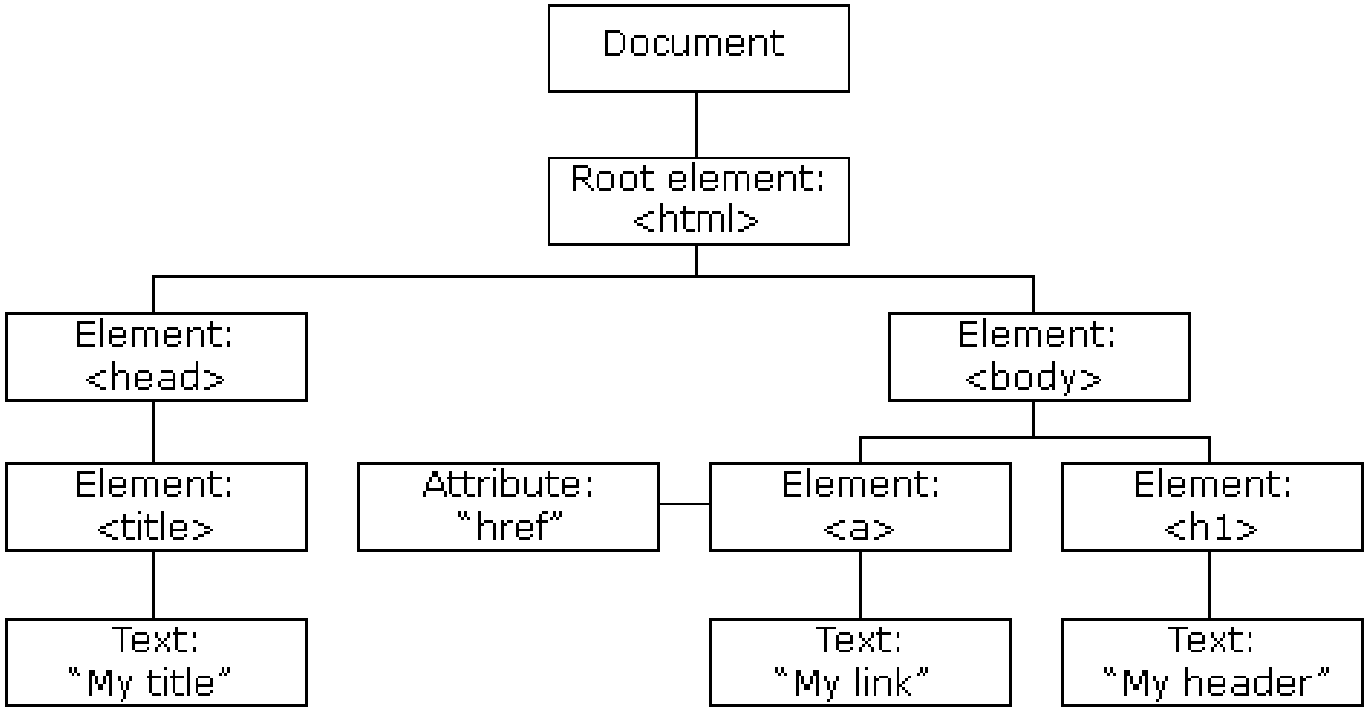
DOM : Document Object Model

Introduction

- Lorsqu'une page web est chargée, le navigateur crée le modèle objet document de la page (**D**ocument **O**bject **M**odel).
- Le DOM HTML est alors construit en tant qu'une arborescence d'objets:



Introduction

Avec l'interface de programmation DOM, Javascript permet de :

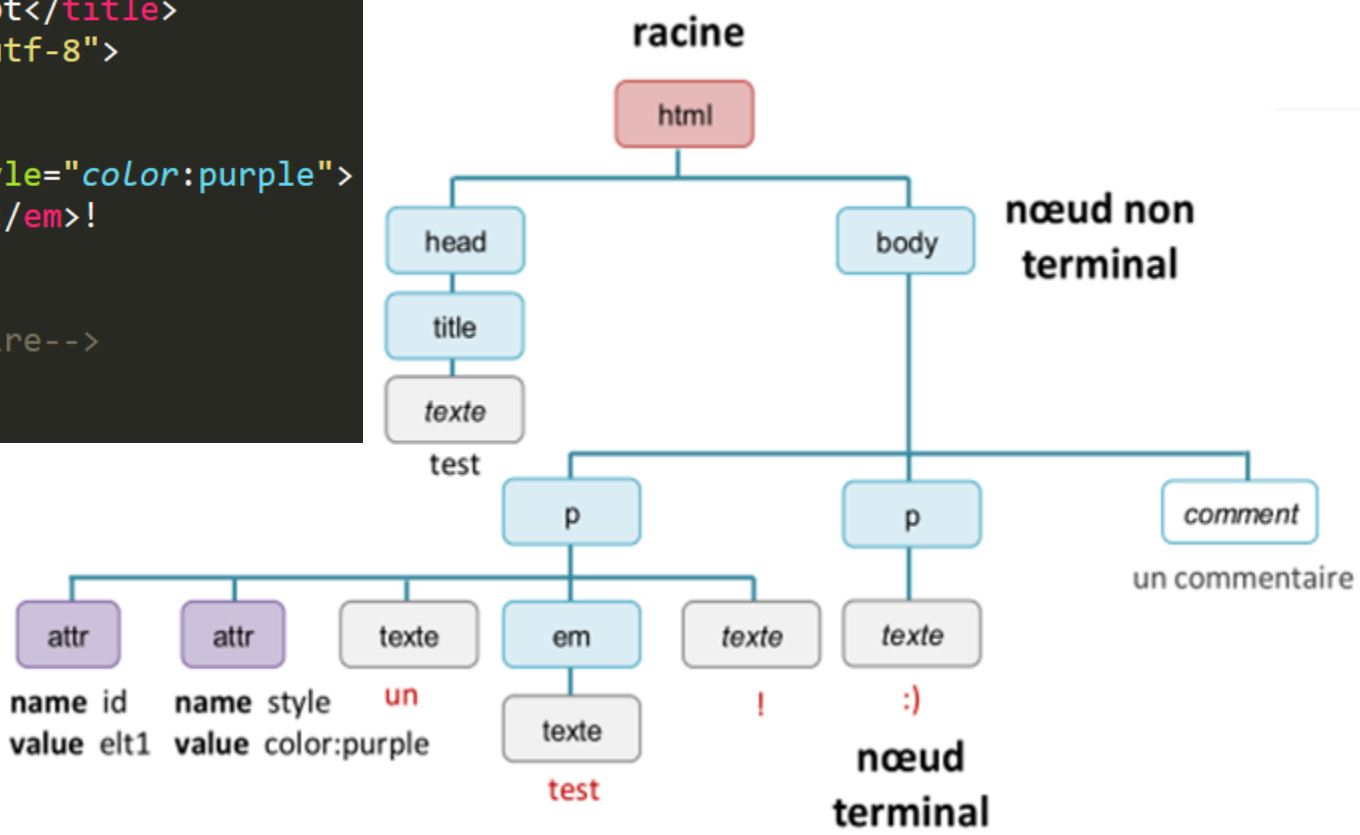
- Changer les éléments HTML d'une page
- Changer les attributs des éléments HTML
- Changer les styles CSS dans la page
- Ajouter/Supprimer des éléments HTML et des attributs
- Réagir aux différents évènements dans la page

HTML DOM:

- DOM est un standard du W3C (World Wide Web Consortium)
- Définit un standard pour permettre aux scripts d'accéder aux documents dynamiquement pour modifier le contenu, la structure et les styles
- DOM définit:
  - Les élément HTML en tant qu'**objets**
  - Les **propriétés** de tous les éléments HTML
  - Les **méthodes** pour manipuler les éléments HTML
  - Les évènements auxquels peuvent réagir les éléments HTML

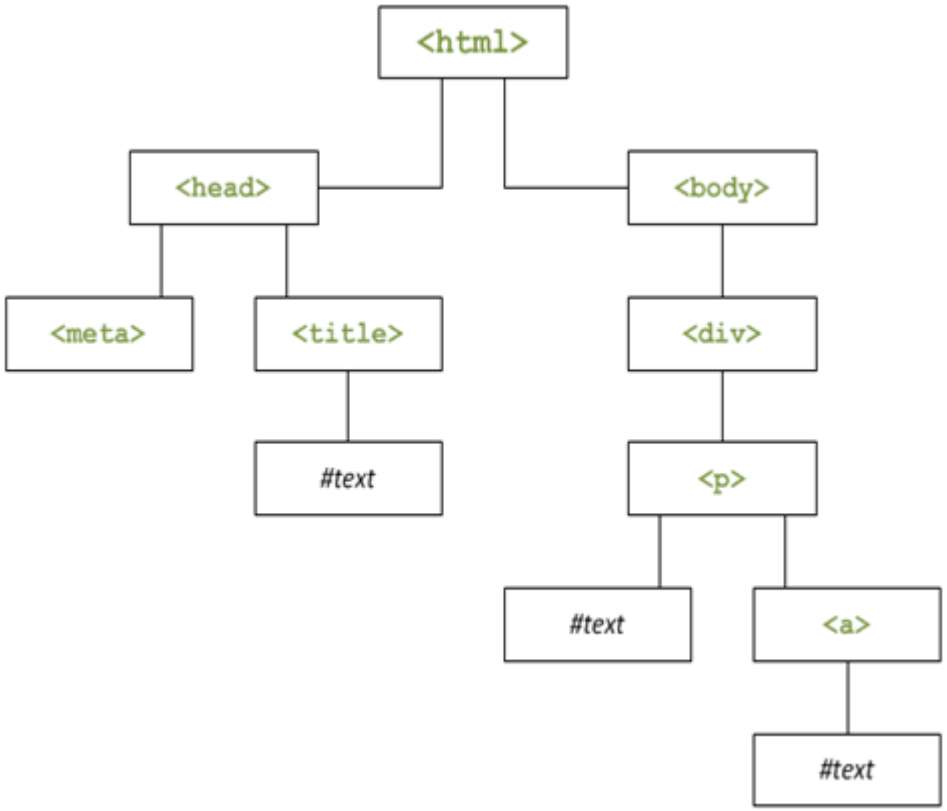
La structure DOM (1)

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Javascript</title>
        <meta charset="utf-8">
    </head>
    <body>
        <p id="elt1" style="color:purple">
            un <em>test</em>!
        </p>
        <p>:)</p>
        <!--un commentaire-->
    </body>
</html>
```



La structure DOM (2)

- le DOM pose comme concept que la page Web est vue comme un arbre, comme une hiérarchie d'éléments. On peut donc schématiser une page Web simple comme ceci :




```
1  <!doctype html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Le titre de la page</title>
6    </head>
7
8    <body>
9      <div>
10       <p>Un peu de texte <a>et un lien</a></p>
11     </div>
12   </body>
13 </html>
```

La structure DOM (3)

- L'élément <html> contient deux éléments, appelés **enfants** : <head> et <body>. Pour ces deux enfants, <html> est l'élément **parent**. Chaque élément est appelé **nœud** (*node* en anglais). L'élément <head> contient lui aussi deux enfants : <meta> et <title>. <meta> ne contient pas d'enfant tandis que <title> en contient un, qui s'appelle #text. Comme son nom l'indique, #text est un élément qui contient du texte.
- NB : le texte présent dans une page Web est vu par le DOM comme un nœud de type #text.

L’arborescence du DOM (1)

- Le schéma adopté des documents est une arborescence hiérarchisée.
- Les différentes composantes d'une telle arborescence sont désignées comme étant des nœuds. L'objet central du modèle DOM est pour cette raison l'objet **node (node = nœud).**
- Il existe différents types de nœuds : les nœuds-élément, les nœuds-enfant, nœuds-associé et les nœuds-texte.

L’arborescence du DOM (2)

Exemple :

<h1 align="center">Bonjour tout le monde</h1>

- L'élément h1 est le nœud de départ
- Ce nœud a d'après les règles du modèle DOM, deux nœuds-enfant et un nœud associé :
  - **Les nœuds enfant** sont d'une part *le nœud texte* avec le mot "Bonjour" suivi d'un espace, d'autre part *le nœud élément* de l'élément i.
  - L'attribut align dans la balise ouvrante <h1> n'est pas par contre un nœud-enfant mais un **nœud associé**.
- Le **nœud-attribut :** la valeur affectée(center).
- Le nœud-élément de l'élément a un nœud-enfant de type **texte** (le mot "tout le monde").

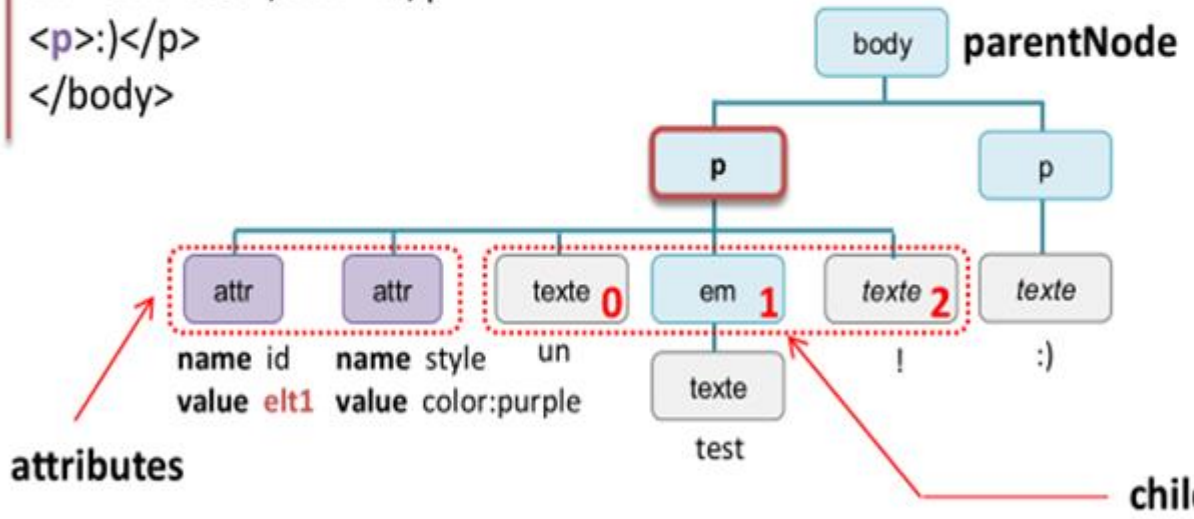
accès aux nœuds (1)

Accès par

ID	document.getElementById('*id_element*')	-> noeud
Nom d'élément	document.getElementsByTagName('*nom_element*')	-> collection de nœuds
Valeur de l'attribut name	document.getElementsByName('*val_attribut*')	-> collection de nœuds

```
<body>
<p id="elt1" style="color:purple">
un <em>test</em> !</p>
<p>:)</p>
</body>
```

un *test* !
:)



Collection = tableau de nœuds

prédéfinies : forms, images, links
fils d'un nœud : childNodes
attributs d'un nœud : attributes
nom : getElementsByName
attr. name : getElementsByTagName

accès à un nœud : [i] ou item(i)
avec i >= 0

```
var el = document.getElementById('elt1');
alert(el.tagName);                  // P
alert(el.parentNode.nodeName);      // BODY
alert(el.childNodes[1].nodeName)    // EM
```

```
var tab_el = document.getElementsByTagName('p');
alert(tab_el[0].childNodes[2].nodeName);  // #text
alert(tab_el[0].childNodes[2].nodeValue);    // !
```

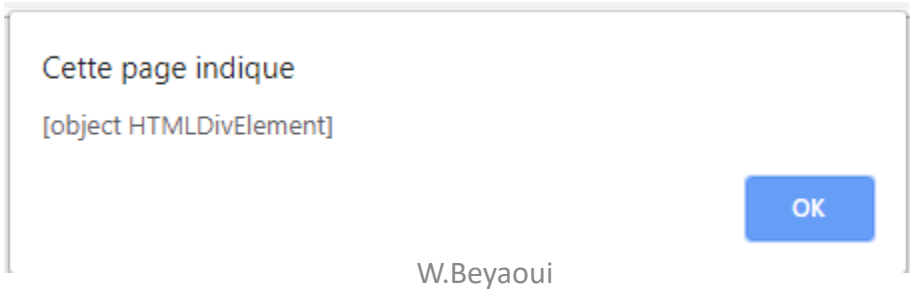
accès aux nœuds (2)

- **getElementById() :**

Cette méthode permet de récupérer, s'il existe, le premier élément HTML dont *l'attribut ID* a la valeur spécifiée. Sinon un objet à la valeur Null sera retourné.

Exemple :

```
<div id="div1" >
    <p>un peu de texte<a>et un lien </a></p>
</div>
<script>
    var divs = document.getElementById('div1');
    alert(div);
</script>
```



accès aux nœuds (3)

- **getElementsByTagName() :**

Cette méthode permet de récupérer, sous la forme d'un tableau, *tous les éléments* dont *la balise* a le nom spécifié.

Exemple: Dans une page, on veut récupérer tous les , et le afficher:

```
var divs = document.getElementsByTagName('div');
for (var i = 0; i < divs.length; i++)
{
    alert('Elément N°' + (i+1) + ':' + divs[i]);
}
```

accès aux nœuds (4)

- **getElementsByName() :** Renvoie une liste (tableau) des éléments qui possèdent l'attribut name passé en paramètre.

Exemple:

```
<div name="up">division 1</div>
<div name="up">division 2</div>
<div name="other">division 3</div>
<script>
    up_divs = document.getElementsByName("up");
    //retourne une liste des 2 éléments div avec name="up"
</script>
```

accès aux nœuds (4)

- **getElementsByClassName() :** Renvoie une liste des éléments ayant le nom de la classe donnée.

Exemple:

```
<div class="entete class1">l'entête de la page</div>
<div class="corps">ceci est le corps du texte</div>
<div class=" entete">une autre entête</div>
<script>
    divs_entete = document.getElementsByClassName("entete");
    //retourne une liste des 2 éléments div avec la liste des class
    contenant la classe entete
```

accès aux nœuds (5)

➢ **querySelector() et querySelectorAll():**

- Permettent de simplifier la sélection d'éléments dans l'arbre DOM.
- Prennent en paramètre un sélecteur CSS.
- La première, querySelector(), renvoie le premier élément trouvé correspondant au sélecteur CSS, tandis que querySelectorAll() va renvoyer *tous* les éléments (sous forme de tableau) correspondant au sélecteur CSS fourni.

Exemple:

```
<div>première division</div>
<div>deuxième division</div>
<div>troisième division</div>
<script>
    uneDiv = document.querySelector("div");
    // retourne le premier élément avec le selecteur div
    divs = document.querySelectorAll("div");
    // retourne un tableau (de taille 3) des éléments avec le selecteur div
</script>
```

Exemple

```html
<div id="menu">
    <div class="item">
        <span>Elément 1</span>
        <span>Elément 2</span>
    </div>
    <div class="publicite">
        <span>Elément 3</span>
        <span>Elément 4</span>
    </div>
</div>
<div id="contenu">
    <span>Introduction au contenu de la page...</span>
</div>
```

```html
<script>
    var uneDiv = document.querySelector("#menu .item span");
    console.log(uneDiv.innerHTML);  //affiche: "Elément1"

    var divs = document.querySelectorAll("#menu .item span");
    console.log(divs.length);   //affiche: "2"

    console.log(divs[0].innerHTML + '-' + divs[1].innerHTML);
    //affiche: "Elément1-Elément2"
</script>
```

Les propriétés nodeType, nodeName et nodeValue

nodeType : retourne les type d'un nœud sous forme d'un nombre entier. Voici un tableau qui liste les types courants, ainsi que leurs numéros.

Numéro	Type
1	nœud élément
2	Nœud attribut
3	Nœud texte
8	Nœud commentaire
9	Nœud document

nodeName : Selon le type de nœud, sa valeur est,

– pour un nœud élément : le nom de sa balise, en toutes majuscules;
– pour un nœud attribut : le nom de l'attribut;
– pour un nœud texte : #text;
– pour un nœud commentaire: #comment;
– pour le nœud document : toujours #document.

nodeValue (ou aussi data) : retourne la valeur d'un nœud textuel (#texte, attribut ou commentaire)

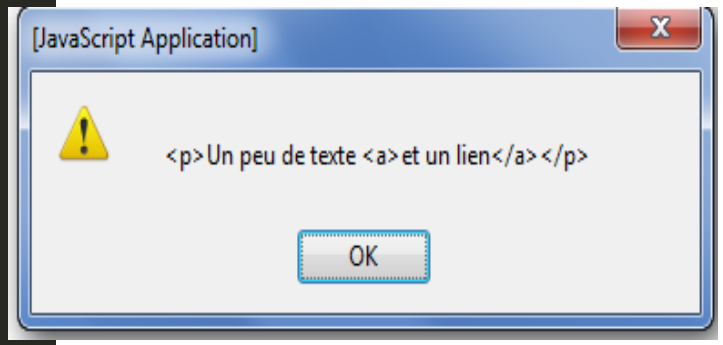
Editer les éléments HTML (1)

Récupérer/modifier du HTML avec *innerHTML:*

Permet de récupérer/modifier le code HTML enfant d'un élément sous forme de texte.

- Récupérer le contenu HTML sous forme de texte :

```
1  <body>
2    <div id="myDiv">
3      <p>Un peu de texte <a>et un lien</a></p>
4    </div>
5
6    <script>
7      var div = document.getElementById('myDiv');
8
9      alert(div.innerHTML);
10   </script>
11 </body>
```



- Modifier le contenu HTML: définir un nouveau contenu :

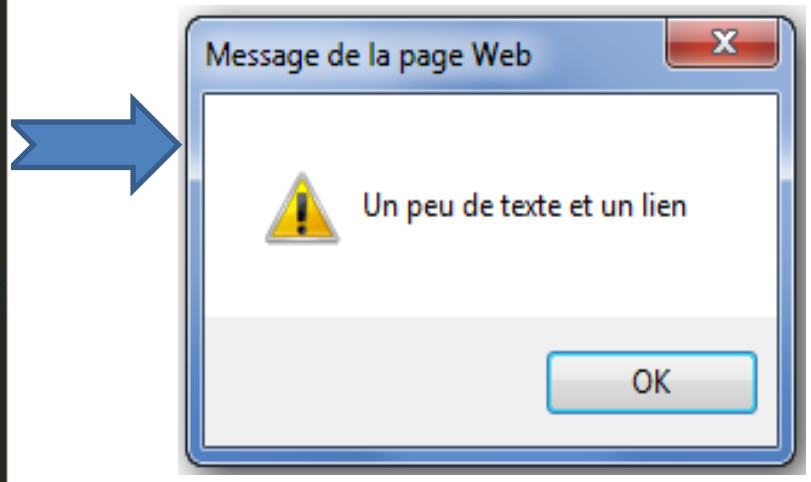
```
var div = document.querySelector('#myDiv');
div.innerHTML = "<h1>un titre au lieu du paragraphe</h1>";
```

Editer les éléments HTML (2)

Récupérer/modifier du texte avec <u>textContent:</u>

- Récupère le texte (du nœud et ses descendants), pas les balises.

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

  <script>
    var div = document.getElementById('myDiv');

    alert(div.textContent);
  </script>
</body>
```



- Modifier le texte du nœud (écrase l'ancien contenu HTML):

```
var div = document.querySelector('#myDiv');
div.textContent =   "un contenu texte simple";
```

Editer les attributs des éléments HTML (1)

- **getAttribute(nom_attribut)** : permet de récupérer un attribut.
- **setAttribute(nom_attribut, valeur)** : permet de modifier un attribut.

```
<body>
    <a id="lien1" href="www.monsite.com">un lien modifié dynamiquement</a>

    <script>
        var unLien = document.querySelector('#lien1');

        alert(unLien.getAttribute('href'));

        //modifier la cible du lien
        unLien.setAttribute('href','www.isetr.rnu.tn');

    </script>
</body>
```

- **hasAttribute(nom_attribut)** : retourne true si l'élément possède l'attribut dont le nome est passé en paramètre.
- **hasAttributes()**: retourne true si l'élément possède des attributs HTML
- **removeAttribute(nom_attribut)** : supprime l'attribut dont le nome est passé en paramètre.

Editer les attributs des éléments HTML (2)

- On peut directement accéder aux attributs **id**, **class** (devient className), **title**, **style, href** (pour la balise a) et **src** (pour la balise img) d'un élément donné à l'aide des propriétés du DOM.

```
<body>
<div id="div1" class="class1 class2">
</div>
<script>
    var div = document.querySelector('#div1')
    console.log(div.id);          //afficher de la valeur de l'attribut id
    div.id = 'div2';              //modifier la valeur de l'attribut id
    console.log(div.className); //afficher de la valeur de l'attribut class
    div.className = 'class3';     //modifier la valeur de l'attribut class
    div.title = 'titre1';         // affecter une valeur à l'attribut title
    div.style = 'color:red';      // affecter une valeur à l'attribut style
</script>
</body>
</html>
```

L'objet style (1)

- on peut interagir avec les styles d'un élément en utilisant un nouvel objet (**style**) défini sur chaque élément du DOM.
- L'objet style possède des propriétés qui correspondent aux déclarations d'attributs de style CSS
- La syntaxe des propriétés de l'objet style est très peu différentes de celles des attributs CSS. Il suffit de remplacer le trait d'union qui sépare deux mot par la majuscule de la première lettre du deuxième mot, Par exemple:
  - **element.style.backgroundColor** au lieu de **background-color**
  - **element.style.borderBottomLeftRadius** au lieu de **border-bottom-left-radius**

L'objet style (2)

Exemple :

en CSS:

```
<style type="text/css">
    #div1{
        font-size:12px; border-bottom: 1px solid #000;
    }
</style>
```

l'équivalent en DOM est :

```
<div id="div1" >test
</div>
<script>
    var div = document.querySelector('#div1');
    div.fontSize = '12px';
    div.style.borderBottom = "1px solid #000";
</script>
```

L'objet classList

- classList est une propriété (en lecture seule) d'un élément DOM HTML
- Retourne la liste des classes d'un éléments
- L'objet classList offre des méthodes pour ajouter, supprimer ou basculer entre ajout/suppression de classes à un élément
- **Propriétés**: **length** retourne le nombre de classes dans la liste
- **Méthodes**:

Méthode	Description
add(*class1, class2, ...*)	Ajoute un ou plusieurs classes à un élément, si la classe existe, elle ne sera pas ajoutée.
contains(*class*)	Retourne "true" si la classe passée en paramètre existe dans la liste, "false" sinon.
item(*index*)	Retourne le nom de la classe à l'index donné. Si l'index n'existe pas, elle retourne "null"
remove(*class1, class2, ...*)	Supprime un ou plusieurs classes
toggle(*class)*	Basculer entre ajout/suppression d'une classe

Les évènements avec DOM (1)

- Nous avons vu précédemment que les éléments d'une page web peuvent réagir à des évènements utilisateur (click par exemple) ou navigateur (chargement ou fermeture de la page par exemple).
- Un gestionnaire d'évènement est le fait d'associer un code Javascript (fonction par exemple) qui sera exécuté lorsqu'un évènement se produira sur un élément,
- Le gestionnaire d'évènement peut être représenté par un attribut HTML de la forme "**onevenement"** (on suivi du nom de l'évènement)
- **Exemple:**

```
<button onclick="this.innerHTML = Date()">La date est?</button>
```

Les évènements avec DOM (2)

addEventListener():

- Cette méthode attache dynamiquement un gestionnaire d'évènement à un élément DOM.
- Elle permet d'attacher plusieurs gestionnaires d'évènements au même élément.
- Syntaxe : *element*.addEventListener(*event*, *function)*
- **Exemple:**

```
<button>bouton1</button>
<script>
    function afficher1(){
        alert('un premier message');
    }
    function afficher2(){
        alert('un deuxième message');
    }
    var bouton = document.querySelector('button');
    var test = false;
    bouton.addEventListener('click', afficher1);
    if(test){
        bouton.addEventListener('click', afficher2);
    }
```

Les évènements avec DOM (3)

removeEventListener():

- Cette méthode détache dynamiquement un gestionnaire d'évènement d'un élément DOM.
- Syntaxe : *element*.removeEventListener(*event*, *function)*
- **Exemple:**

```
<button>bouton1</button>
<script>
    function afficher1(){
        alert('un premier message');
    }
    function afficher2(){
        alert('un deuxième message');
    }
    var bouton = document.querySelector('button');
    var test = false;
    bouton.addEventListener('click', afficher1);
    if(!test){
        bouton.removeEventListener('click', afficher1);
        bouton.addEventListener('click', afficher2);
    }
</script>
```

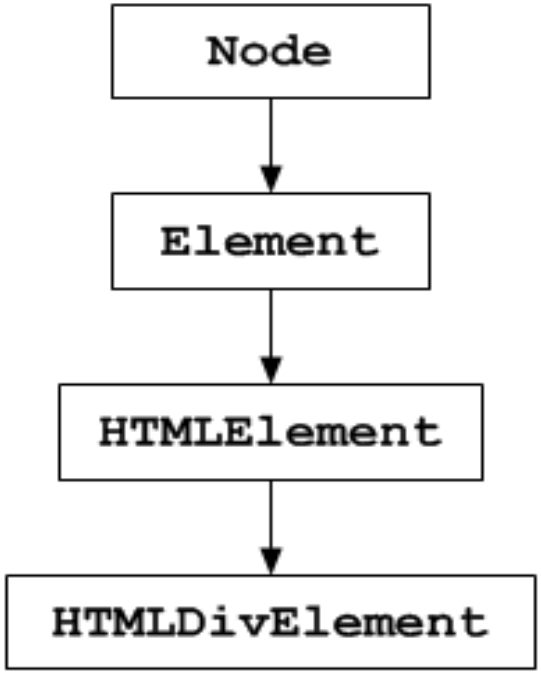
L'objet "Event"

- Lorsqu'un évènement se produit, un objet de type "Event" est créé puis passé à la fonction exécutée par le gestionnaire d'évènement.
- En fonction de l'évènement, l'objet "Event" peut être de type Event, MouseEvent, KeyBoardEvent, AnimationEvent, DragEvent, TouchEvent, UiEvent,…
- Le lien w3schools suivant présente la liste des objets "Event" ainsi que les propriétés et les méthodes qui les appartiennent: https://www.w3schools.com/jsref/dom_obj_event.asp
- **Exemple:**

```
<input type="text">
<div id="position"></div>
<script>
    //afficher le caractère tapé dans l'élémnt input
    function afficher(e){ alert(e.key); }

    //afficher les coordonnées du pointeur souris dans la page
    function position(e){
        div.textContent = e.clientX + ',' + e.clientY;  }

    var div = document.querySelector('#position');
    var input = document.querySelector('input');

    document.addEventListener('mousemove', position);
    input.addEventListener('keypress', afficher);
</script>
```

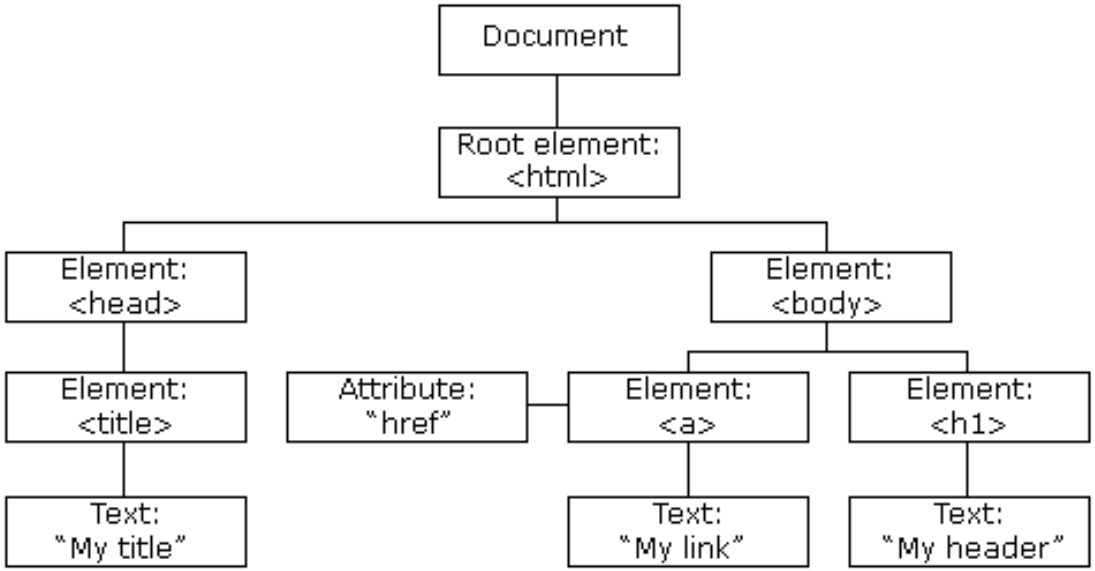
Notion d'héritage

- Un élément est un objet **HTMLDivElement**
- Un objet, en Javascript, peut appartenir à différents groupes.
- Un élément est un HTMLDivElement, qui est un sous-objet d'**HTMLElement** qui est lui-même un sous-objet d'**Element**. Element est enfin un sous-objet de **Node**.
- L'objet Node apporte un certain nombre de propriétés et de méthodes **communes** à tous les sous-objets.
- Les sous-objets *héritent* les propriétés et les méthodes de leurs objets parents.



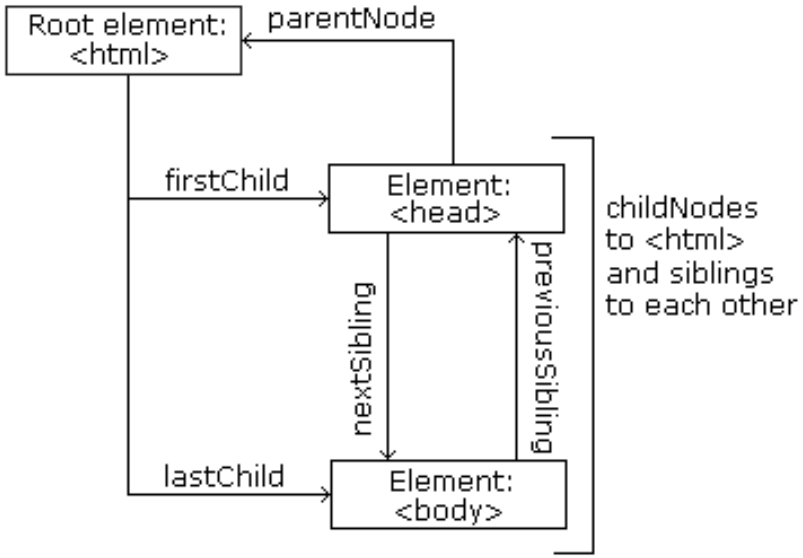
Naviguer entre les nœuds (1)

- Dans un document HTML, tout est un nœud:
  - Le document entier est un nœud "document"
  - Chaque élément HTML est un nœud "Element«
  - Le contenu d'un élément est un nœud texte
  - Chaque attribut est un nœud "attribute"
  - Chaque commentaire est un nœud "comment"



Naviguer entre les nœuds (2)

- Les nœuds dans un arbre DOM sont reliés par des relations: parent, frères et enfants
- L'élément "html" est l'élément racine
- Chaque nœud possède un parent sauf pour la racine
- Un nœud peut avoir des nœuds enfants
- Les nœuds frères (ou sœurs) possèdent le même parent



- On peut accéder au document entier par l'une des propriétés suivantes:
  - document.body: le corps du document
  - document.documentElement: le document entier

Naviguer entre les nœuds (3)

```
<html>
  <head>
    <title>Exemple DOM</title>
  </head>
  <body>
    <h1>Première partie</h1>
    <p id="p1">Bienvenue!</p>
  </body>
</html>
```

Dans l'exemple ci-dessous:

- <html> est la racine et n'a pas de parent
- <html> est le nœud parent de <head> et <body>
- <head> est le premier fils, <body> est le dernier fils de <html>
- <head> possède un seul enfant <title>
- <title> a un seul enfant (nœud texte): "Exemple DOM"
- <body> a deux enfant: <h1> et <p>
- <h1> a un seul enfant (nœud texte): "Première partie"
- <p> a un seul enfant (nœud texte): "Bienvenue!" et un nœud attribut (id):"p1"
- <h1> et <p> sont des nœuds frères

Naviguer entre les nœuds (4)

Pour naviguer entre les nœuds, on peut utiliser les propriétés suivantes:

- **parentNode** : le nœud parent
- **childNodes**: un tableau contenant les nœuds enfants
- **firstChild**: le premier nœud enfant
- **lastChild**: le dernier nœud enfant
- **nextsibling**: le nœud frère suivant
- **previousSibling**: le nœud frère précédent

Naviguer entre les nœuds (5)

Exemple:

```
<body>
    <h1>Exemple</h1>
    <div id="entete">
        un peu de texte
        <p>un paragraphe</p>
        <a href="#">un lien</a>
        encore du texte
    </div>
    <div id="main">
        une deuxième division
    </div>
<script>
    var entete = document.querySelector('#entete');
    console.log(entete.parentNode.nodeName);    //affiche: BODY
    console.log(entete.previousSibling.previousSibling);//affiche: <h1>
    console.log(entete.nextSibling.nextSibling);//affiche: <div id="main">
    console.log(entete.childNodes.length);// affiche :5
    console.log(entete.firstChild);// affiche : "un peu de texte"
    console.log(entete.lastChild);// affiche : "encore du texte"

</script>
```

Remarque :Les espaces et retours à la ligne effectués dans le code HTML sont généralement considérés comme des nœuds textuels par les navigateurs.

Naviguer entre les nœuds (6)

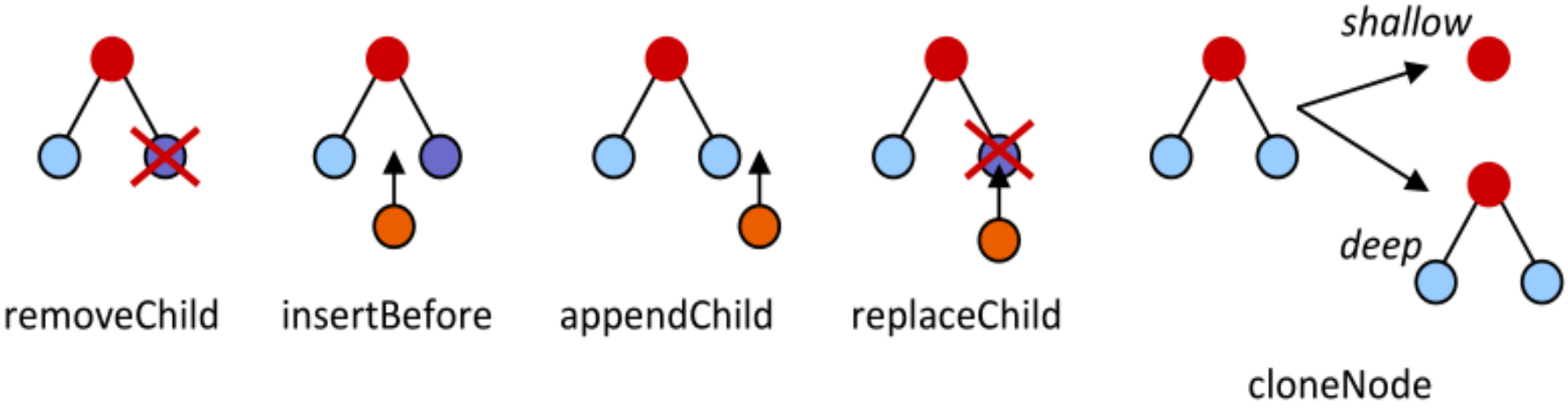
L'objet "document" propose des "raccourcis" pour accéder à certains éléments d'une page web:

- **document.head**: l'élément <head>
- **document.body**: l'élément <body>
- **document.title**: l'élément <title>
- **document.images**: collection d'éléments <img>
- **document.links**: collection d'éléments <a> et <area> ayant un attribut href
- **document.forms**: collection d'éléments <form>
- **document.anchors**: collection d'éléments <a> ayant un attribut name

Modification de l'arbre DOM

- Sur n'importe quel nœud « n » :

```
n.removeChild(nodeToRemove) → objet supprimé
n.insertBefore(nodeToInsert, childRef) → objet inséré
n.appendChild(nodeToAppend) → objet inséré
n.replaceChild(newNode, oldNode) → objet remplacé
n.cloneNode(deepOrNot) → objet résultat de la copie
```



Créer et insérer des éléments

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

- On crée l'élément.
- On lui affecte des attributs.
- On l'insère dans le document

1. 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 :

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

Créer et insérer des éléments

2. Affectation des attributs :

on définit les attributs, soit avec *setAttribute()*, soit directement avec les propriétés adéquates.

```
newLink.id    = 'mns_link';
newLink.href  = 'http://www.monsite.com';
newLink.title = 'Découvrez mon site !';
```

Créer et insérer des éléments

3. Insertion de l'élément :

On utilise la méthode *appendChild()* pour insérer l'élément.

appendchild 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 :

```
1  <!doctype html>
2  <html>
3    <head>
4      <meta charset="utf-8" />
5      <title>Le titre de la page</title>
6    </head>
7
8    <body>
9      <div>
10       <p id="myP">Un peu de texte <a>et un lien</a></p>
11     </div>
12   </body>
13 </html>
```

Créer et insérer des éléments

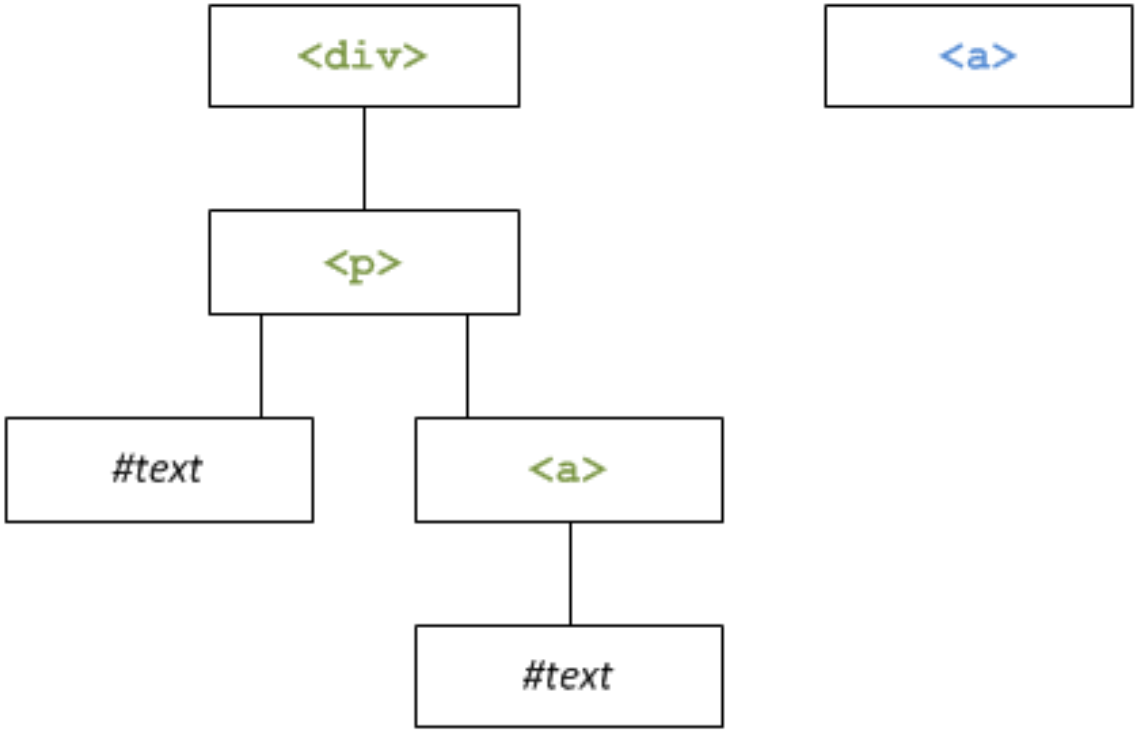
Insertion de l'élément :

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

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

Créer et insérer des éléments

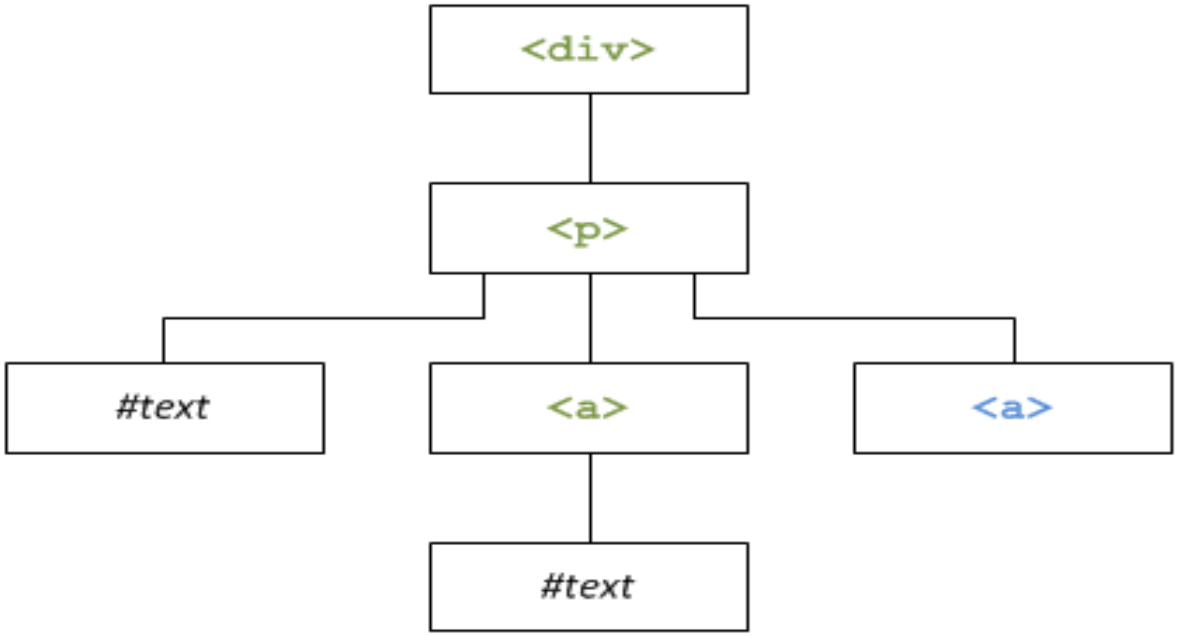
Explication : avant d'insérer notre élément <a>, la structure DOM du document ressemble à ceci :



Avant d'insérer notre élément <a>, la structure DOM du document ressemble à cette figure

Créer et insérer des éléments

On voit que l'élément <a> existe, mais n'est pas lié. Un peu 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 :



L'élément est placé en tant que dernier enfant

Créer et insérer des éléments

Ajouter des nœuds textuels :

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

```
var newLinkText = document.createTextNode("mon site");

newLink.appendChild(newLinkText);
```

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 !';
  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>
```

Résultat :

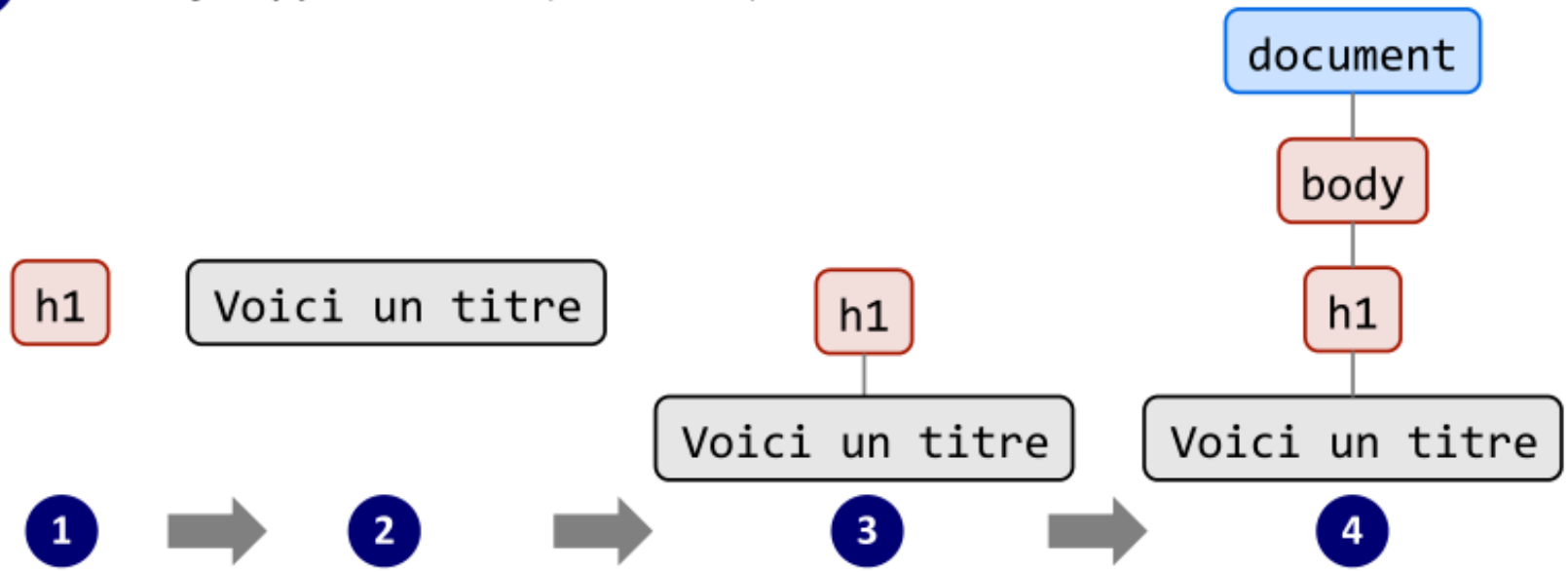
Un peu de texte <a>et un lien <a id=" sdz_link"

href="http://blog.crdp-versailles.fr/rimbaud" title="Découvrez le blog de la Classe Actu

! ">Le Tonnerre de Rimbaud</

Exemple : création d'un nœud <h1>

```
var d = window.document;

1 var noeudH1 = d.createElement("h1");
2 var noeudTexte = d.createTextNode("Voici un titre");

3 noeudH1.appendChild(noeudTexte);

4 d.body.appendChild(noeudH1);
```



ajouter des éléments dans une page

```
// recherche du noeud parent
var divParent = document.getElementById('divParent');

// création des nouveaux noeuds
var nouveauDiv = document.createElement('div');
var nouveauLabel = document.createElement('label');
var nouveauInput = document.createElement('input');

// paramétrage des nouveaux noeuds
nouveauLabel.appendChild(document.createTextNode("Mon nouveau label :"));
nouveauLabel.htmlFor = 'nouveauId';

nouveauInput.name = 'nouveau';
nouveauInput.id = 'nouveauId';
nouveauInput.type = 'text';

// raccord des noeuds
divParent.appendChild(nouveauDiv);
nouveauDiv.appendChild(nouveauLabel);
nouveauDiv.appendChild(nouveauInput);
```

Cloner un élément

- cloneNode() : Cette méthode requiert un paramètre booléen (true ou false) : si vous désirez cloner (copier) le nœud avec (true) ou sans (false) ses enfants et ses différents attributs.
- Exemple : on crée un élément <hr />, et on veut un deuxième, donc on clone le premier :

```
<script>
// 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);
</script>
```

Remplacer un élément par un autre

- replaceChild() : Cette méthode accepte deux paramètres : le 1 est le nouvel élément, et le 2 est l'élément à remplacer. Cette méthode s'utilise sur tous les types de nœuds (éléments, nœuds textuels, etc.).
- Exemple : le contenu textuel (pour rappel, il s'agit du premier enfant de <a>) va être remplacé par « et un hyperlien ». La méthode replaceChild() est exécutée sur l'élément <a>, c'est-à-dire le nœud parent du nœud à remplacer.

```
<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>
```

Résultat :

Un peu de texte <a>et un hyperlien

Supprimer un élément

- Pour en supprimer un élément, on utilise *removeChild()*. Cette méthode prend en paramètre le nœud enfant à retirer. Soit le code HTML avec un script ressemble à ceci :

```
1 var link = document.getElementsByTagName('a')[0];
2
3 link.parentNode.removeChild(link);
```

- la méthode removeChild() retourne l'élément supprimé, ce qui veut dire qu'il est parfaitement possible de supprimer un élément HTML pour ensuite le réintégrer où on le souhaite dans le DOM :

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

var oldLink = link.parentNode.removeChild(link);
// On supprime l'élément et on le garde en stock

document.body.appendChild(oldLink);
// On réintègre ensuite l'élément supprimé où on veut et quand on veut
</script>
```

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 :

```
<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 un élément avant un autre

- 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. Exemple :

```
<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>
```

Résultat :

Un peu de texte > en emphase légère <a>et un hyperlien