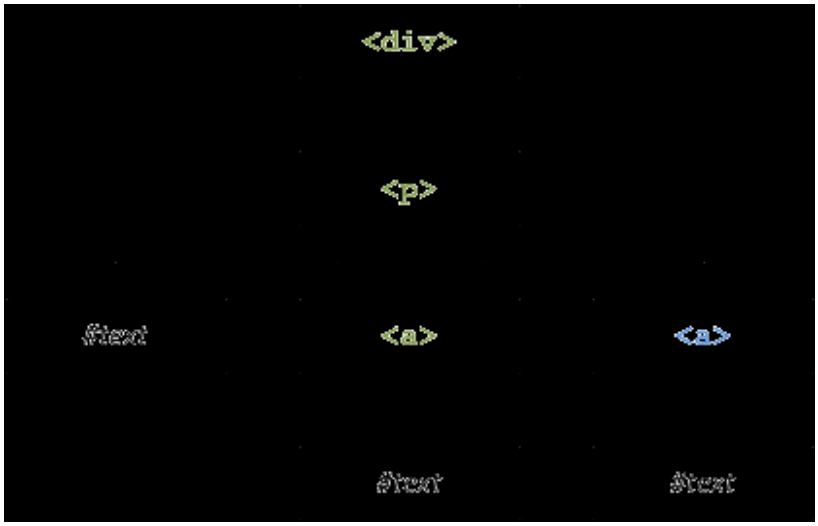
L'insertion se fait ici aussi avec **appendChild()**, sur l'élément newLink. Afin d'y voir plus clair, résumons le code :

Code : HTML - Sélectionner

```
<body>
  <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://www.siteduzero.com';
    newLink.title = 'Découvrez le Site du Zéro !';
    newLink.setAttribute('tabindex', '10');

    document.getElementById('myP').appendChild(newLink);

    var newLinkText = document.createTextNode("Le Site du Zéro");

    newLink.appendChild(newLinkText);
  </script>
</body>
```

Essayer !

Voici donc ce qu'on obtient, sous forme schématisée :



Il y a quelque chose à savoir : le fait d'insérer, via **appendChild()**, n'a aucune incidence sur l'ordre d'exécution des instructions. Cela veut donc dire qu'on peut travailler sur les éléments HTML et les nœuds textuels sans qu'ils soient au préalable insérés dans le document. Par exemple, on pourrait ordonner le code ci-dessus comme ceci :

Code : JavaScript - Sélectionner

```
var newLink = document.createElement('a');
var newLinkText = document.createTextNode("Le Site du Zéro");
```

```
newLink.id    = 'sdz_link';
newLink.href  = 'http://www.siteduzero.com';
newLink.title = 'Découvrez le Site du Zéro !';
newLink.setAttribute('tabindex', '10');

newLink.appendChild(newLinkText);

document.getElementById('myP').appendChild(newLink);
```

Ici, on commence par créer les deux éléments (le lien, et le nœud de texte), puis on affecte les variables au lien et on lui ajoute le nœud textuel. A ce stade-ci, l'élément HTML contient le nœud textuel, mais cet élément n'est pas encore inséré dans le document :

La dernière instruction insère alors le tout.

 Nous vous conseillons d'organiser votre code comme le dernier exemple ci-dessus, c'est-à-dire avec la création de tous les éléments au début, puis les différentes opérations d'affection, et enfin, l'insertion des éléments les uns dans les autres, et pour terminer, l'insertion dans le document. Au moins comme ça c'est structuré, clair et surtout bien plus performant !

appendChild() retourne une *référence* (voir ci-après pour plus de détails) pointant sur l'objet qui vient d'être inséré. Cela peut servir dans le cas où vous n'avez pas déclaré de variable intermédiaire lors du processus de création de votre élément.
Par exemple, le code ci-dessous ne pose pas de problème :

Code : JavaScript - Sélectionner

```
var span = document.createElement('span');
document.body.appendChild(span);

span.innerHTML = 'Du texte en plus !';
```

En revanche, si vous retirez l'étape intermédiaire (la première ligne) pour gagner une ligne de code alors vous allez

être embêté pour modifier le contenu :

Code : JavaScript - Sélectionner

```
document.body.appendChild(document.createElement('span'));

span.innerHTML = 'Du texte en plus !'; // La variable "span"
n'existe plus... Le code plante.
```

La solution à ce problème est d'utiliser la référence retournée par **appendChild()** de la façon suivante :

Code : JavaScript - Sélectionner

```
var span = document.body.appendChild(document.createElement
('span'));

span.innerHTML = 'Du texte en plus !'; // Là, tout fonctionne !
```

Notions sur les références

En Javascript et comme dans beaucoup de langages, le contenu des variables est "passé par valeur". Cela veut donc dire que si une variable **nick1** contient le prénom "Clarisse" et qu'on affecte cette valeur à une autre variable, la valeur est copiée dans la nouvelle. On obtient alors deux variables distinctes, contenant la même valeur :

Code : JavaScript - Sélectionner

```
var nick1 = 'Clarisse';
var nick2 = nick1;
```

Si on modifie la valeur de **nick2**, la valeur de **nick1** reste inchangée : normal, les deux variables sont bien distinctes.

Les références

Outre le "passage par valeur", Javascript possède un "passage par référence". En fait, quand une variable est créée, sa valeur est mise en mémoire par l'ordinateur. Pour pouvoir retrouver cette valeur, elle est associée à une adresse que seul l'ordinateur connaît et manipule (on ne s'en occupe pas).

Quand on passe une valeur par référence, on transmet l'adresse de la valeur, ce qui va permettre d'avoir deux variables qui contiennent une même valeur !

Malheureusement, un exemple théorique d'un passage par référence n'est pas vraiment envisageable à ce stade du tutoriel, il faudra attendre d'aborder le chapitre sur la création d'objets. Cela dit, quand on manipule une page Web avec le DOM, on est confronté à des références, tout comme dans le chapitre suivant sur les événements.

Les références avec le DOM

Schématiser le concept de référence avec le DOM est assez simple : deux variables peuvent accéder au même élément. Regardez cet exemple :

Code : JavaScript - Sélectionner

```
var newLink     = document.createElement('a');
var newLinkText = document.createTextNode('Le Site du Zéro');

newLink.id  = 'sdz_link';
newLink.src = 'http://www.siteduzero.com';

newLink.appendChild(newLinkText);

document.getElementById('myP').appendChild(newLink);

// On récupère, via son ID, l'élément fraîchement inséré
var sdzLink = document.getElementById('sdz_link');

sdzLink.src = 'http://www.siteduzero.com/forum.html';

// newLink.src affiche bien la nouvelle URL :
alert(newLink.src);
```

La variable **newLink** contient en réalité une référence vers l'élément **<a>** qui a été créé. **newLink** ne contient pas l'élément, il contient une adresse qui pointe vers ce fameux **<a>**.

Une fois que l'élément HTML est inséré dans la page, on peut y accéder de nombreuses autres façons, comme avec **getElementById()**. Quand on accède à un élément via **getElementById()**, on le fait aussi au moyen d'une référence.

Ce qu'il faut retenir de tout ça, c'est que les objets du DOM sont **toujours** accessibles par référence, et c'est la raison pour laquelle ce code ne fonctionne pas :

Code : JavaScript - Sélectionner

```
var newDiv1 = document.createElement('div');
var newDiv2 = newDiv1; // On tente de copier le <div>
```

Et oui, si vous avez tout suivi, **newDiv2** contient une référence qui pointe vers le même ** ** que **newDiv1**. Mais comment dupliquer un élément alors ? Et bien il faut le cloner, et c'est ce que nous allons voir maintenant !

Cloner, remplacer, supprimer...

Cloner un élément

Pour cloner un élément, rien de plus simple : **cloneNode()**. Cette méthode accepte un paramètre booléen (**true** ou **false**) : si vous désirez cloner le nœud avec (**true**) ou sans (**false**) ses petits-enfants.

Petit exemple très simple : on crée un élément **<hr />**, et on en veut un deuxième, et donc on clone le premier :

Code : JavaScript - Sélectionner

```
// On va cloner un élément créé :
var hr1 = document.createElement('hr');
var hr2 = hr1.cloneNode(false); // Il n'a pas d'enfants...

// Ici, on clone un élément existant :
var paragraph1 = document.getElementById('myP');
var paragraph2 = paragraph1.cloneNode(true);
```

```
// Et attention, l'élément est cloné, mais pas "inséré" tant que
l'on n'a pas appelé appendChild() :
paragraph1.parentNode.appendChild(paragraph2);
```

Essayer !

 **cloneNode()** peut être utilisée tant pour cloner des nœuds textuels que des éléments HTML.

Remplacer un élément par un autre

Pour remplacer un élément par un autre, rien de plus simple, il y a **replaceChild()**. Cette méthode accepte deux paramètres : le premier est le nouvel élément, et le deuxième est l'élément à remplacer. Tout comme **cloneNode()**, cette méthode s'utilise sur tous les types de nœuds (éléments, nœuds textuels...).

Dans l'exemple suivant, le contenu textuel (pour rappel, il s'agit du premier enfant de **<a>**) du lien va être remplacé par un autre. La méthode **replaceChild()** est exécutée sur le <a>, c'est-à-dire le nœud parent du nœud à remplacer.

Code : HTML - Sélectionner

```
<body>
  <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>
</body>
```

Essayer !

Supprimer un élément

Pour insérer un élément, on utilise **appendChild()**, et pour en supprimer un, on utilise **removeChild()**. Cette méthode prend en paramètre le nœud enfant à retirer. Si on se calque sur le code HTML de l'exemple précédent, le script ressemble à ceci :

Code : JavaScript - Sélectionner

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

link.parentNode.removeChild(link);
```

 Il n'y a pas besoin de récupérer **myP** (l'élément parent) avec **getElementById()**, autant le faire directement avec **parentNode**.

Autres actions

Vérifier la présence d'éléments enfants

Rien de plus facile pour vérifier la présence d'éléments enfants : **hasChildNodes()**. Il suffit d'utiliser cette méthode sur l'élément de votre choix : si cet élément possède au moins un enfant, la méthode renverra `true` :

Code : HTML - Sélectionner

```
<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>
```

Insérer à la bonne place : insertBefore()

La méthode **insertBefore()** permet d'insérer un élément avant un autre. Elle reçoit deux paramètres : le premier est l'élément à insérer, tandis que le deuxième est l'élément avant lequel l'élément va être inséré :

Code : HTML - Sélectionner

```
<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>
```

Essayer !

 Comme pour **appendChild()**, cette méthode s'applique sur l'élément parent.

Une bonne astuce : insertAfter()

Javascript met à disposition **insertBefore()**, mais pas **insertAfter()**. C'est dommage, car parfois c'est assez utile. Qu'à cela ne tienne, créons donc une telle fonction.

Malheureusement, il ne nous est pas possible, à ce stade-ci du tutoriel, de créer une méthode, qui s'appliquerait comme ceci :

Code : JavaScript - Sélectionner

```
element.insertAfter(newElement, afterElement)
```

Non, il va falloir nous contenter d'une "simple" fonction :

Code : JavaScript - Sélectionner

```
insertAfter(newElement, afterElement)
```

Algorithme

Pour insérer après un élément, on va d'abord récupérer l'élément parent. C'est logique, puisque l'insertion de l'élément va se faire soit via **appendChild()**, soit via **insertBefore()** : si on veut ajouter notre élément après le dernier enfant, c'est simple, il suffit d'appliquer **appendChild()**. Par contre, si l'élément après lequel on veut insérer notre élément n'est pas le dernier, on va utiliser **insertBefore()** en ciblant l'enfant suivant, avec **nextSibling** :

Code : JavaScript - Sélectionner

```
function insertAfter(newElement, afterElement) {
    var parent = afterElement.parentNode;

    if (parent.lastChild === afterElement) { // Si le dernier
élément est le même que l'élément après lequel on veut insérer, il
suffit de faire appendChild()
        parent.appendChild(newElement);
    } else { // Dans le cas contraire, on fait un insertBefore() sur
l'élément suivant
        parent.insertBefore(newElement, afterElement.nextSibling);
    }
}
```

Minis-TD : recréer une structure DOM

Afin de s'entraîner à jouer avec le DOM, voici quatre petits exercices. Pour chaque exercice, une structure DOM sous forme de code HTML vous est donnée, et il vous est demandé de re-créer cette structure en utilisant le DOM.

 Notez que la correction donnée est une solution possible, et qu'il en existe d'autres. Chaque codeur possède un style de code, une façon de réfléchir, d'organiser et de présenter son code, surtout ici, où les possibilités sont nombreuses.

Premier exercice

Énoncé

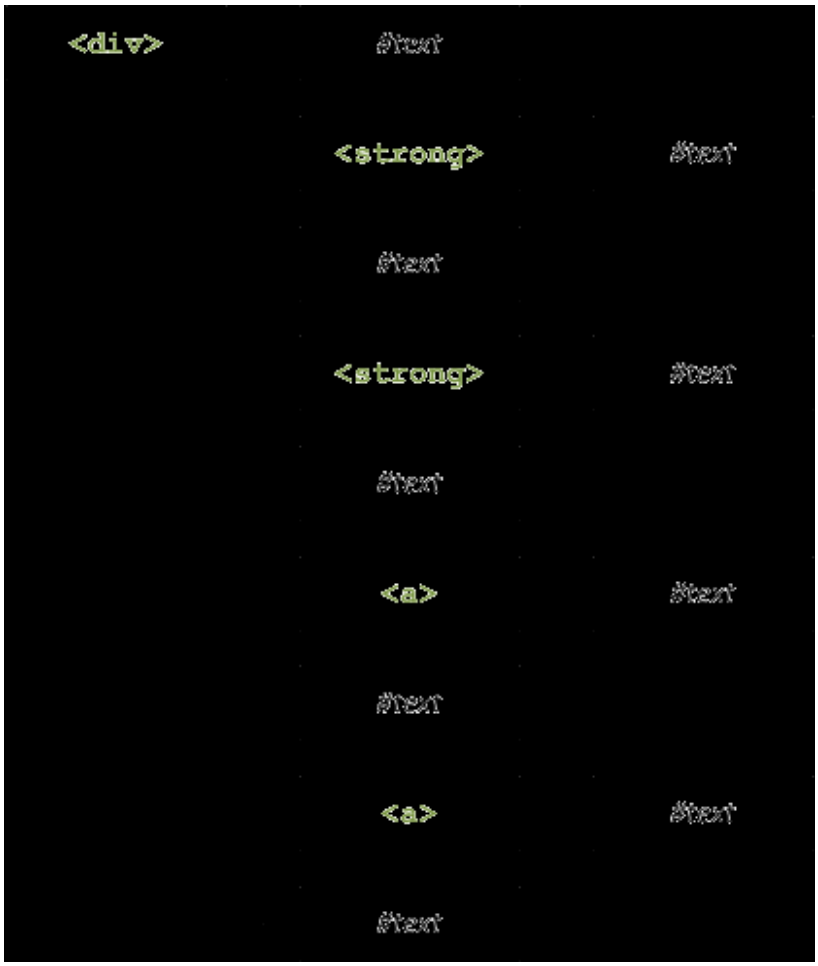
Pour ce premier exercice, nous vous proposons de recréer "du texte" mélangé à divers éléments tels des **<a>** et des ****. C'est assez simple, mais pensez bien à ne pas vous emmêler les pinceaux avec tous les nœuds textuels !

Code : HTML - Sélectionner

```
<div id="divTD1">
  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>
```

Corrigé

Secret (cliquez pour afficher)



Code : JavaScript - Sélectionner

```
// On crée l'élément conteneur
var mainDiv = document.createElement('div');
    mainDiv.id = 'divTD1';

// On crée tous les noeuds textuels, pour plus de facilité
var textNodes = [
    document.createTextNode('Le '),
    document.createTextNode('World Wide Web Consortium'),
    document.createTextNode(', abrégé par le sigle '),
    document.createTextNode('W3C'),
    document.createTextNode(', est un '),
    document.createTextNode('organisme de standardisation'),
    document.createTextNode(' à but non-lucratif chargé de
promouvoir la compatibilité des technologies du '),
    document.createTextNode('World Wide Web'),
    document.createTextNode('.')
];

// On crée les deux <strong> et les deux <a>
var w3cStrong1 = document.createElement('strong');
var w3cStrong2 = document.createElement('strong');

w3cStrong1.appendChild(textNodes[1]);
w3cStrong2.appendChild(textNodes[3]);
```

```
var orgLink = document.createElement('a');
var wwwLink = document.createElement('a');

orgLink.href  =
'http://fr.wikipedia.org/wiki/Organisme_de_normalisation';
orgLink.title = 'Organisme de normalisation';
orgLink.appendChild(textNodes[5]);

wwwLink.href  = 'http://fr.wikipedia.org/wiki/World_Wide_Web';
wwwLink.title = 'World Wide Web';
wwwLink.appendChild(textNodes[7]);

// On insère le tout dans mainDiv
mainDiv.appendChild(textNodes[0]);
mainDiv.appendChild(w3cStrong1);
mainDiv.appendChild(textNodes[2]);
mainDiv.appendChild(w3cStrong2);
mainDiv.appendChild(textNodes[4]);
mainDiv.appendChild(orgLink);
mainDiv.appendChild(textNodes[6]);
mainDiv.appendChild(wwwLink);
mainDiv.appendChild(textNodes[8]);

// On insère mainDiv dans le <body>
document.body.appendChild(mainDiv);
```

Essayer !

Par mesure de facilité, tous les nœuds textuels sont contenus dans le tableau **textNodes**. Ça évite de faire 250 variables différentes. Une fois les nœuds textuels créés, on crée les éléments **<a>** et ****. Une fois que tout cela est fait, on insère le tout, un élément après l'autre, dans le div conteneur.

Deuxième exercice

Énoncé

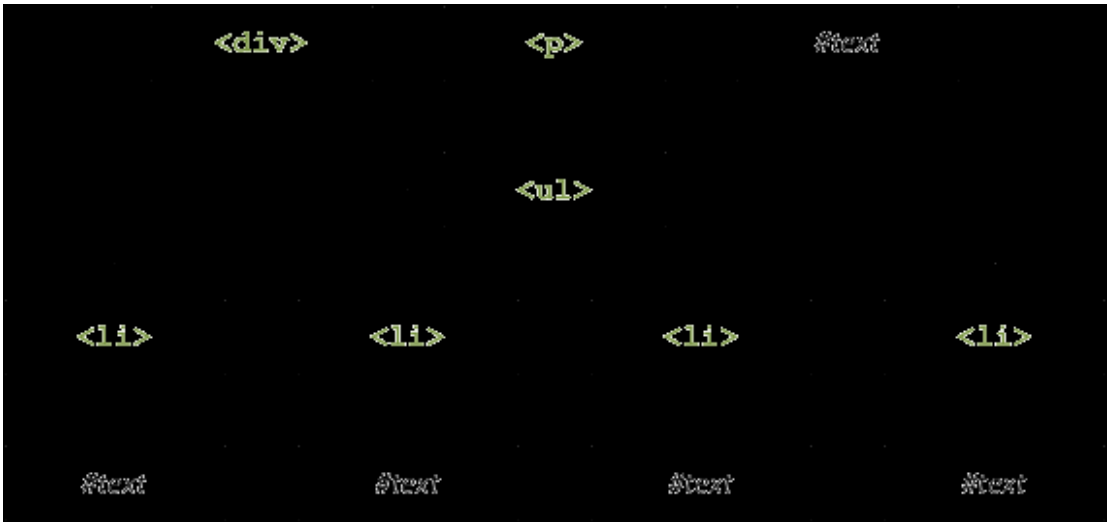
Code : HTML - Sélectionner

```
<div id="divTD2">
  <p>Langages basés sur ECMAScript :</p>

  <ul>
    <li>JavaScript</li>
    <li>JScript</li>
    <li>ActionScript</li>
    <li>EX4</li>
  </ul>
</div>
```

On ne va tout de même pas créer 4 éléments **<li>** "à la main"... Utilisez une boucle **for** ! Et souvenez-vous, les éléments textuels dans un tableau.

Corrigé

Secret (cliquez pour afficher)



Code : JavaScript - Sélectionner

```
// On crée l'élément conteneur
var mainDiv = document.createElement('div');
    mainDiv.id = 'divTD2';

// On crée tous les noeuds textuels, pour plus de facilité
var languages = [
    document.createTextNode('JavaScript'),
    document.createTextNode('JScript'),
    document.createTextNode('ActionScript'),
    document.createTextNode('EX4')
];

// On crée le paragraphe
var paragraph     = document.createElement('p');
var paragraphText = document.createTextNode('Langages basés sur
ECMAScript :');
paragraph.appendChild(paragraphText);


// On crée la liste, et on boucle pour ajouter chaque item
var uList = document.createElement('ul'),
    uItem;

for (var i = 0, c=languages.length; i < c; i++) {
    uItem = document.createElement('li');

    uItem.appendChild(languages[i]);
    uList.appendChild(uItem);
}

// On insère le tout dans mainDiv
mainDiv.appendChild(paragraph);
mainDiv.appendChild(uList);

// On insère mainDiv dans le <body>
document.body.appendChild(mainDiv);
```

Essayer !

Les nœuds textuels de la liste à puces sont crées par le biais du tableau **languages**, et pour créer chaque élément <li>, il suffit de boucler sur le nombre d'items du tableau.

Troisième exercice

Énoncé

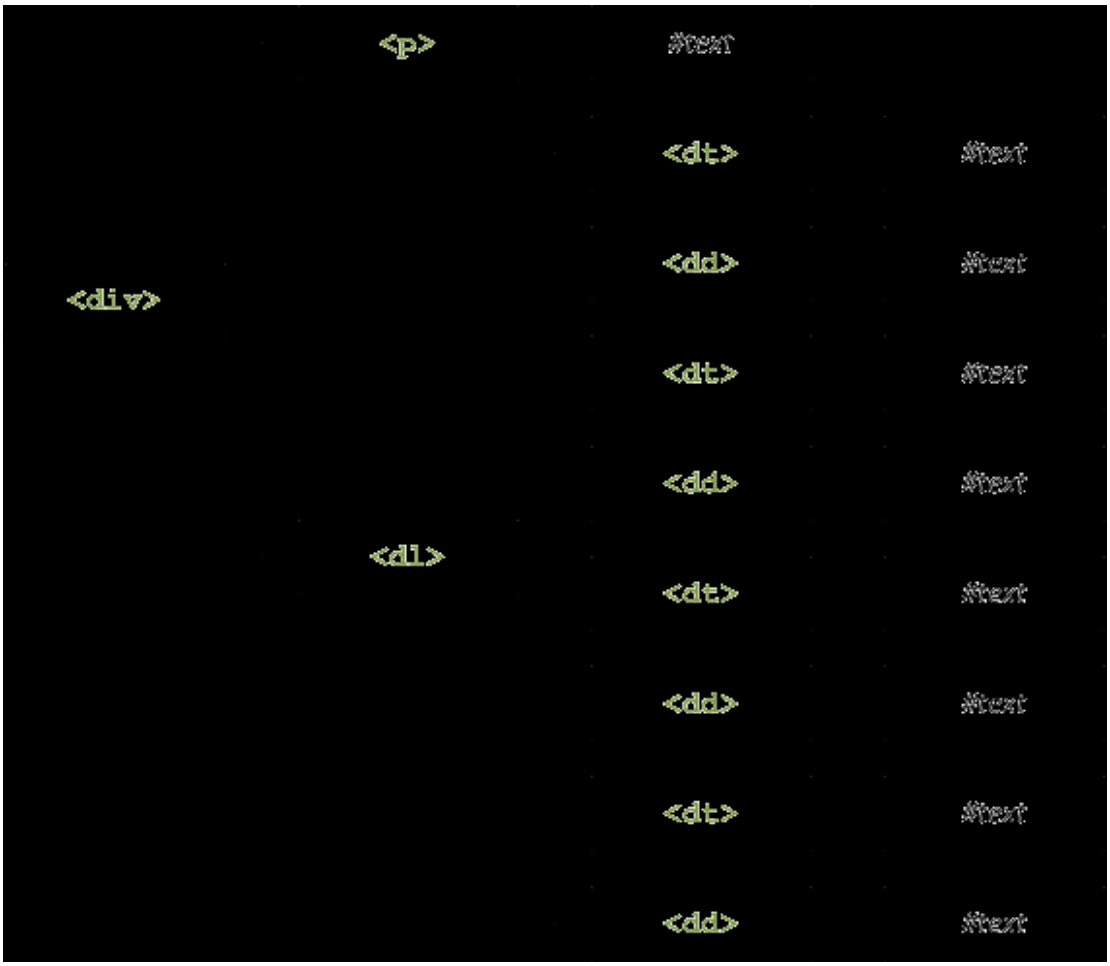
Voici une version légèrement plus complexe de l'exercice précédent. Le schéma de fonctionnement est le même, mais ici, le tableau **languages** contiendra des objets littéraux, et chacun de ces objets contiendra deux propriétés : le nœud du **<dt>** et le noeud du **<dd>**.

Code : HTML - Sélectionner

```
<div id="divTD3">
  <p>Langages basés sur ECMAScript :</p>

  <dl>
    <dt>JavaScript</dt>
    <dd>JavaScript est un langage de programmation de scripts
principalement utilisé dans les pages web interactives mais aussi
coté serveur.</dd>

    <dt>JScript</dt>
    <dd>JScript est le nom générique de plusieurs implémentations
d'ECMAScript 3 créées par Microsoft.</dd>

    <dt>ActionScript</dt>
    <dd>ActionScript est le langage de programmation utilisé au sein
d'applications clientes (Adobe Flash, Adobe Flex) et serveur (Flash
media server, JRun, Macromedia Generator).</dd>

    <dt>EX4</dt>
    <dd>ECMAScript for XML (E4X) est une extension XML au langage
ECMAScript.</dd>
  </dl>
</div>
```

Corrigé

Secret (cliquez pour afficher)



Code : JavaScript - Sélectionner

```
// On crée l'élément conteneur
var mainDiv = document.createElement('div');
    mainDiv.id = 'divTD3';

// On place le texte dans des objets, eux-même placés dans un
tableau
// Par facilité, la création des nœuds textuels se fera dans la
boucle
var languages = [
    { t: 'JavaScript',
      d: 'JavaScript est un langage de programmation de scripts
principalement utilisé dans les pages web interactives mais aussi
coté serveur.' },
    { t: 'JScript',
      d: 'JScript est le nom générique de plusieurs
implémentations d\'ECMAScript 3 créées par Microsoft.' },
    { t: 'ActionScript',
      d: 'ActionScript est le langage de programmation utilisé au
sein d\'applications clientes (Adobe Flash, Adobe Flex) et
serveur (Flash media server, JRun, Macromedia Generator).' },
    { t: 'EX4',
      d: 'ECMAScript for XML (E4X) est une extension XML au
langage ECMAScript.' }
];
```

```
// On crée le paragraphe
var paragraph     = document.createElement('p');
var paragraphText = document.createTextNode('Langages basés sur
ECMAScript :');
paragraph.appendChild(paragraphText);


// On crée la liste, et on boucle pour ajouter chaque item
var defList = document.createElement('dl'),
    defTerm, defTermText,
    defDefn, defDefnText;

for (var i = 0, c=languages.length; i < c; i++) {
    defTerm = document.createElement('dt');
    defDefn = document.createElement('dd');

    defTermText = document.createTextNode(languages[i].t);
    defDefnText = document.createTextNode(languages[i].d);

    defTerm.appendChild(defTermText);
    defDefn.appendChild(defDefnText);

    defList.appendChild(defTerm);
    defList.appendChild(defDefn);
}

// On insère le tout dans mainDiv
mainDiv.appendChild(paragraph);
mainDiv.appendChild(defList);

// On insère mainDiv dans le <body>
document.body.appendChild(mainDiv);
```

Le tableau contient des objets comme ceci :

Code : JavaScript - Sélectionner

```
{
    t: 'Terme',
    d: 'Définition'}
```

Essayer !

Créer une liste de définitions (**<dl>**) n'est pas plus compliqué qu'une liste à puces normale, la seule chose qui diffère est que **<dt>** et **<dd>** sont ajoutés conjointement au sein de la boucle.

Quatrième exercice

Énoncé

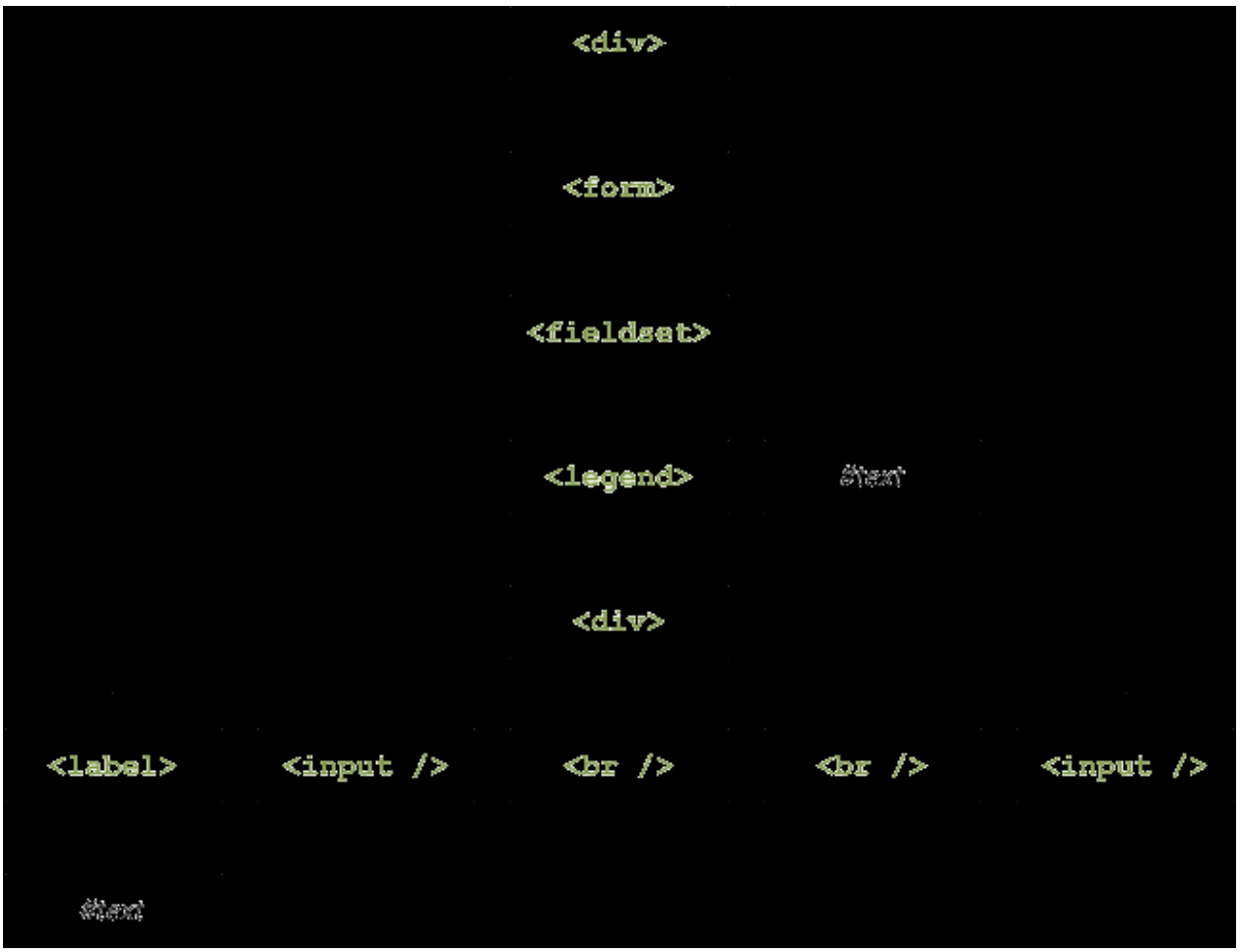
Un rien plus corsé... quoique. Ici, la difficulté réside dans le nombre important d'éléments à imbriquer les uns dans les autres. Si vous procédez méthodiquement, vous avez peu de chance de vous planter.

Code : HTML - Sélectionner

```
<div id="divTD4">
  <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>
```

Corrigé

Secret (cliquez pour afficher)



Code : JavaScript - Sélectionner

```
// On crée l'élément conteneur
var mainDiv = document.createElement('div');
    mainDiv.id = 'divTD4';

// Création de la structure du formulaire
var form       = document.createElement('form');
var fieldset   = document.createElement('fieldset');
var legend     = document.createElement('legend'),
    legendText = document.createTextNode('Uploader une image');
var center     = document.createElement('div');

form.action  = 'upload.php';
form.enctype = 'multipart/form-data';
form.method  = 'post';

center.setAttribute('style', 'text-align: center');

legend.appendChild(legendText);

fieldset.appendChild(legend);
fieldset.appendChild(center);

form.appendChild(fieldset);

// Création des champs
var label =     document.createElement('label'),
    labelText = document.createTextNode('Image à uploader :');
var input =     document.createElement('input');
var br =        document.createElement('br');
var submit =    document.createElement('input');

input.type = 'file';
input.id   = 'inputUpload';
input.name = input.id;

submit.type  = 'submit';
submit.value = 'Envoyer';

label.appendChild(labelText);

center.appendChild(label);
center.appendChild(input);
center.appendChild(br);
center.appendChild(br.cloneNode(false)); // On clone, pour mettre
un 2e <br />
center.appendChild(submit);

// On insère le formulaire dans mainDiv
mainDiv.appendChild(form);

// On insère mainDiv dans le <body>
document.body.appendChild(mainDiv);
```

Essayer !

Comme il y a beaucoup d'éléments à créer, pourquoi ne pas diviser le script en deux : la structure du formulaire, et les champs. C'est plus propre, et on s'y retrouve mieux.

Conclusion des TD

Il est très probable que vous n'ayez pas organisé votre code comme dans les corrections, ou que vous n'avez pas utilisé les mêmes idées, comme utiliser un tableau, ou même un tableau d'objets. Votre code est certainement bon, mais retenez une chose : essayez d'avoir un code clair et propre, tout en étant facile à comprendre, ça vous simplifiera la tâche !

Q.C.M.

Quelle est la différence d'utilisation entre **nodeValue** et **data** si on les applique à l'élément suivant ?

Code : HTML - Sélectionner

```
<strong>Du texte !</strong>
```

- Aucune différence, dans les deux cas il faut passer par le nœud textuel
- Avec nodeValue, on peut récupérer le contenu de l'élément sans passer par le nœud textuel
- Avec data, il faut d'abord accéder au nœud textuel de notre élément
- Aucune différence, dans les deux cas on peut se passer du nœud textuel

nextSibling permet :

- d'accéder au nœud suivant
- d'accéder au nœud précédent
- d'accéder au nœud parent
- de ne rien faire du tout

Quand on crée un élément avec **createElement()**, il est obligatoire de définir ses attributs avant d'utiliser **appendChild()**.

- Vrai
- Faux

Comment copier un élément HTML ?

- En copiant simplement la variable
- En utilisant cloneNode()
- En passant l'élément en paramètre d'une fonction

Correction !

Statistiques de réponses au QCM

Après un gros chapitre comme celui-ci, quoi de mieux que d'enchaîner sur un autre, le petit frère du maniement du DOM : les événements. Accrochez-vous, les choses sérieuses commencent !