

Les APIs JAVASCRIPTS

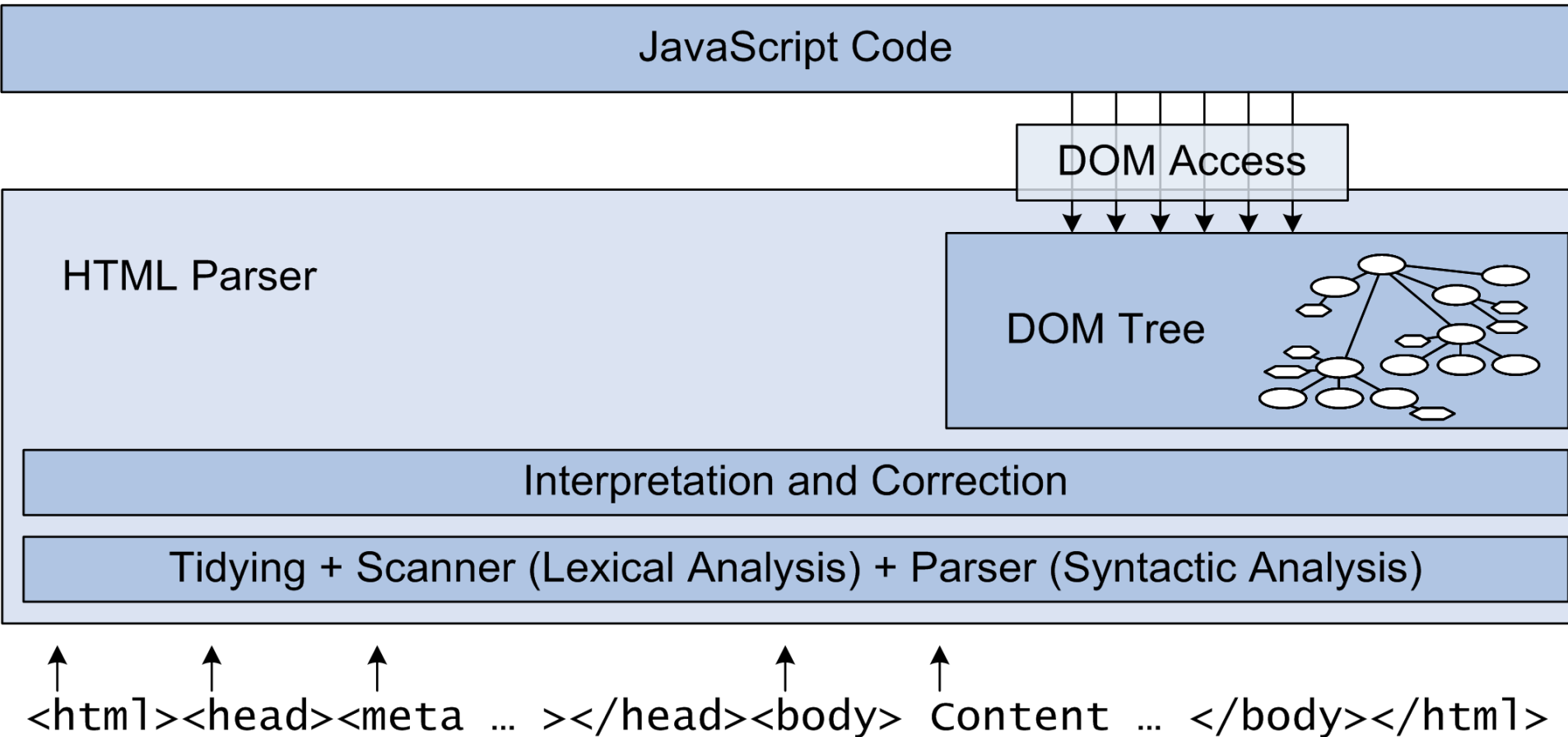
DOM

- Les versions de DOM:
 - DOM0 (proposé par Netscape); date de 1998, DOM1 la première version proposée par W3C
 - DOM2 la version standard et stable
 - DOM3
 - DOM4 (Recommandation 2015)
- DOM comprend deux parties :
 - Une partie DOM core : qui s'applique aussi à XML, avec quelques petites différences, et une autre spécifique au HTML.
 - Une partie DOM events (event handlers)

Les étapes d'interprétation des pages HTML

- Recevoir la page .html du serveur en tant que document de type text/HTML (MIME)
- Parser le document et corriger les différentes erreurs potentielles
- Interpréter le document comme si il contenait pas d'erreurs
- Apporter les ressources reliées à la page (avec un GET: CSS, images, JavaScript, ...)
- Construire l'arbre DOM du document
- Appliquer les règles de styles CSS (interne, externe etc.)
- Visualiser le document (structure)
- Commencer à exécuter le code JS
- Détecter les événements du clavier, souris, timer etc. et exécuter le code.
- Tout détruire et reprendre les mêmes étapes quand l'utilisateur quitte vers une page différente.

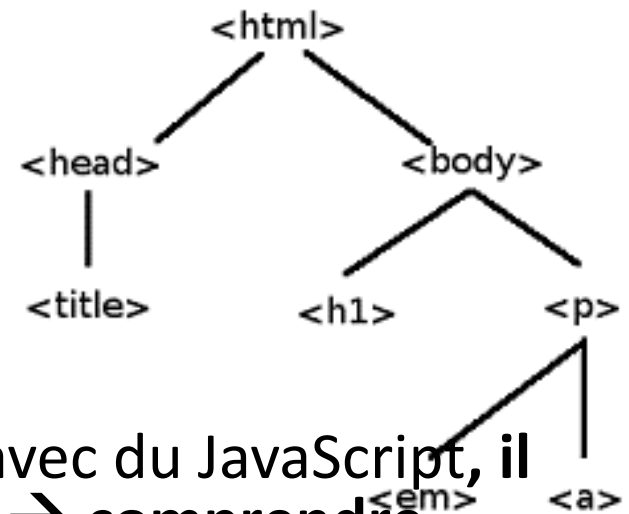
Fonctionnement interne du navigateur



Notion d'arbre

- On peut schématiser une page HTML par un arbre, comme celui-ci:

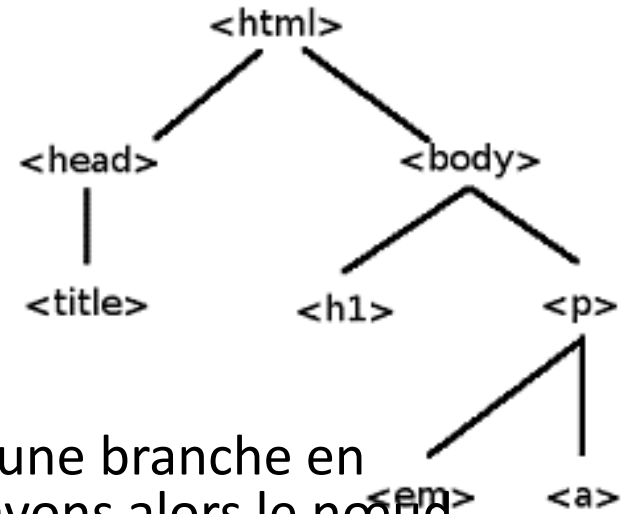
```
<html>  
  <head>  
    <title>...</title>  
  </head>  
  <body>  
    <h1>...</h1>  
    <p>...<em>...</em>...<a>...</a>  
  </body>  
</html>
```



- Pourquoi faire ?
- Lorsque l'on va manipuler ces pages avec du JavaScript, il **va falloir se balader dans notre page → comprendre comment la page est ordonnée devient alors très important.**

La notion de nœuds

- Dans notre exemple,
- HTML est le **parent** de HEAD et BODY.
- H1 est un **enfant** de BODY.



- En langage Javascript, chaque séparation d'une branche en plusieurs s'appelle un **node** (**nœud**). Nous avons alors le nœud HTML, le nœud HEAD, etc. Il existe deux grands types de node :
 - **élément** : les nœuds que l'on va appeler éléments sont les balises du HTML que vous connaissez.
 - **texte** : ces nœuds sont en fait du texte brut qui se trouve entre les balises.

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<!--je suis un commentaire-->
<title>Bienvenue!!</title>
</head>
<body>
<a href="http://www.w3schools.com/">
ceci est un lien vers le site W3C</a>
<h1>Ceci est un titre</h1>
<p>Ceci est un paragraphe.</p>
<b>Ce texte est en gras</b><br/>
<i>Ce texte est en italique</i><br/>
</body>
</html>

```



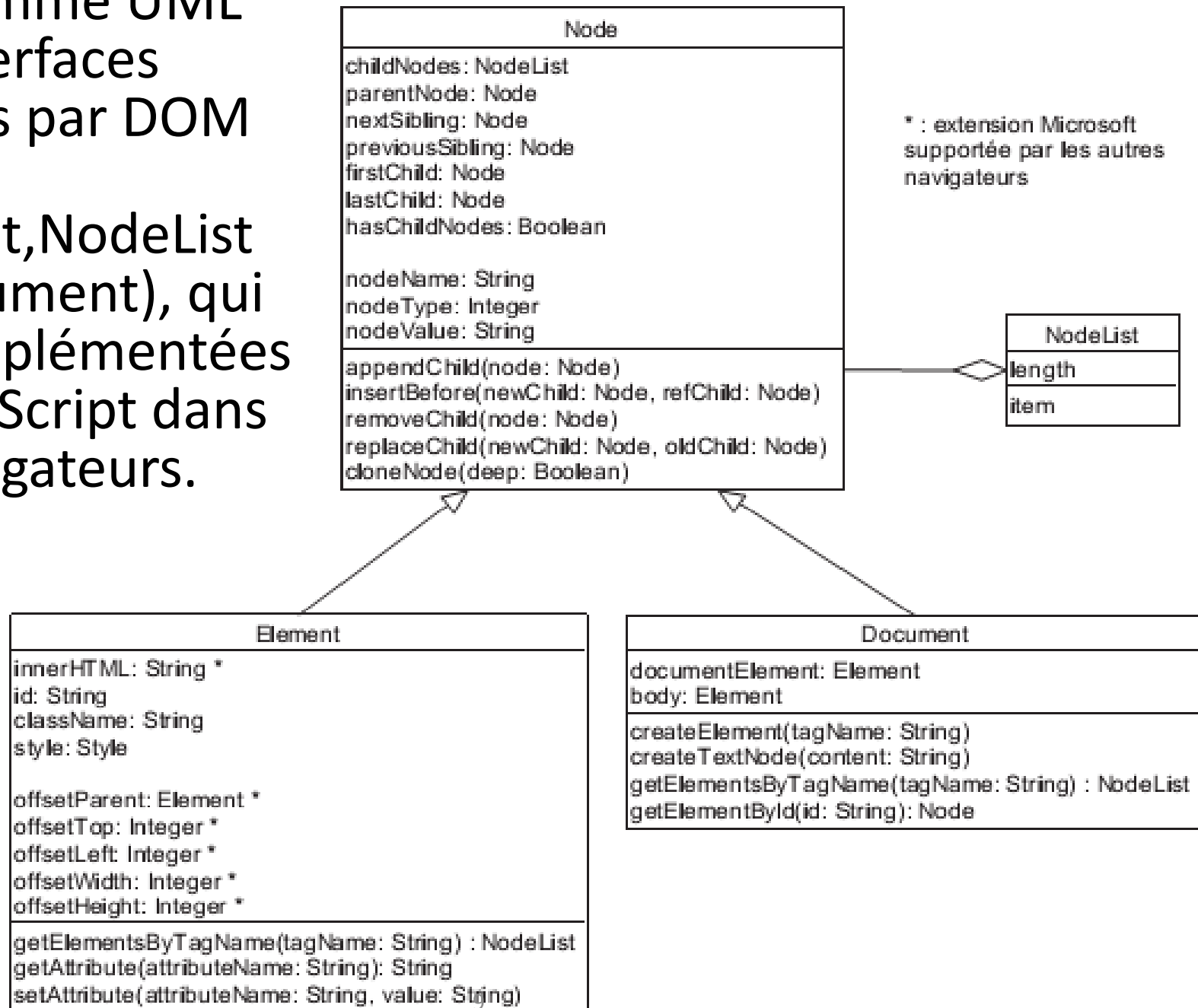
This DOM tree diagram was created using Ian Hickson's [Live DOM viewer](https://software.hixie.ch/utilities/js/live-dom-viewer/).

(<https://software.hixie.ch/utilities/js/live-dom-viewer/>)

Types de noeuds (node)

- **Element node:** An element, as it exists in the DOM.
- **Root node:** The top node in the tree, which in the case of HTML is always the HTML node (other markup vocabularies like XML will have different root elements).
- **Child node:** A node directly inside another node. For example, A is a child of BODY in the above example.
- **Descendant node:** A node anywhere inside another node.
- **Parent node:** A node which has another node inside it. For example, BODY is the parent node of A in the above example.
- **Sibling nodes:** Nodes that sit on the same level in the DOM tree. For example, H1 and P are siblings in the above example.
- **Text node:** A node containing a text string.

- diagramme UML des interfaces définies par DOM (Node, Element, NodeList et Document), qui sont implémentées en JavaScript dans les navigateurs.



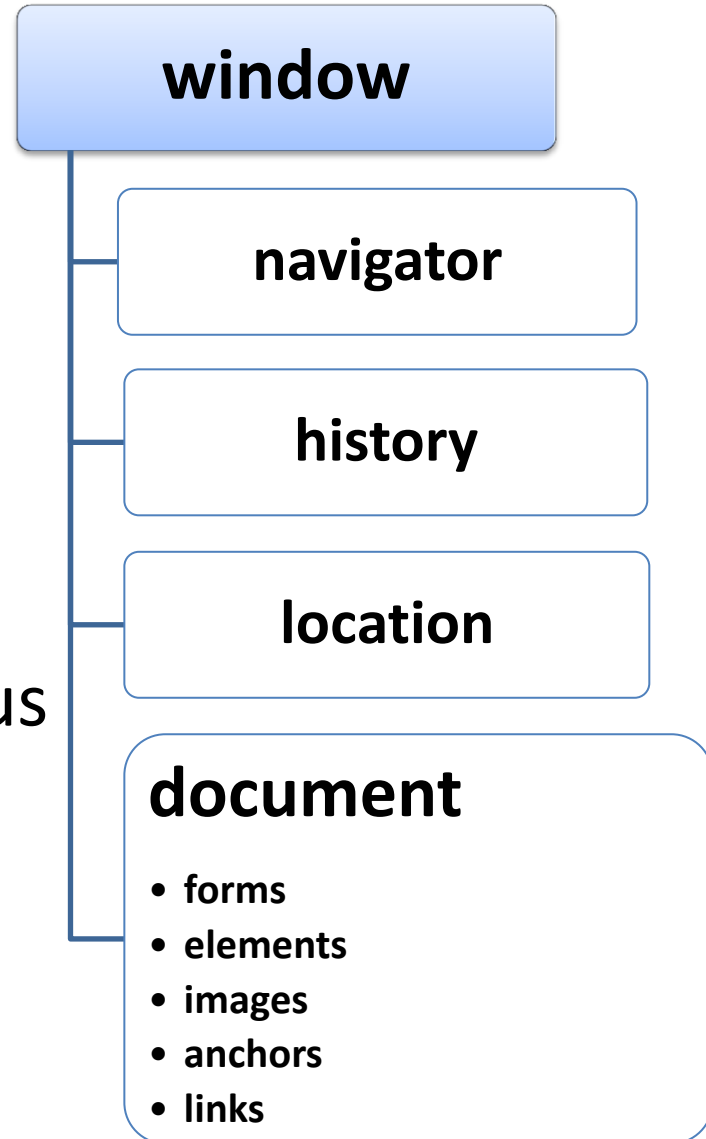
DOM

- Le HTML DOM définit le document de HTML comme **collection d'objet**. Les objets ont des méthodes et des propriétés.
- L'objet de **document** est le parent de tous autres objets dans des documents de HTML, qui est le fils de l'objet **window**.
- **La manipulation dynamiquement** des objets:
 - de l'interface (navigator, window, ...)
 - du document (body, form, table, ...)

(objets créés automatiquement par le navigateur lors du chargement de mon document)
- Comment ? DOM propose de représenter un document sous la forme d'un **arbre**. Toutes les balises HTML sont donc des **nœuds de l'arbre** et les feuilles sont soit des balises sans contenu, soit le texte de la page HTML.

Notion d'objet prédéfini dans JS

- **Objets prédéfinis du navigateur gérés par JS :**
 - **window** object
 - **navigator** object
 - **history** object
 - **location** object Etc.
- Dans une page Web, l'objet le plus élevé dans la hiérarchie est la fenêtre du navigateur : **window**.
 - **alert()**, **confirm()** et **prompt()** sont des méthodes de **window**



Méthode 1: Accès direct a un élément

- L'objet **document**, sous objet de **window**, est le parent de tous autres objets dans un documents de HTML. Il faut passer par lui pour accéder à **tous** les autres.
- L'objet de **document.body** représente l'élément de <body> des documents de HTML.
- **Méthode 1 : accès direct**
- Comme il y a 1 seul objet body associé à 1 objet document, avec **document.body**

Méthode 1: Accès direct a un élément- exemple

- Exemple de HTML DOM : comment la couleur de fond d'un document HTML peut être changée en jaune quand l'utilisateur clique là-dessus.?

```
<html><head>
<script>
function ChangeColor() {
document.write("The background color is: ");
document.write(document.body.bgColor);
document.body.bgColor="yellow"; }
</script>
</head>
<body bgcolor="red" onclick="ChangeColor()">
Click on this document!
</body></html>
```

Méthode 2

- Comment accéder à un élément paragraphe, table, etc. d'un document HTML sachant qu'il peut y avoir 0 à +sieurs occurrences de cet élément dans un même document HTML?
- Méthode d'accès directe n'est plus valable!

Méthode 2: id pour se positionner dans une page HTML

- La 2ème méthode pour retrouver un élément dans notre page:
 - Avec les **id** et les **class** que vous avez utilisé pour mettre en forme votre page, les id sur une page étaient uniques pour permettre l'identification (id) à coup sûr d'un élément dans une page.
- Les ids + DOM permet de « se promener » sur notre page.
- **Méthode 2 : se positionner dans un document grâce à un id :**

var elem= document.getElementById("mon_id");

- ATTENTION à l'écriture : get**E**lement**B**y**I**d !!!!!
- Nous créons une variable que l'on place à l'endroit voulu grâce à la méthode **getElementById** qui prend en argument l'**id**.

id pour se positionner dans une page HTML

Manipulation de CSS : exemple

```
<html>
  <head>
    <script>
      function Test ()
      { document.getElementById("paragraphe").style.color = "blue"
      ; }
    </script>
  </head>
  <body>
    <p id="paragraphe" style="color:red">un texte</p>
    <a href="#" onclick="Test()">Test</a>
  </body></html>
```

- Explication : L'exemple contient un paragraphe avec le nom **id paragraphe** et un lien que si on le clique, il appelle la fonction Test(). Cette fonction change la propriété CSS color du paragraphe, de telle sorte que le paragraphe perde sa couleur rouge et devienne bleu.

Recommendation

```
document.getElementById("paragraphe").style.color = "blue"
```

- Cette écriture n'est pas vraiment recommandée !
- Essayez de laisser la modification de style avec du CSS !!

Correction :

- Au lieu de :

```
document.getElementById("paragraphe").style.color = "blue" ;
```

- Optez pour :

```
document.getElementById("paragraphe").className = "enBleu";
```

- Et déclarer au niveau de vos style la classe « enBleu » :

```
.enBleu {color:blue;}
```

Manipulation de style

Syntaxe : `document.getElementById("id").style.property="value"`

Règles générales :

- On peut changer de cette façon n'importe quelle propriété CSS, d'un nœud ayant "id" pour identifiant
- Le nom de la propriété en JavaScript est identique au nom CSS, sauf que les traits d'unions sont remplacés par une majuscule sur la lettre suivante.
- Les valeurs des propriétés en JavaScript sont identiques aux valeurs CSS (mais doivent être mises entre guillemets).

Exemple :

```
<html><head> <script>
function setFont(){
document.getElementById("p1").style.fontFamily="arial,sans-serif";
</script></head><body><p id="p1">This is an example
paragraph.</p><form>

```

Méthode 3: name pour se positionner dans une page HTML

- De même il est possible de se positionner dans un document grâce à un **name** prédéfini:

Syntaxe

: `window.document.getElementsByName("nom")`

• **Attention : `getElementsByName` (avec un "s")**
Retourne un tableau (Array) d'objets HTML ayant
un nom défini comme valeur de l'attribut **name**

```
<form name="f1">
<input name="fruit" type="checkbox" value="B" />Banane<br />
<input name="fruit" type="checkbox" value="K" />Kiwi<br />
<input name="fruit" type="checkbox" value="R" />Raisin <br /><br />
<input type="button"
onclick="var x=document.getElementsByName('fruit'); alert(x.length);"
value="how many elements named 'fruit'?"/>
</form>
</body>
</html>
```

Methode 4: utilisation de `document.getElementsByTagName()`

- **Méthode 4** : permet de **récupérer toutes les balises identiques** à partir du document, ou d'un nœud inférieur (dans l'arbre).
 - Vous pouvez utiliser cette méthode pour compter le nombre de balises qu'il y a sur votre page, par exemple, ou pour naviguer dans votre document si vous connaissez bien sa structure.
 - Exemple :

```
var titreList = document.getElementsByTagName("h1");  
  
titreList.length; // nombre de h1 dans le document  
titreList.item(2); // accéder à la 3ème balise h1 rencontrée  
titreList[2]; // accéder à la 3ème balise h1 rencontrée
```
 - Dans tous les cas, quand le nœud voulu n'existe pas, la méthode utilisée renvoie **undefined**. Vous avez donc désormais la possibilité de vous promener à volonté dans votre fichier HTML.

Utilisation de document.getElementsByTagName()

Manipulation de CSS : exemple

- Exemple:

```
document.getElementsByTagName("img");
```

- Cet exemple va retourner un tableau contenant tous les images de la page. Il est ainsi possible d'accéder à chacun des éléments. Si on veut accéder au second de la page, on tapera: `document.getElementsByTagName('img')[1];`

- Propriétés :

- On retrouve presque toujours les mêmes attributs qu'en HTML.
- Exemple: Modifier l'adresse d'une image et ses dimensions :

```
monImage =  
document.getElementsByTagName("img")[1];  
monImage.src = "banniere.jpg";  
monImage.width = "800";  
monImage.height = "200";
```

Methode 6 : querySelector

- Returns the first [Element](#) within the document that matches the specified group of selectors.

```
element = document.querySelector(selectors);  
/*selectors is a string containing one or more CSS selectors separated by commas.*/
```

Example :

```
var e1 = document.querySelector(".myclass");  
//returns the first element in the document with the class "myclass"
```

Methode 7 : querySelectorAll

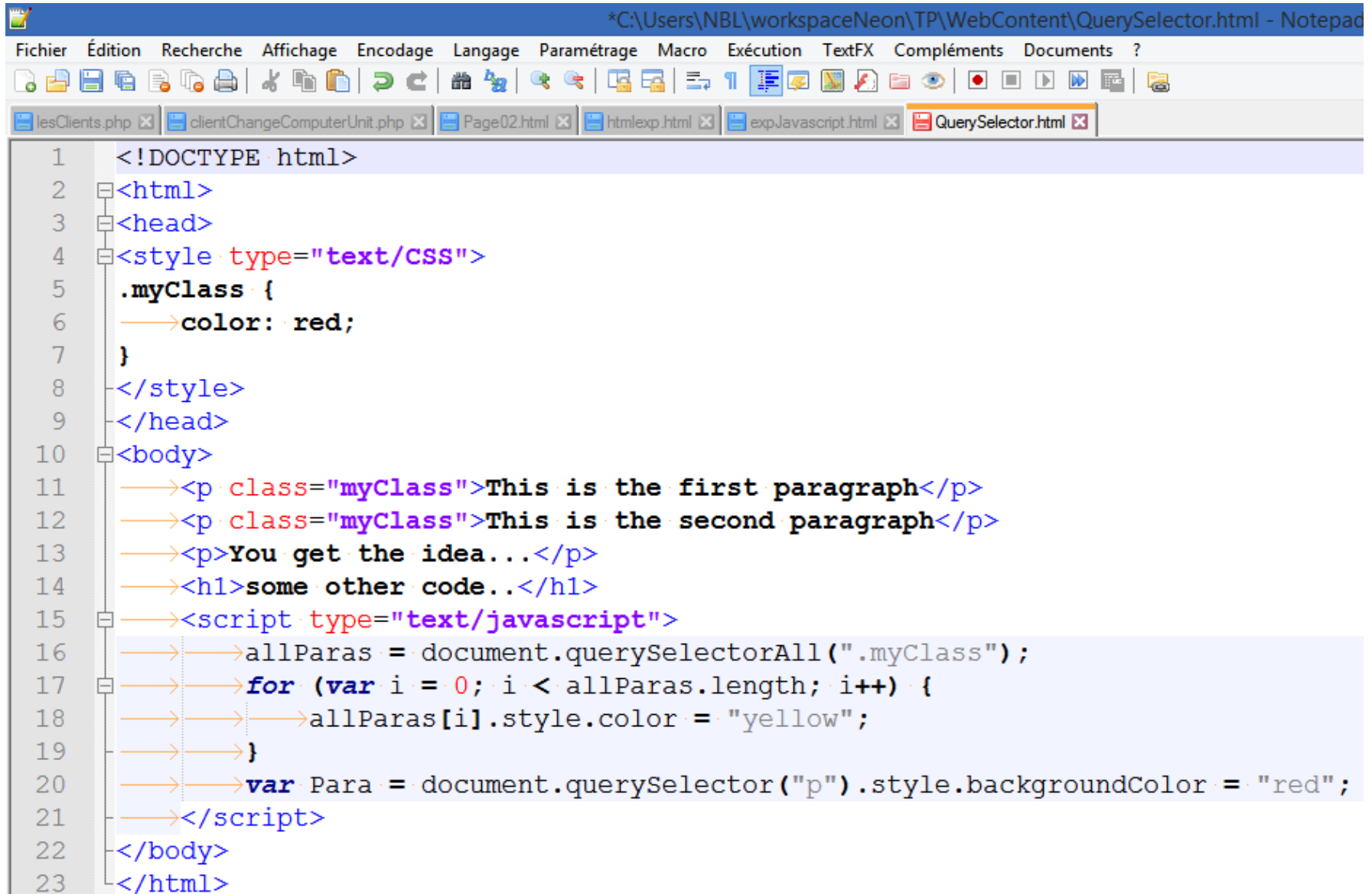
- Returns an Array of [Elements](#) within the document that matches the specified group of selectors.

```
element = document.querySelector(selectors);  
/*selectors is a string containing one or more CSS selectors separated by commas.*/
```

Example :

```
var el = document.querySelector(".myclass");  
//returns an array of elements in the document with the //class "myclass"
```

Example



```
*C:\Users\NBL\workspaceNeon\TP\WebContent\QuerySelector.html - Notepad++
Fichier  Édition  Recherche  Affichage  Encodage  Langage  Paramétrage  Macro  Exécution  TextFX  Compléments  Documents  ?

lesClients.php x clientChangeComputerUnit.php x Page02.html x htmlxp.html x expJavaScript.html x QuerySelector.html x

1  <!DOCTYPE html>
2  <html>
3  <head>
4  <style type="text/CSS">
5      .myClass {
6          color: red;
7      }
8  </style>
9  </head>
10 <body>
11     <p class="myClass">This is the first paragraph</p>
12     <p class="myClass">This is the second paragraph</p>
13     <p>You get the idea...</p>
14     <h1>some other code..</h1>
15     <script type="text/javascript">
16         allParas = document.querySelectorAll(".myClass");
17         for (var i = 0; i < allParas.length; i++) {
18             allParas[i].style.color = "yellow";
19         }
20         var Para = document.querySelector("p").style.backgroundColor = "red";
21     </script>
22 </body>
23 </html>
```


Méthode 8 : utilisation des méthodes prédéfinies des objets

- **Méthode 8 :** se positionner dans un document grâce à des méthodes qui vont nous permettre de parcourir les nœuds d'un document :

```
var node = document.getElementById("mon_id");  
var parent = node.parentNode;  
//place la variable parent sur le noeud parent de node  
  
var childList = node.childNodes;  
//récupère tous les enfants de node dans un tableau childNodesList  
  
var child1 = node.firstChild;  
//récupère le premier enfant de node  
  
var childx = node.lastChild;  
//récupère le dernier enfant de node  
  
var frerePrec = node.previousSibling;  
//récupère le frère précédent de node  
var frereSuiv = node.nextSibling;  
//récupère le frère suivant
```

Ajouter du code XHTML dans une page (1/2)

- Pour ajouter des balises proprement, il faut tout construire brique par brique. Le principe est assez simple :
 - D'abord, on doit (1) récupérer toutes les briques ; ensuite (2) on fait des assemblages ; et enfin (3) on accroche le tout là où on veut.

(1) pour récupérer nos briques:

```
var titre = document.createElement("h1");  
//Ici on crée une balise de type h1.  
//Si on voyait ce qu'on a créé, on aurait : <h1></h1>
```

```
titre.setAttribute("class","Titre_rouge");  
//Ici on ajoute un attribut à notre balise
```

```
var lien = document.createElement("a");  
lien.setAttribute("href","toto.html");  
var texte = document.createTextNode("Titraïlle");  
//Ici on a créé un texte
```

Ajouter du code XHTML dans une page (2/ 2)

(2) Pour assembler nos briques, et surtout mettre le tout dans notre page HTML :

```
lien.appendChild(texte);  
    //La, on accroche en dernier enfant (donc premier,  
    //puisque c'est le seul) le text node créé juste au-  
    dessus.  
    // On aura donc un lien de ce type :  
    //<lien url="toto.html">Titraillle</lien>  
  
titre.appendChild(lien);  
    //là, on accroche notre lien à la balise h1
```

- On a donc un groupe de balises virtuelles sous cette forme prêtes à être mises dans le XHTML :

```
<h1><a href="toto.html">Titraillle</a></h1>
```

Méthode relationnelle d'ajout de balise

- Il existe 2 autres méthodes prédéfinies pour accrocher une balise à une autre, qui sont :

```
parent.insertBefore(child, referenceChild);  
//Ici, on accroche notre balise child à la  
balise parent, et dans l'ordre, elle se retrouve  
juste avant la balise referenceChild
```

```
parent.replaceChild(newChild, oldChild);  
//Ici, on remplace la balise enfant de parent  
oldChild par la balise newChild
```

Manipulation de style

```
node.className="nouvelle_classe"  
//pour changer le nom de la classe CSS à laquelle appartient  
le noeud.  
node.style.borderStyle="valeur"  
//pour changer le style de bordure d'un noeud.
```

- **Règles générales :**
 - On peut changer de cette façon n'importe quelle propriété CSS, d'un noeud.
 - Le nom de la propriété en JavaScript est identique au nom CSS, sauf que les traits d'unions sont remplacés par une majuscule sur la lettre suivante.
 - Les valeurs des propriétés en JavaScript sont identiques aux valeurs CSS (mais doivent être mis entre guillemets).

exercice

- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side web APIs/Manipulating documents](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Manipulating_documents)
[\(shopping list\)](#)
- [https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building blocks/Image gallery](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Building_blocks/Image_gallery)

DOM: objet form

- L'objet **form** est un sous-objet de **document**. Cet objet représente les formulaires de la page HTML.
- Propriétés
 - **name** : nom (unique) du formulaire
 - **method** : méthode de soumission (0=GET, 1=POST)
 - **action** : action déclenchée par la validation du formulaire
 - **target** : fenêtre de destination de la réponse (si elle existe)
 - **elements[]** : tableau des éléments du formulaires
 - **length** : nombre d'éléments du formulaire

DOM: objet form

- Méthodes
 - `submit()` : soumet le formulaire
 - `reset()` : réinitialise le formulaire
- Événements
 - `onSubmit(method)` : action à réaliser lorsque le formulaire est soumis
 - `onReset(method)` : action à réaliser lorsque le formulaire est réinitialisé

Méthodes et événements des éléments du Formulaire

- **Méthodes spécifiques aux formulaires**
 - **focus()** : donne le focus à cet élément (pour une zone de texte, place le curseur à l'intérieur)
 - **blur()** : enlève le focus de cet élément (en quelque sorte le contraire de **focus**)
 - **Click()** : simule un clic de souris sur cet élément
 - **select()** : sélectionne ("surligne") le texte de ce champ
- **Évènements spécifiques aux formulaires**
 - **onFocus** : lorsque l'élément reçoit le focus (pour une zone de texte, quand on place le curseur à l'intérieur)
 - **onBlur** : lorsque l'élément perd le focus (en quelque sorte le contraire de **onFocus**)
 - **onChange** : lorsque la valeur / l'état de l'élément change (quand on coche la case, qu'on modifie le texte, etc.)
 - **onSelect** : lorsqu'on sélectionne (quand on "surligne") le texte de ce champ

DOM: Exemples

- Exemple de formulaire :

```
...<body>
  <form name="general">
    <input type="text" name="champ1"
    value="default">
  </form>
</body>...
```

- Accès à l'objet correspondant au formulaire précédent: il y a trois possibilités :
`document.forms["general"]`
`document.forms[0]`
`document.general`

DOM: Exemples

- Accès aux éléments du formulaire

- 3 possibilités :

- ```
document.forms["general"].elements["champ1"]
```

- ```
document.forms["general"].elements[0]
```

- ```
document.forms["general"].champ1
```

- Accès aux propriétés d'un élément

- ```
document.forms["general"].elements["champ1"].value
```

Text Boxes (zones de texte)

- Une zone de texte permet une saisie de l'utilisateur (et pourrait aussi être utilisé pour l'affichage)
- Le code JavaScript peut accéder au contenu

dans l'exemple ci-dessous comme :

```
...<body>
<form id="BoxForm" name="BoxForm">
<p> Enter your name here
document.BoxForm.userName.value
<input type="text" name="userName" id="userName" size="12"
value="" />
<br />
<input type="button" value="Click Me"
onclick="alert('Thanks, ' +
document.forms['BoxForm'].userName.value +
36 ' , I needed that.');" /></p>
</form>
```

Read/Write Text Boxes

- De même, on peut changer le contenu d'une zone par affectation
- Note: le contenu est du texte brut, sans formatage HTML

```
<body>
```

- Les contenus sont accessibles comme une

```
<form id="BoxForm" name="BoxForm">  
<p> Enter a number here
```

```
<input type="text" size="12" name="number" id="number"
```

```
value="2" />  
parseFloat ou parseInt pour convertir en nombre :
```

```
<br /><br />
```

```
<input type="button" value="Double"
```

```
  onclick="document.forms['BoxForm'].number.value=
```

```
parseFloat(document.forms['BoxForm'].number.value)
```

```
* 2;" />
```

```
</p>
```

```

<html>
  <head>
    <title> Fun with Text Boxes </title>
    <script>
      function FahrToCelsius(tempInFahr)
      // tempInFahr is a number (Fahrenheit)
      // Returns temperature in Celsius
      {
        return (5/9)*(tempInFahr - 32);
      }
    </script>
  </head>

```

```

<body>
  <form id="BoxForm" id="BoxForm">
    <p> Temperature in Fahrenheit:
    <input type="text" name="Fahr" size=
      onchange="document.forms['BoxForm']

```

```

FahrToCelsius(parseFloat(document.forms['BoxForm'].Fahr.value));"/>

```

```

    &nbsp; <tt>----</tt> &nbsp;
    <input type="text" name="Celsius" size="10" value=""
      onfocus="blur();" />

```

```

    in Celsius </p>
  </form> </body></html>

```

- **onchange** : Événement qui se déclenche lorsque le contrôle est modifié. Par exemple, lorsqu'on change le contenu d'un objet d'un formulaire et qu'on quitte cet objet .
- **Onfocus** : Événement qui se déclenche lorsque l'élément reçoit le focus (devient actif) soit par action de l'outil de pointage (souris), soit par la navigation tabulée (touches du clavier).
- **blur()** : La méthode javascript **blur()** d'un objet HTML qui permet de supprimer le focus clavier d'une balise HTML .

Utilité: ne pas permettre la modification du
38
champs par une saisie.

Text Box Validation

- Qu'arrive-t-il si jamais l'utilisateur entre un texte et pas un nombre dans la zone de texte Fahr de l'exemple précédent ?
- Solution: ajouter des instructions de validation
 - Initialiser la zone de texte avec une valeur valide ou le vide
 - Lorsque l'évt **onchange** est activé, vérifier que la nouvelle valeur est valide (sinon, reset)

Text Box Validation : exemple

Verify.js

```
function VerifyNum(textBox) {  
    // Assumes: textBox is a text box  
    // Returns: true if textBox contains a number, else  
    // false + alert  
    var boxValue = parseInt(textBox.value);  
    if ( isNaN(boxValue) ) {  
        // isNaN fonction predefinie de JS  
        //Retourne true si boxValue n'est pas un nombre  
  
        alert("You must enter a number value!");  
        textBox.value = "";  
        return false;  
    }  
    return true;  
}
```


Validation Example

```
<html><head><title> Fun with Text Boxes </title>
<script type="text/javascript" src="verify.js">
</script>
<script type="text/javascript">
    function FahrToCelsius(tempInFahr)
    {
        return (5/9)*(tempInFahr - 32);
    }
</script>
</head>
<body>
<form id="BoxForm" name="BoxForm">
    <p> Temperature in Fahrenheit:
    <input type="text" name="Fahr" size="10" value="0"
        onchange="if (VerifyNum(this)) { // "this" refers to current
element
                document.forms['BoxForm'].Celsius.value =
                    FahrToCelsius(parseFloat(this.value));
                }" />
    &nbsp; <tt>----&gt;</tt> &nbsp;
    <input type="text" name="Celsius" size="10" value=""
onfocus="blur();" />
```

Text Areas : rappel

- Dans un formulaire, **TEXT** est limite a une seule ligne input/output
- **TEXTAREA** est similaire en fonctionnalité a TEXT, mais peut spécifier n'importe quel nombre de ligne ou de colonne.

```
<textarea name="TextAreaName" rows="NumRows" cols="NumCols">  
Initialisation Text  
</textarea>
```

- *Note:* contrairement a TEXT, **TEXTAREA** a une balise fermante.
 - L'initialisation de TEXTAREA est mise entre la paire de balise.
- Exactement comme pour l'élément TEXT, le texte de **TEXTAREA** ne peut être formaté en utilisant HTML.

TEXTAREA Example

```
<html>
<head>
  <title> Fun with Textareas </title>
  <script src="verify.js"> </script>
  <script>
    function Table(low, high, power){
      // Results: displays table of numbers between low &
      high, raised to power
      var message = "i: i^" + power + "\n-----
\n";
      for (var i = low; i <= high; i++)
      {
        message = message + i + ": " +
Math.pow(i, power) + "\n";
      }
      document.forms['AreaForm'].Output.value =
message;
    }
  </script>
</head>
```

```

<body><form id="AreaForm">
  <div style="text-align: center;">
    <p> Show the numbers from <input type="text" name="lowRange"
size="4"
      value="1"  onchange="VerifyNum(this);" />
                                to <input type="text"
name="highRange" size="4"
      value="10" onchange="VerifyNum(this);" />
      raised to the power of <input type="text" name="power"
size=3
      value=2    onchange="VerifyNum(this);" />
    <br/><br/>
    <input type="button" value="Generate Table"
onclick="Table(parseInt(document.forms['AreaForm'].lowRange.va
lue), parseInt(document.forms['AreaForm'].highRange.value),
parseInt(document.forms['AreaForm'].power.value));"
/><br/><br/>
    <textarea name="Output" rows="20"
cols="15">initialisation</textarea>
  </p></div></form></body></html>

```

Check buttons (cases a cocher)

```
<html><head>
<title> Check Boxes </title>
<script>
  function processCB() {
    var boxes =
document.forms['BoxForm'].cb.length;
    var s="";
    for (var i = 0; i < boxes; i++)
    {
      if (document.forms['BoxForm'].cb[i].checked) {
+ " " + document.forms['BoxForm'].cb[i].value
      }
    }
    if (s == "")
    { s = "nothing"; }
    alert("You selected " + s);
  }</script> </head>
```

Check buttons (cases a cocher) la suite

```
<body>
  <form id="BoxForm">
<p>Which of these things is unavoidable
  in life (select one or more)?<br/><br/>
<input type="checkbox" name="cb"
  value="Death" />Death<br/>
<input type="checkbox" name="cb"
  value="Taxes" />Taxes<br/>
<input type="checkbox" name="cb"
  value="Robbie" />Robbie Williams<br/>
<br/> <input type="button" value="Done"
  onclick="processCB()" />
</p>
</form></body></html>
```

Check buttons (using a slightly different method)

```
<html><head> <title> Using checkboxes </title>
<script>
    function processCB()
    { var s="";
      var cb=document.forms['BoxForm'].elements['cb'];
      for (var i = 0; i < cb.length; i++)
          { if (cb[i].checked)
              { s = s + cb[i].value + " "; }
            }
      if (s == "")
          { s = "nothing"; }
      alert("You selected " + s); }
</script>
</head>
<body>
    <form id="BoxForm">
        <p> Which of these things is unavoidable in life (select one or more)?<br/><br/>
        &nbsp;<input type="checkbox" name="cb" value="Death" />Death<br/> &nbsp;  
        <input type="checkbox" name="cb" value="Taxes" />Taxes<br/> &nbsp;  
        <input type="checkbox" name="cb" value="failure" />failure<br/> <br/>
        <input type="button" value="Done" onclick="processCB()" /> </p>
    </form> </body> </html>
```

DOM: objet Navigator

- L'objet **navigator** est un sous-objet de **window**. Cet objet donne des informations sur le navigateur utilisé
- Propriétés
 - **appName** : nom de code interne du navigateur
 - **appVersion** : nom réel du navigateur
 - **language** : version du navigateur
 - **platform** : langage du navigateur (fr, en) sous Navigator 4
 - **plugins** : système d'exploitation (MacPPC)ilisé
 - **plugins[]** : tableau des plugins installés chez le client
 - ...

Notion d'objet prédéfini dans JS

- Cet objet **window** contient entre autres l'objet **document** qui lui même contient tous les objets contenus dans la page Web (paragraphes, formulaires, etc...).
- Si une page Web contient plusieurs cadres (frames), l'un objet **window** contiendra un tableau d'objet **frame**.

L'objet **navigator** contient des informations

```
<html><body>
<script type="text/javascript">
document.write("Name: " + navigator.appName) ;
//affichera Name: nom de votre navigateur exécutant ce script
</script>
</body></html>
```

objet Document

- L'objet **document** est un sous-objet de **window**. Cet objet représente la page HTML affichée dans le navigateur.
- Propriétés
 - **forms[]** : tableau des formulaires de la page
 - **forms.length** : nombre de formulaire(s) de la page
 - **links[]** : tableau des liens de la page
 - **links.length** : nombre de lien(s) de la page
 - **anchors[]** : tableau des ancres internes ()
 - **anchors.length** : nombre de d'ancre(s) interne(s)
 - **images[]** : tableaux des images
- *Remarque : les tableaux contiennent les éléments dans l'ordre de leur apparition dans le code HTML!!!!*

objet document

- Propriétés
 - `title` :titre du document
 - `location` :URL du document
 - `lastModified` :date de dernière modification
 - `referrer` :URL de la page d'où arrive l'utilisateur
 - `bgColor` :couleur de fond
 - `fgColor` :couleur du texte
 - `linkColor`, `vlinkColor`, `alinkColor` :couleurs utilisées pour les liens hypertextes

DOM: Exemples

- Exemple : donner le nombre de formulaire

<html>
<body> dans le document

```
<form name="Form1"></form>  
<form name="Form2"></form>  
<form></form>
```

```
<p>Number of forms:  
<script type="text/javascript">  
document.write(document.forms.length) ;  
//affichera number of forms:3  
</script></p>  
</body>  
</html>
```

DOM: objet document

- Méthodes
 - `write(string)` : écrit une chaîne dans le document
 - `writeln(string)` : idem + caractère de fin de ligne
 - `clear()` : efface le document
 - `close()` : ferme le document

document Object

- Internet Explorer, Firefox, Opera, etc. permettent l'accès à votre document HTML en utilisant

`document.write(...)`

Affiche du text sur le page,

`document.URL`

Donne l'adresse URL de votre page HTML,

`document.lastModified`

Donne la date et l'heure de modification du code HTML de la page en cours,

```
<html>
<head>
  <title>Documentation page /title>
</head>

<body>
  <table width="100%">
    <tr>
      <td><i>
        <script>
          document.write(document.URL);
        </script>
      </i></td>
      <td style="text-align: right;"><i>
        <script>
          document.write(document.lastModified);
        </script>
      </i></td>
    </tr>
  </table>
</body>
</html>
```

navigator Object

- navigator.appName : le nom du navigateur utilisé,
- navigator.appVersion : la version du navigateur,

```
<!-- MSIE.css -->
```

```
a {text-decoration:none;
    font-size:larger;
    color:red;
    font-family:Arial}
a:hover {color:blue}
```

```
<!-- Netscape.css -->
```

```
a {font-family:Arial;
    color:white;
    background-color:red}
```

```
<html>
<head>
  <title>Dynamic Style Page</title>

  <script type=
    if (navigator.appName == "Netscape") {
      document.write('<link rel=stylesheet ' +
        'type="text/css" href="Netscape.css">');
    }
    else {
      document.write('<link rel=stylesheet ' +
        'type="text/css" href="MSIE.css">');
    }
  </script>
</head>

<body>
Here is some text with a
<a href="javascript:alert('GO AWAY')">link</a>.
</body>
</html>
```

objet Window

- L'objet **window** représente la fenêtre de votre navigateur. Ça va être un objet très utilisé, car il possède de nombreux sous-objets.
- Propriétés
 - **self** : fenêtre courante
 - **opener** : la fenêtre (si elle existe) qui a ouvert la fenêtre courante
 - **top** :fenêtre principale (qui a crée toutes les fenêtres)
 - **status** : message dans la barre de statut
 - **defaultstatus** : message par défaut de la barre de statut
 - **name** : nom de la fenêtre
 - ...

objet Window

- Méthodes
 - `alert(string)` :ouvre une boîte de dialogue avec le message passé en paramètre
 - `Confirm(string)` :ouvre une boîte de dialogue avec les boutons OK et cancel
 - `prompt(string)` :affiche une fenêtre de saisie
 - `resizeBy(l,h)` :pour rétrécir ou agrandir la fenêtre
 - `resizeTo(l,h)` :largeur et hauteur de la fenêtre à agrandir ou réduire
 - `scroll(int x, int y)` :positionnement aux coordonnées (x,y)
 - `open(URL, string name, string options)` :ouvre une nouvelle fenêtre contenant le document identifié par l'URL
 - `close()` :ferme la fenêtre
 - ...

Window : exemple

- Les boîtes d'alerte sont très bien pour l'affichage des messages court mais ne sont pas bien adapté pour l'affichage de beaucoup de texte, ou de texte nécessitant une mis en forme
- Elles sont pas intégrées intégré dans la page, oblige l'utilisateur à fermer explicitement la boîte de dialogue.
- QUESTION: Peut-on utiliser à la place **document.write**?
- NON : on risque d'écraser la page en cours, y compris les éléments .
- Solution : Ouvrir une nouvelle fenêtre du navigateur et y écrire .

Window : autres méthodes

Ouvrir une nouvelle fenêtre du navigateur et y écrire :

```
<script>
var OutputWindow = window.open();
// open a window and assign a name to that object

OutputWindow.document.open();
// open that window for writing

OutputWindow.document.write("un texte");
// write text to that window as before
OutputWindow.document.close();
// close the window
</script>
```

Window Example 2

```
<html>
<head>
  <title> Fun with Buttons </title>
  <script>
    function Help()
    // Results: displays a help message in a separate window
    {
      var OutputWindow = window.open();
      OutputWindow.document.open();

      OutputWindow.document.write("This might be a context-" +
        "sensitive help message, depending on the " +
        "application and state of the page.");

    }
  </script>
</head>
<body>
  <form id="ButtonForm">
    <p> <input type="button" value="Click for Help"
      onclick="Help();" /> </p>
  </form>
</body>
</html>
```

Window Example Refined

```
<html>
<head>
  <title> Fun with Buttons </title>
  <script>
    function Help()
    // Results: displays a help message in a separate window
    {
      var OutputWindow =
        window.open("", "", "status=0,menubar=0,height=200,width=200");
      OutputWindow.document.open();

      OutputWindow.document.write("This might be a context-" +
        "sensitive help message, depending on the " +
        "application and state of the page.");

    }
  </script>
</head>
<body>
  <form id="ButtonForm">
    <p> <input type="button" value="Click for Help"
      onclick="Help();" />  </p>
  </form>
</body></html>
```

Les arguments de `window.open`: 1st arg spécifie HREF, 2nd arg spécifie internal name, et 3rd arg spécifie window properties (e.g., size)

Manipulation de window : exemple

```
<html><head>
  <script type="text/javascript">
    function OpenWindow() {
      Info = open("fichier.html", "secondefenetre") ;}
  </script></head>
  <body onload="OpenWindow()">
    <p><a href="" onclick="Info.close()">Fermer la
fenetre</a></p>
  </body></html>
```

- Explication : L'exemple ouvre à la lecture du **fichier une deuxième fenêtre du nom de Info**.
- Dans le fichier est défini un lien. Si l'utilisateur clique sur le lien, la deuxième fenêtre est fermée.

Manipulation objet document :

Examples

Le nombre de lien hypertexte interne dans la page:

```
<html>
<body>

<a name="html">HTML Tutorial</a><br />
<a name="css">CSS Tutorial</a><br />
<a name="xml">XML Tutorial</a><br />
<a>SQL Tutorial</a>

<p>Number of named anchors:
<script type="text/javascript">
document.write(document.anchors.length);
</script>
</p>

</body>
</html>
```

Your Result:

HTML Tutorial
CSS Tutorial
XML Tutorial
SQL Tutorial

Number of named anchors: 3

Le nombre d'image dans la page:

```
<html>
<body>

<br />

<br /><br />

<script type="text/javascript">
document.write("This document contains: " + document.images.length
+ " images.");
</script>

</body>
</html>
```

Your Result:



This document contains: 2 images.

JavaScript & Timeouts

- **setTimeout** : peut être utilisée pour définir une exécution différée d'un script.

```
setTimeout (JavaScriptCodeToBeExecuted,  
MillisecondsUntilExecution)
```

- Exemple : un lien vers la nouvelles page quand une page change d'emplacement
- Exemple :

```
<input type="button" value="click me"  
onclick="setTimeout ('window.alert (\ 'Hello! \ ' )  
' , 2000) " />
```

Une boite de message qui contient Hello sera afficher après 2 secondes

Exemple (une redirection/va charger la page newhome.html après 3sec)

```
<html>
  <head>
<script>
  function Move()
    // Results: sets the current page contents to be
newhome.html
    {
      self.location.href = "newhome.html";
    }
</script>
</head>
<body onload="setTimeout('Move()', 3000);">
  <p> This page has moved to <a
href="newhome.html">newhome.html</a>.  </p>
</body>
</html>
```

Another Timeout Example

```
<html> <head>
<script>
    function timeSince()
    // Assumes: document.forms['CountForm'].countdown exists in the page
    // Results: every second, recursively writes current countdown in the
box
    {
        // CODE FOR DETERMINING NUMBER OF DAYS, HOURS, MINUTES, AND SECONDS
        // UNTIL GRADUATION (see the file for this code!!!)

        document.forms['CountForm'].countdown.value=
            days + " days, " + hours + " hours, " +
            minutes + " minutes, and " + secs + " seconds";

        setTimeout("timeSince();", 1000);
    }
</script>
</head>
<body onload="timeSince();" >
    <form id="CountForm">
        <div style="text-align: center;">
            <p> Countdown to Graduation <br />
            <textarea name="countdown" id="countdown" rows="4" cols="15"
                style="font-family: Courier;" onfocus="blur();"></textarea>
        </p>
    </div></form> </body></html>
```

Exercice 1

- soit le document html suivant :
- `<!doctype html>`
- `<html>`
- `<head>`
- `<title>styles</title>`
- `<meta charset="utf-8">`
- `</head>`
- `<body>`
- `<p id="parag1">Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.`

Exercice 1 Suite

- Ecrire la fonction `changer_style` qui permet de styler le paragraphe au clic du bouton par :
 - une couleur blanche.
 - un background noir.
 - une bordure noire pointillée de 1px.
 - un retrait de 5px.
- Définir les propriétés précédentes dans une classe "active" et modifier la fonction `changer_style` de telle façon qu'elle ajoute la classe "active" au paragraphe.

Exercice 2

Le but de cet exercice est de développer le slideshow de la capture d'écran suivante :

- Un lien sur le lien **MOVE NEXT** affiche l'image suivante dans la galerie d'images en appelant la fonction **MoveNext**.
- De même un lien sur le lien **MOVE PREVIOUS** affiche l'image précédente dans la galerie d'images en appelant la fonction **MovePrevious**.
- Télécharger des images de votre choix P1.jpg, P2.jpg , P3.jpg, P4.jpg.
- Les images seront affichées dans le slideshow avec une taille de 100px⁶⁹ 100px. En double

Exercice 3

Le but de cet exercice est de créer le menu déroulant suivant avec une liste de sites web

- Lorsqu'on sélectionne **W3G** et on clique sur **GO** le site web de W3C school s'ouvre.
- De même pour les autres
- Indication : utiliser la propriété **location** de l'objet **window** et **selectedIndex** de la zone

Selected du formulaire

Exercice 4

Construisez un petit formulaire dans une page QCM.html qui contiendra certaines questions d'un QCM à réponse unique et qui contiendra certaines informations

- Nom (20 caractères)
- CIN (20 caractères)
- Classe bouton radio (A et B)
- Adresse email
- Licence liste déroulante (IAG, GL, RT)
- Validation du questionnaire QCM
- Lorsque l'utilisateur remplit correctement une zone le curseur passe à la zone qui suit.
- Ajouter une fonction qui vérifie que tous les champs sont remplis et que l'email contient @et un point(.)
- Si les conditions ne sont pas vérifiées alerter l'utilisateur pour l'aviser du problème.