Université de Msila

Faculté des mathématiques et de l'informatique

Département d'informatique

3ième année Licence ISIL

Document Object Model (DOM)

Présenté par : Meliouh.A

Le Document Object Model (DOM) 2

- Le **Document Object Model** (abrégé **DOM**) est une interface de programmation pour les documents XML et HTML.

- Une interface de programmation, qu'on appelle aussi une API (pour Application Programming Interface), est un ensemble d'outils qui permettent de faire communiquer entre eux plusieurs programmes ou, dans le cas présent, différents langages.

- Le DOM est donc une API qui s'utilise avec les documents XML et HTML, et qui va nous permettre, via le JavaScript, d'accéder au code XML et/ou HTML d'un document.

- C'est grâce au DOM que nous allons pouvoir modifier des éléments HTML (afficher ou masquer un par exemple), en ajouter, en déplacer ou même en supprimer.

Le Document Object Model (DOM) 3

- **Qu'est-ce que le DOM ?**

- Le Document Object Model d'une page web va être créé automatiquement par le navigateur lors du chargement de la page.

- Le navigateur lit donc le html, puis en dégage une structure qu'il mémorise (arbre). Lorsque l'on interagit ensuite avec la page par du code (javascript principalement), on agit sur le dom, pas sur le code html d'origine.

- Le DOM HTML est un standard de programmation reconnu par tous et considère les éléments HTML comme des **objets**. Ainsi, les éléments HTML vont posséder des propriétés et des méthodes.

- Le DOM HTML va également permettre de déclencher ce qu'on appelle des **événements**,

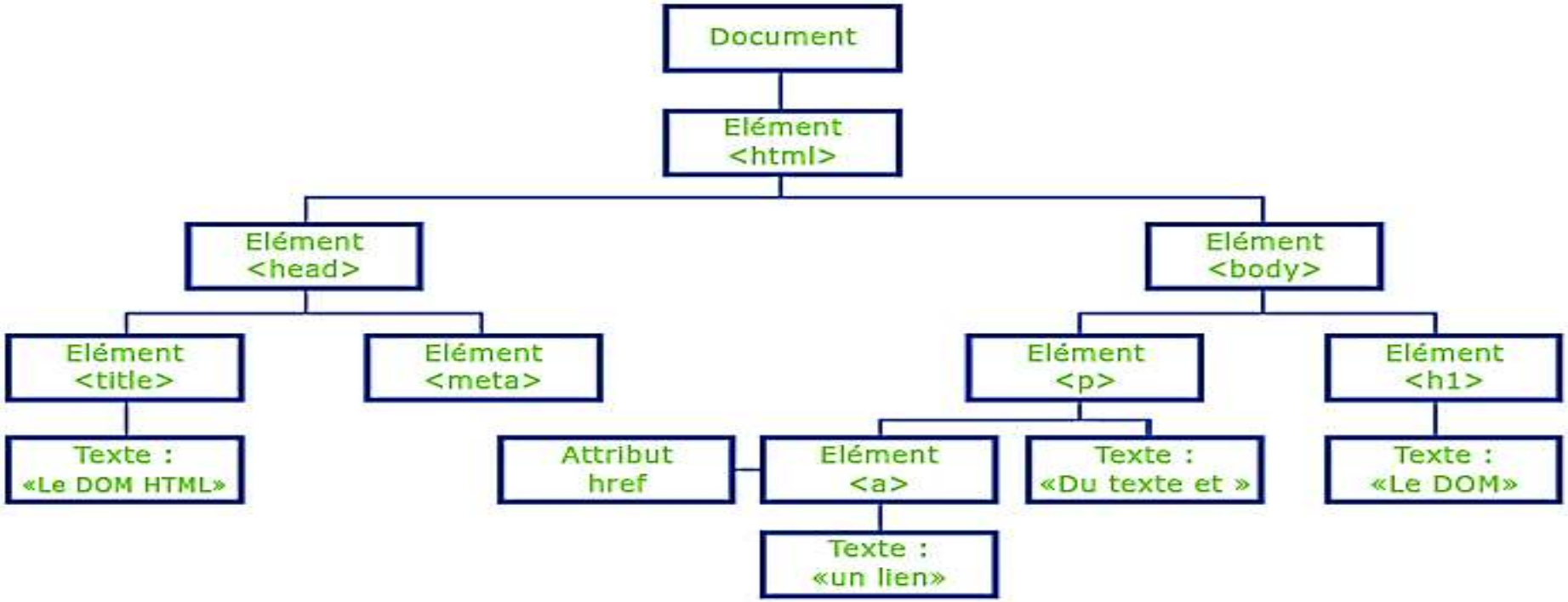
Représentation d'un document

- Un document HTML, aux yeux du DOM, est un arbre composé de **nœuds (node)**, qui représentent les ***balises*** (appelées ***éléments*** ou **ELEMENT_NODE**), le ***texte*** (**TEXT_NODE**), et les ***attributs***.
- prenons l'exemple d'une page HTML très simple et observons sa représentation sous forme de structure DOM

```
<!DOCTYPE html>
<html>
    <head>
        <title>Le DOM HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Le DOM</h1>
        <p>Du texte et <a href="http://pierre-giraud.com">un lien</a></p>
    </body>
</html>
```

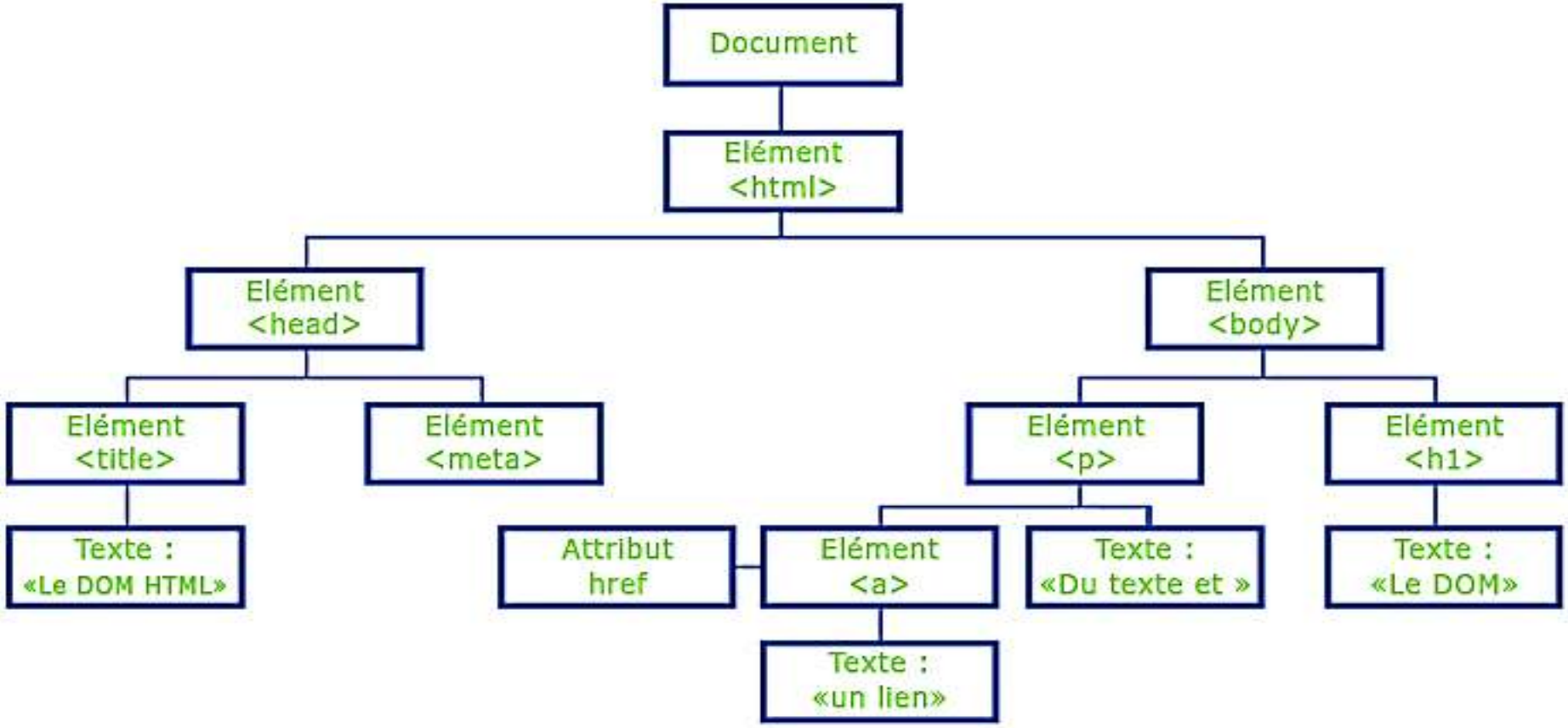
Représentation d'un document


```
<!DOCTYPE html>
<html>
    <head>
        <title>Le DOM HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>Le DOM</h1>
        <p>Du texte et <a href="http://pierre-giraud.com">un lien</a></p>
    </body>
</html>
```



Représentation d'un document

- voici le DOM relatif à cette page qui va être créé par le navigateur :



- On voit un premier nœud de type **Element** représenté par l'élément html. Ce nœud possède deux **enfants** : head et body qui sont également deux nœuds de type Element et qui possèdent eux mêmes d'autres enfants.
- Notez que head et body sont des nœuds « **frères** » : ils ont le même parent mais il n'y a pas de relation de hiérarchie entre eux.

Accéder / ajouter, insérer / modifier /supprimer un élément HTML

ACCEDER A UN ELEMENT HTML

- Pour accéder aux éléments de notre page HTML, nous allons toujours devoir passer par l'objet **Document.**
- L'objet **document** possède les méthodes principales :

1. La méthode **getElementById() ;**
2. La méthode **getElementsByTagName() ;**
3. La méthode **getElementsByName() ;**
4. La méthode **getElementsByClassName() ;**
5. La méthode **querySelector() ;**
6. La méthode **querySelectorAll().**

ACCEDER A UN ELEMENT HTML

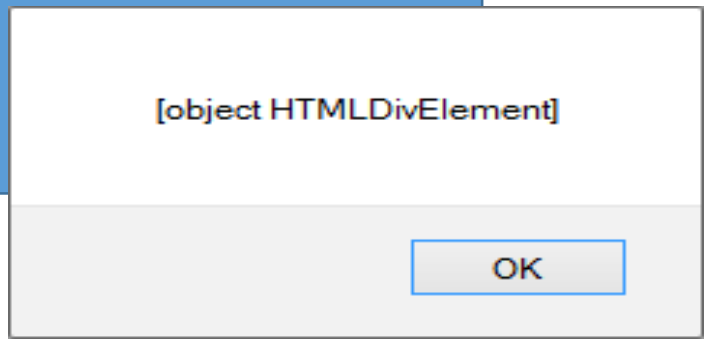
La méthode getElementById() :

- va nous permettre de cibler un élément HTML possédant un attribut **id** en particulier. C'est certainement la méthode la plus utilisée.
- Si l'élément est trouvé, getElementById() va renvoyer l'élément en tant qu'objet. Si aucun élément n'est trouvé, la méthode renverra la valeur **null**.

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

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

    alert(div);
</script>
```



La méthode getElementsByTagName() :

- Faites très attention dans le nom de cette méthode : il y a un « **s** » à Element**s**. C'est une source fréquente d'erreurs.
- Cette méthode permet de récupérer, sous la forme d'un **tableau**, tous les éléments de la famille.

```
var divs = document.getElementsByTagName('div');

for (var i = 0 ; i < divs.length  ; i++) {
    alert('Element n° ' + (i + 1) + ' : ' + divs[i]);
}
```

- **Deux petites astuces :**

1. Cette méthode est accessible sur n'importe quel élément HTML et pas seulement sur l'objet *document*.
2. En paramètre de cette méthode vous pouvez mettre une chaîne de caractères contenant un astérisque * qui récupérera tous les éléments HTML contenus dans l'élément ciblé.

ACCEDER A UN ELEMENT HTML

La méthode getElementsByName() :

- permet de ne récupérer que les éléments qui possèdent un attribut **name** que vous spécifiez.

La méthode getElementsByClassName () :

- Permet d'accéder aux éléments HTML disposant d'un attribut **class** en particulier. Cette méthode est une méthode de l'objet Document.

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p>Du texte et <a href="http://pierre-giraud.com">un lien</a></p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        var tableau = document.getElementsByClassName('para');
        alert('Notre page contient ' + tableau.length +
              ' paragraphe portant la class "para"');
    </script>
</body>
```

ACCEDER A UN ELEMENT HTML

Les méthodes querySelector() et querySelectorAll()

- Les méthodes **querySelector()** et **querySelectorAll()** permettent d'accéder à des éléments HTML correspondant à un certain sélecteur CSS, que ce soit un **id**, une **class**, un **type d'élément**, un **attribut** ou autre.
- Les deux méthodes sont récentes et ne sont pas supportées par les très vieilles versions des navigateurs, leur support commence à partir de la version 8 d'Internet Explorer, pour les autres navigateurs vous n'avez normalement pas de soucis à vous faire.
- **querySelector()** va renvoyer des informations relatives au **premier** élément trouvé correspondant au sélecteur CSS sélectionné,
- tandis que **querySelectorAll()** va renvoyer des informations sur **tous** les éléments correspondants.
- Une nouvelle fois, un tableau va être créé lorsqu'on utilise querySelectorAll().

ACCEDER A UN ELEMENT HTML

Les méthodes querySelector() et querySelectorAll()

```
<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>
```

```
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); //
Affiche : "Élément 1 - Élément 2"
```

ACCEDER A UN ELEMENT HTML

Les méthodes querySelector() et querySelectorAll()

- **Rappel sur les sélecteurs**

*	Sélectionne tout	
tag	Toute balise <tag>	p
tag.class	Tout <tag> de classe class	p.intro
#id	La balise identifiée par id	#firstname
tag:pseudoclass	Sélection de contenu spécial	p::first-letter, a:visited, p:first-child, input:checked,
tag[att=val]	Tout <tag> ayant attribut att égal à val	[target], [title~=flower], a[href^="https"], a[href$=".pdf"],
sel1, sel2	Chacun des sélecteurs	div, p
parent child	child s'il est un fils de parent	div p
parent > child	child seulement s'il est un fils direct	div > p
sister ~ brother	brother s'il suit sister	p ~ ul
sister + brother	brother s'il suit immédiatement sister	div + p

Les méthodes querySelector() et querySelectorAll()

- **Remarque**
  - **getElementById('id')** est équivalent à **querySelector(#id)**,
  - **getElementsByName('name')** est équivalent à **querySelector([name=...])**,
  - **getElementsByTagName('tag')** est équivalent à **querySelectorAll(tag).**
  - **getElementsByClassName** est équivalent à **querySelectorAll(.class).**

La propriété innerHTML

- **innerHtml** est une propriété des éléments DOM, et représente le code HTML compris *à l'intérieur* d'une balise. Par exemple, si j'ai la balise :

```
<ul id='maliste'>
<li> un</li>
<li> deux</li>
</ul>
```

Alors, **document.getElementById("maliste").innerHTML** vaut:

```
<li> un</li>
<li> deux</li>
```

- L'intérêt de cette propriété est qu'on peut l'utiliser pour modifier le contenu d'une balise.

La propriété innerHTML

- **Exemple 2**:

```
<p><span id="testInnerHTML">
Du texte à remplacer.... </span></p>

<p><button onclick="javascript:remplacerInnerHTML()">remplacer
texte</button></p>

<script type="text/javascript">
      function remplacerInnerHTML() {
            document.getElementById('testInnerHTML').innerHTML=
            "le <b>texte de remplacement</b>";
      }
</script>
```

Modifier la valeur d'un attribut HTML

- Pour modifier la valeur d'un attribut HTML, il suffit d'affecter une nouvelle valeur à l'attribut ciblé par son nom (href par exemple). .

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte et <a href="http://pierre-giraud.com">un lien</a></p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        //On modifie la valeur de l'attribut href de notre lien
        document.querySelector('a').href = 'http://wikipedia.org';
    </script>
</body>
```

- **Attention** Si vous souhaitez modifier la valeur d'un attribut **class** : ce mot est un mot réservé en JavaScript et on ne peut donc pas l'utiliser. Utiliser **className** à sa place.

Modifier le css d'un attribut HTML

- Pour ajouter ou modifier le style CSS d'un élément HTML, on va utiliser la propriété style de l'objet Element suivie de la propriété CSS à ajouter / modifier.

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte et <a href="http://pierre-giraud.com">un lien</a></p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        // On veut que notre titre s'affiche en orange
        document.getElementById('gros_titre').style.color = 'orange';

        //On attribue une taille de 40px à notre titre
        document.getElementById('gros_titre').style.fontSize = '40px';
    </script>
</body>
```

- **Attention** lorsqu'on utilise cette propriété, il faut **supprimer le tiret** des propriétés CSS qui en contiennent et mettre une majuscule au deuxième mot (font-size devient fontSize par exemple)..

Créer un nouvel élément HTML

Pour créer un nouvel élément HTML en JavaScript, nous utiliserons généralement la méthode **createElement()** de l'objet Document.

Cette méthode va prendre en argument le nom de l'élément HTML que l'on souhaite créer.

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte</p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        //On crée un élément de type p
        document.createElement('p');
    </script>
</body>
```

- **Remarque:** Nous avons ici créé un nouvel élément p. Cependant, l'élément ne contient pour le moment ni attribut ni contenu textuel, et n'a pas encore été inséré à l'intérieur de la page à un endroit précis.

Ajouter un attribut et du texte à un élément HTML

- Pour ajouter des attributs, nous allons procéder de la meme façon que pour modifier la valeur d'un attribut HTML ou exceptionnellement avec la méthode **setAttribute().**
- Pour ajouter du texte, nous allons utiliser la méthode **createTextNode()** qui, comme son nom l'indique, va créer un nouveau nœud de type texte.

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte</p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        //On crée un élément de type p
        var newPara = document.createElement('p');

        //On ajoute un attribut id à notre paragraphe
        newPara.id = 'nouveau';

        //On crée un noeud de type texte
        var texte = document.createTextNode('Inséré !');
    </script>
</body>
```

- il nous reste donc à insérer le texte dans le nouveau paragraphe puis le paragraphe dans la page.

Insérer du texte et un élément dans une page HTML

- Pour insérer le texte dans le nouveau élément et l'élément dans le flux de la page, nous allons utiliser la méthode **appendChild().**
- La méthode appendChild() va insérer un objet en tant que dernier enfant d'un autre objet. Cette méthode appartient à l'objet Element et va prendre le nom de l'objet à insérer en argument.

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte</p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        //On crée un élément de type p
        var newPara = document.createElement('p');

        //On ajoute un attribut id à notre paragraphe
        newPara.id = 'nouveau';

        //On crée un noeud de type texte
        var texte = document.createTextNode('Inséré !');

        //On insère le texte dans notre paragraphe
        newPara.appendChild(texte);

        /*On insère finalement notre élément en tant que
         *dernier enfant de body (auquel on accède directement
         *avec "document.body", tout simplement)*/
        document.body.appendChild(newPara);
    </script>
</body>
```

Insérer un élément HTML à un endroit précis

- Pour insérer un élément dans un endroit précis d'une page HTML on peut utiliser la méthode **insertBefore()**, qui va insérer un objet juste avant un élément comme son nom l'indique.

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte</p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        //On crée un élément de type p
        var newPara = document.createElement('p');

        //On ajoute un attribut id à notre paragraphe
        newPara.id = 'nouveau';

        //On crée un noeud de type texte
        var texte = document.createTextNode('Inséré !');

        //On insère le texte dans notre paragraphe
        newPara.appendChild(texte);

        //On accède à notre premier paragraphe
        var para1 = document.querySelector('.para');

        //On insère notre nouveau paragraphe juste avant
        document.body.insertBefore(newPara, para1);
    </script>
</body>
```

Supprimer un élément HTML en JavaScript

- Pour retirer un élément HTML d'une page en JavaScript, nous allons utiliser la méthode removeChild().
- Cette méthode va supprimer un élément HTML enfant ciblé relativement à son parent. Nous allons donc appliquer cette méthode à partir de l'élément parent comme ceci :

```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte</p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        /*On veut supprimer notre titre, on commence donc
         *par y accéder*/
        var titre = document.getElementById('gros_titre');

        //On accède ensuite à l'élément parent de h1 (body)
        var parent = document.body;

        //On supprime finalement notre titre avec removeChild
        parent.removeChild(titre);
    </script>
</body>
```

Modifier un élément HTML en JavaScript

- Pour modifier ou remplacer des nœuds / éléments HTML par d'autres, on va utiliser la méthode replaceChild(),
- Cette méthode va prendre deux arguments : la valeur de remplacement et le nœud qui doit être remplacé.


```
<body>
    <h1 id="gros_titre">Le DOM</h1>
    <p class="para">Du texte</p>
    <p class="para">Un deuxième paragraphe</p>

    <script>
        //On accède à notre élément h1 en JavaScript
        var titre = document.getElementById('gros_titre');

        //On accède ensuite à l'élément parent de h1 (body)
        var parent = document.body;

        //On crée une valeur de remplacement
        var  nouveauTitre = document.createElement('h2');

        //On ajoute du texte et un id à notre h2
        nouveauTitre.id = 'titre_moyen';
        nouveauTitre.innerHTML = 'Titre modifié en Js !'

        //On remplace finalement h1 par h2
        parent.replaceChild(nouveauTitre, titre);
    </script>
</body>
```

Navigation dans l'arbre DOM

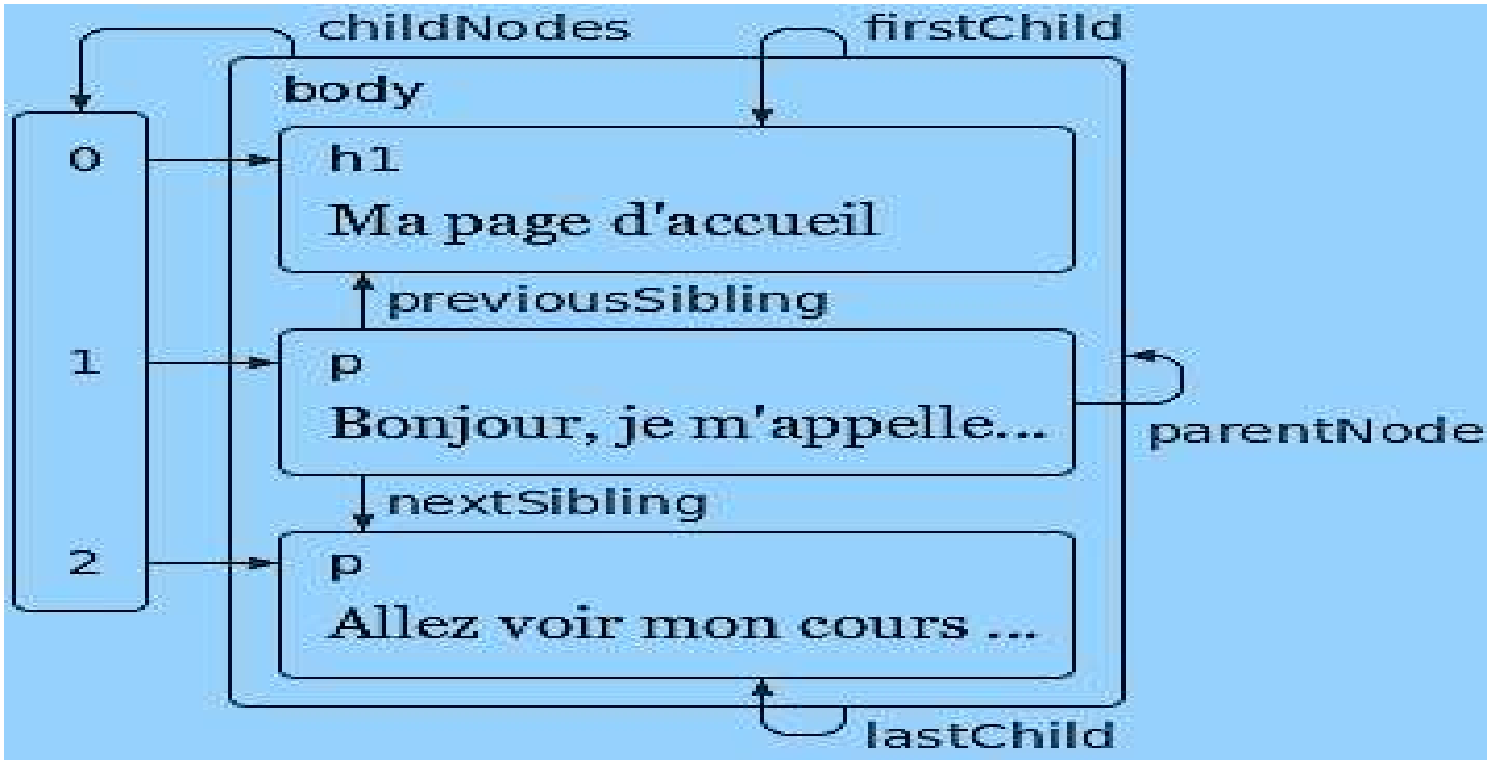
Navigation dans l'arbre

- Les méthodes de base pour naviguer dans l'arbre:

  Fils : children, firstElementChild, lastElementChild, …

  Sœurs : nextElementSibling, previousElementSibling,

  Parent : parentElement.



Navigation dans l'arbre

La propriété parentNode

- La propriété parentNode va nous permettre d'accéder au nœud parent d'un certain nœud, et donc de nous déplacer dans le DOM de notre page HTML.
- Cette propriété est une propriété de l'objet **Element**.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Le DOM HTML</title>
        <meta charset="utf-8">
    </head>
    <body>
        <h1 id="gros_titre">Le DOM</h1>

        <div>
            <p class="para">Du texte</p>
            <p class="para">Un deuxième paragraphe</p>
        </div>

        <script>
            //On accède à notre premier élément p
            var p = document.querySelector('.para');

            //On accède ensuite à notre div avec parentNode
            var div = p.parentNode;

            //On peut ensuite maniupler notre div à loisir
            div.style.color = 'orange';
        </script>
    </body>
</html>
```

La propriété childNodes

- La propriété **childNodes** permet d'accéder aux nœuds enfants d'un certain nœud HTML.
- Cette propriété va renvoyer un **tableau** contenant les enfants d'un certain nœud.

```
<html>
<body>
 <div>Begin</div>
 <ul>
     <li>Information</li>
 </ul>
 <div>End</div>
 <script>
  for (let i = 0; i < document.body.childNodes.length; i++) {
   alert( document.body.childNodes[i] );
  }
 </script>
 ...more stuff...
</body>
</html>
```

```
[object Text][object HTMLDivElement]
[object Text] [object HTMLUListElement]
[object Text][object HTMLDivElement]
[object Text][object HTMLScriptElement]
[object Text]
```

Les propriétés firstChild et lastChild

- Ces deux propriétés permettent d'accéder respectivement au premier et au dernier enfant d'un nœud.
- **Exemple**

```
<div id="myDIV">
  <p>Looks like first child</p>
  <span>Looks like last Child</span>
</div>

<script>
var x = document.getElementById("myDIV").firstChild.nodeName;
alert ( x ) ; // Affiche #text
</script>
```

- Si on enlève les blancs du code source, il y aura plus de nœuds #text dans et <p> deviendra le premier enfant(**firstChild**).

```
<div id="myDIV"><p>First child</p><span>Last Child</span></div>
```

```
var x = document.getElementById("myDIV").firstChild.nodeName; // x= P
var y = document.getElementById("myDIV").lastChild.nodeName; // y= span
```

Les propriétés nextSibling et previousSibling

- Ces deux propriétés permettent d'accéder respectivement au nœud « frère » (c'est-à-dire de même niveau) suivant le nœud ciblé ou à celui précédent le nœud ciblé.
- **Exemple**

```
<ul>
<li id="item1">Coffee (first li)</li>
<li id="item2">Tea (second li)</li>
</ul>
<script>
var x = document.getElementById("item1").nextSibling.innerHTML;
var y = document.getElementById("item2").previousSibling.innerHTML;
alert(x); // affiche Tea (second li)
alert(y); // affiche Coffee (first li)
```

Navigation dans l'arbre

Autres propriétés

- **innerHTML** : non standard (mais implémenté partout), code HTML interne de l'élément ;
- **textContent** : contenu textuel de l'élément (tout le texte, en ignorant les balises). Pour IE: à partir d'IE 9.
- **parentNode** : noeud parent ;
- **childNodes** : tableaux des noeuds fils. Chaque noeud fils est, soit un noeud élément (de nodeType 1), soit un noeud texte (nodeType 3) ;
- **firstChild** : équivalent de childNodes[0] ;
- **lastChild** : dernier fils ;
- **nodeValue** : texte contenu dans un nœud texte. On peut modifier cette valeur.

Navigation dans l'arbre

Autres propriétés

- **nodeName** : nom de la balise ;
- **nodeType** : 1 pour éléments (balises) et 3 pour un noeud texte;
- **id** : identifiant du noeud ;
- **className** : classe (au sens CSS) de l'élément ;
- **style** : accès aux propriétés CSS de l'élément ;
- **getAttribute(NOM)** : valeur d'un de ses attributs. Ne fonctionne pas pour l'attribut "class" sous IE.
- **setAttribute(NOM,VALEUR)** : pour un noeud/élément, permet de fixer la valeur d'un attribut. Ne fonctionne pas sous IE pour les styles css ni pour l'attribut "class".

Les Evènements en JavaScripT

LES EVENEMENTS EN JAVASCRIPT

- Les **évènements** correspondent à des actions effectuées soit par un utilisateur, soit par le navigateur lui-même.

- Par exemple, lorsqu'un utilisateur clique sur un bouton HTML ou lorsque le navigateur va finir de charger une page web, on va parler d'événement.

- Parfois, on va vouloir « attacher » une action spécifique à un évènement, comme par exemple modifier la taille des textes sur une page lorsque l'utilisateur clique sur un bouton.

Des événements courants

Nom de l'événement	Action pour le déclencher
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 de la souris sur l'élément
mousemove	Faire déplacer le curseur sur l'élément
load	chargement terminé d'une ressource et de ses dépendances
DOMContentLoaded	chargement terminé du DOM

Des événements courants

Nom de l'événement	Action pour le déclencher
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.)
input	Taper un caractère dans un champ de texte
select	Sélectionner le contenu d'un champ de texte (input,textarea, etc.)

LES EVENEMENTS EN JAVASCRIPT

Exemple


```html
<!DOCTYPE html>
<html>
    <head>
        <title>Les évènements</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1 id="gros_titre">Les évènements</h1>

        <p onclick="alert('Bravo !');">Cliquez-moi, cliquez-moi !</p>
        <p>Un deuxième <strong>paragraphe</strong></p>
    </body>
</html>
```

L'utilisation du mot clef " this " dans la gestion d'évènements

Dans le contexte de la gestion d'événements, **this** va faire référence à l'objet (représentant l'élément HTML) qui est le sujet de l'événement.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Les évènements</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1 id="gros_titre">Les évènements</h1>

        <p onclick="this.textContent='Merci !';">Cliquez-moi, cliquez-moi !</p>
        <p>Un deuxième <strong>paragraphe</strong></p>
    </body>
</html>
```

La méthode JavaScript addEventListener()

- La méthode **addEventListener()** va nous permettre de lier du code à un évènement. On parlera alors de **gestionnaire d'évènements**.
- Le code sera alors exécuté dès le déclenchement de l'évènement.
- Cette méthode appartient à l'objet **Element** et va avoir besoin de deux arguments pour fonctionner :
  - Le nom de l'évènement déclencheur de l'action et
  - Le code relatif à l'action à effectuer.
- Les événements vont une nouvelle fois avoir des noms similaires aux attributs HTML **mais ne vont plus être précédés du « on »** (par exemple, onclick devient click).

LES EVENEMENTS EN JAVASCRIPT

La méthode JavaScript addEventListener()

```
<body>
    <h1 id="gros_titre">Les évènements</h1>

    <p>Cliquez-moi, cliquez-moi !</p>
    <p>Un deuxième <strong>paragraphe</strong></p>
    <script>
        //On accède à notre premier paragraphe
        var p1 = document.querySelector('p');

        //On accroche un gestionnaire d'évènements à p1
        p1.addEventListener('click',changeTexte);

        /*On construit notre fonction changeTexte qui ne sera
         *exécutée que lors du déclenchement de l'évènement*/
        function changeTexte(){
            this.innerHTML = '<strong>Bravo !</strong>';
            this.style.color = 'orange';
        };
    </script>
</body>
```

Les évènements

Cliquez-moi, cliquez-moi !

Un deuxième **paragraphe**

Les évènements

Bravo !

Un deuxième **paragraphe**

La méthode JavaScript addEventListener()

- L'un des grands avantages de la méthode addEventListener() est de pouvoir lier plusieurs gestionnaires d'évènements **de même type** sur un élément HTML.

```
<script>
    //On accède à notre premier paragraphe
    var p1 = document.querySelector('p');

    //On accroche deux gestionnaires d'évènements à p1
    p1.addEventListener('click',Message1);
    p1.addEventListener('click',Message2);

    //Création des fonctions
    function Message1(){
        alert('Première boîte !');
    };

    function Message2(){
        alert('Deuxième boîte !');
    };
</script>
```

La méthode JavaScript addEventListener()

- Elle va également nous permettre de lier plusieurs **évènements différents** à un même élément HTML.

```
<script>
    //On accède à notre premier paragraphe
    var p1 = document.querySelector('p');

    //On accroche deux gestionnaires d'évènements à p1
    p1.addEventListener('mouseover',Fonction1);
    p1.addEventListener('mousedown',Fonction2);

    //Création des fonctions
    function Fonction1(){
        this.innerHTML = 'Cliquez moi maintenant !';
        this.style.backgroundColor = 'orange';
    };

    function Fonction2(){
        this.innerHTML = 'Bravo !';
        this.style.color = '#26C';
        this.style.fontWeight = 'bold';
        this.style.fontSize= '24px';
    };
</script>
```

L'objet Event

- L'objet **event** permet de fournir une multitude d'informations sur l'événement actuellement déclenché. Par exemple, vous pouvez récupérer

- quelles sont les touches actuellement enfoncées, les coordonnées du curseur, l'élément qui a déclenché l'événement… Les possibilités sont nombreuses !

-
```
p1.addEventListener('click', message);

Function message(e) {
// L'argument « e » va récupérer une référence vers l'objet « Event »
    alert(e.type);
// Ceci affiche le type de l'événement (click, mouseover, etc.)
};
```

L'objet Event

Quelques fonctionnalités de l'objet Event

- **currentTarget**: élément à qui le gestionnaire est associé ;
- **target**: élément qui a déclenché l'événement (peut être un fils de currentTarget) ;
- **type**: le type d'évènement qui a été déclenché.
- **clientX**, **clientY**: Récupérer la position du curseur (très utile lors du **drag&drop**)
- **which :** quelle touche a été appuyée ;

LES EVENEMENTS EN JAVASCRIPT

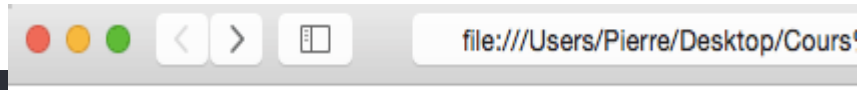
L'objet Event

```
<body>
    <h1 id="gros_titre">Les évènements</h1>
    <div id="div1">
        <p id="p1">Qui est le déclencheur ?
    </div>

    <script>
        //On accède à div1 et p1
        var div1 = document.getElementById('div1');
        var p1 = document.getElementById('p1');

        //On demande à div1 de réagir en cas de clic
        div1.addEventListener('click', Message);

        //On utilise target et currentTarget durant l'évènement
        function Message(event){
            this.innerHTML = 'Element déclencheur : ' + event.target +
            '<br/>Element portant l\'évènement : ' + event.currentTarget;
        }
    </script>
</body>
```



Les évènements

Element déclencheur : [object HTMLParagraphElement]
Element portant l'évènement : [object HTMLDivElement]

LES EVENEMENTS EN JAVASCRIPT

L'objet Event

```
<body>
    <h1 id="gros_titre">Les évènements</h1>
    <div id="div1">
        <p id="p1">Quel type d'évènement ?</p>
    </div>

    <script>
        //On accède à div1 et p1
        var div1 = document.getElementById('div1');
        var p1 = document.getElementById('p1');

        //On demande à div1 de réagir en cas de clic
        div1.addEventListener('click', Message);

        //On utilise target et currentTarget durant l'évènement
        function Message(event){
            this.innerHTML = 'Type d\'évènement déclenché : ' + event.type;
        }
    </script>
</body>
```

Browser Object Model (BOM)

Le Browser Object Model (BOM)

- **Qu'est-ce que le BOM ?**

- Le **BOM**, ou **B**rowser **O**bject **M**odel va nous permettre d'accéder au navigateur, et notamment à la fenêtre ouverte actuellement en utilisant le JavaScript.
- L'objet le plus célèbre et le plus utilisé du BOM est l'objet **Window**.
- L'objet **Window** va tout simplement représenter la fenêtre du navigateur.
- Vous devez savoir que toutes les fonctions, variables et objets **globaux** appartiennent automatiquement à l'objet Window.
- Cependant, l'objet Window est dit **implicite**. Cela signifie que nous n'aurons généralement pas besoin de le mentionner pour utiliser les méthodes (ou fonctions globales) et propriétés (ou variables globales) lui appartenant.

Le Browser Object Model (BOM)

- Qu'est-ce que le BOM ?

```html
<!DOCTYPE html>
<html>
    <head>
        <title>Le Browser Object Model</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1 id="gros_titre">Le BOM</h1>

        <script>
            //Ecrire ceci :
            window.alert('Avec window');

            //Est équivalent à cela :
            alert('Sans window');
        </script>
    </body>
</html>
```

Le Browser Object Model (BOM)

- **Qu'est-ce que le BOM ?**

```
<!DOCTYPE html>
<html>
    <head>
        <title>Le Browser Object Model</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1 id="gros_titre">Le BOM</h1>

        <script>
            //Ecrire ceci :
            window.document.getElementById('gros_titre');

            //Est équivalent à cela :
            document.getElementById('gros_titre');
        </script>
    </body>
</html>
```

- **Méthode open() de l'objet Window:**
- open(): ouvrir un nouvel onglet ou une nouvelle fenêtre.
- 4 arguments:
  1. URL de destination
  2. permettre de choisir où doit s'ouvrir notre nouvelle page (onglet courant, nouvel onglet, nouvelle fenêtre, etc.). La valeur qui va nous intéresser ici est **_blank** qui permet d'ouvrir une nouvelle fenêtre.
  3. liste d'éléments nous permettant d'agir directement sur la fenêtre ouverte (largeur, hauteur, position de la fenêtre, affichage d'une barre de menu ou pas, etc.).
  4. Finalement, le dernier argument va être un booléen spécifiant si la nouvelle fenêtre ouverte doit remplacer la fenêtre actuelle dans l'historique de navigation ou être ajoutée à celui-ci.

Le Browser Object Model (BOM)

- **Méthode open() de l'objet Window: exemple**

```
<body>
    <h1 id="gros_titre">Le BOM</h1>
    <p>Cliquez pour ouvrir un nouvel onglet !</p>

    <script>
        //On accède à notre paragraphe
        var para = document.querySelector('p');

        //On attache un gestionnaire d'évènement click
        para.addEventListener('click', fenetre);

        //On utilise open()
        function fenetre(){
            window.open('http://pierre-giraud.com', '_blank', 'width = 500, height = 300');
        }
    </script>
</body>
```

- **Méthode close() de l'objet Window:**
- La méthode close() va elle permettre de fermer un onglet / une fenêtre. Cette méthode n'a pas besoin d'argument.

Méthode resizeTo() de Window

- La méthode resizeTo() va nous permettre de redimensionner une fenêtre ou un onglet. Cette méthode s'utilise très simplement sur la fenêtre ou l'onglet de notre choix.

- Cette méthode va prendre comme arguments la largeur et la hauteur en pixels auxquelles on souhaite redimensionner notre fenêtre ou notre onglet.

Le Browser Object Model (BOM)

- **le BOM Contient les sous objets suivants :**

  - Screen
  - Navigator
  - Location
  - history
  - document

L'objets Screen :

- L'objet Screen nous donne accès à des informations concernant l'écran de vos visiteurs, comme la taille ou la résolution de l'écran par exemple.
- Connaître ces informations va nous permettre d'adapter nos pages web pour un affichage optimisé pour chaque visiteur.
- L'objet Screen possède six propriétés intéressantes :
  - **width** : retourne la largeur totale de l'écran ;
  - **availWidth** : retourne la largeur de l'écran moins celle de la barre de tâches ;
  - **height** : retourne la hauteur totale de l'écran ;
  - **availHeight** : retourne la hauteur de l'écran moins celle de la barre de tâches ;
  - **colorDepth** : retourne la profondeur de la palette de couleur de l'écran en bits ;
  - **pixelDepth** : retourne la résolution de l'écran en bits par pixel.

Le Browser Object Model (BOM)

L'objets Screen :

```
<!DOCTYPE html>
<html>
    <head>
        <title>Le Browser Object Model</title>
        <meta charset="utf-8">
    </head>

    <body>
        <h1 id="gros_titre">Le BOM</h1>
        <p></p>

        <script>
            var hauteur = screen.height;
            var hauteurDispo = screen.availHeight;
            var reso = screen.pixelDepth;

            var para = document.querySelector('p');
            para.innerHTML =
                'Hauteur de l\'écran : ' + hauteur +
                '<br>Hauteur dispo :' + hauteurDispo +
                '<br>Résolution : ' + reso + ' bits/px';
        </script>
    </body>
</html>
```

L'objets Navigator :

- L'objet **Navigator** va nous donner des informations sur le navigateur de vos visiteurs en soi ainsi que sur les préférences enregistrées (langue, etc.).
- L'objet Navigator possède dix propriétés :
  - **language** : retourne la langue définie dans le navigateur ;
  - **geolocation** : retourne un objet « geolocation » qui peut être utilisé pour définir la localisation de l'utilisateur ;
  - **product** : retourne le nom du moteur utilisé par le navigateur ;
  - **cookieEnabled** : détermine si les cookies sont autorisés ou non ;
  - **appName** : retourne le nom du navigateur ;
  - **appCodeName** : retourne le nom de code du navigateur ;
  - **appVersion** : retourne la version du navigateur utilisée ;
  - **online** : détermine si le navigateur est en ligne ou pas ;
  - **platform** : détermine pour quelle plateforme le navigateur est compilé ;
  - **userAgent** : retourne l'en-tête du fichier user-agent envoyé par le navigateur au serveur ;
- De plus, Navigator possède la méthode **javaEnabled()**, qui détermine si Java a été activé sur le navigateur ou pas.

Le Browser Object Model (BOM)

L'objets Navigator :

```
<body>
    <h1 id="gros_titre">Le BOM</h1>
    <p></p>

    <script>
        var langue = navigator.language;
        var navigateur = navigator.appName;
        var version = navigator.appVersion;
        var moteur = navigator.product;
        var cookieAutorise = navigator.cookieEnabled;

        var para = document.querySelector('p');
        para.innerHTML =
            'Langue utilisée : ' + langue +
            '<br>Nom du navigateur : ' + navigateur +
            '<br>Version : ' + version +
            '<br>Moteur : ' + moteur +
            '<br>Cookie ? : ' + cookieAutorise;
    </script>
</body>
```

L'objets Location :

- L'objet **Location** va nous fournir des informations relatives à l'URL de la page actuelle.
- L'objet Location possède neuf propriétés :
  - **hash**, qui retourne la partie ancre d'une URL ;
  - **search**, qui retourne la partie recherche de l'URL ;
  - **pathname**, qui retourne le chemin de l'URL ;
  - **href**, qui retourne l'URL complète ;
  - **hostname**, qui retourne le nom de l'hôte ;
  - **port**, qui retourne le port de l'URL ;
  - **protocole**, qui retourne le protocole de l'URL ;
  - **host**, qui retourne le nom de l'hôte et le port relatif à l'URL ;
  - **origin**, qui retourne le nom de l'hôte, le port et le protocole de l'URL.

- Location possède également trois méthodes très intéressantes :
  - **assign()** qui va charger un nouveau document ;
  - **reload()** qui va recharger le document ;
  - **replace()** qui va remplacer le document actuel par un autre.

Le Browser Object Model (BOM)


```
<body>
    <h1 id="gros_titre">Le BOM</h1>
    <button id='charger'>Charger un nouveau document</button>
    <button id='recharger'>Recharger le document</button>
    <button id='changer'>Remplacer le document</button>

    <p></p>

    <script>
        var para = document.querySelector('p');
        var charger = document.getElementById('charger');
        var recharger = document.getElementById('recharger');
        var changer = document.getElementById('changer');

        charger.addEventListener('click', charge);
        recharger.addEventListener('click', recharge);
        changer.addEventListener('click', change);
```

L'objets Location :

```
        function charge(){
            location.assign('http://pierre-giraud.com');
        }
        function recharge(){
            location.reload();
        }
        function change(){
            location.replace('http://pierre-giraud.com');
        }

        //Récupère l'url complète
        var url = location.href;

        //Récupère le chemin de la page
        var chemin = location.pathname;

        para.innerHTML = 'URL : ' +url+ '<br>Chemin : ' +chemin;
    </script>
</body>
```

L'objets History :

- **History** va nous permettre de nous déplacer dans l'historique de nos visiteurs et nous donner des informations relatives à celui-ci.
- L'objet Location possède neuf propriétés :
  - La propriété **length** nous permet de connaître le nombre d'URL dans l'historique ;
  - La méthode **back()** nous permet de charger la dernière URL disponible dans l'historique (l'URL la plus récente) ;
  - La méthode **forward()** nous permet de charger l'URL suivant une première URL dans l'historique (l'URL directement la plus récente par rapport à une première).
  - La méthode **go()** nous permet de charger une URL spécifique dans l'historique.

L'objets History :

```
<body>
    <h1 id='gros_titre'>Le BOM</h1>
    <button id='precedent'>Charge la dernière page visitée</button>
    <button id='suivant'>Charge la page suivante dans l'historique</button>
    <button id='specifique'>3 pages en arrière</button>

    <p></p>

    <script>
        var para = document.querySelector('p');
        var precedent = document.getElementById('precedent');
        var suivant = document.getElementById('suivant');
        var specifique = document.getElementById('specifique');

        precedent.addEventListener('click', arriere);
        suivant.addEventListener('click', avant);
        specifique.addEventListener('click', spec);
```

Le Browser Object Model (BOM)

L'objets History :

```
        function arriere(){
            history.back();
        }

        function avant(){
            history.forward();
        }

        function spec(){
            history.go(-5);
        }

        var histo = history.length;

        para.innerHTML = 'URLs : ' + histo;
    </script>
</body>
```

Référence

Cours Applications web et sécurité

http://defeo.lu/aws/

COURS COMPLET JAVASCRIPT

http://pierre-giraud.com/javascript/cours-complet/javascript-presentation.php