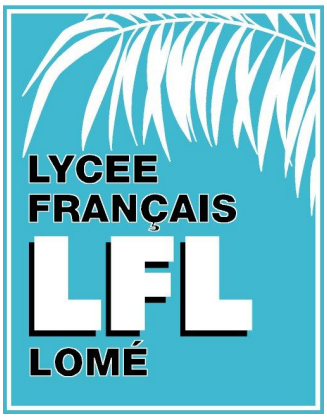
Le langage

Javascript

Partie 3



Sommaire :

Maintenant que nous avons vu le noyau du langage Javascript, nous allons maintenant nous intéresser à l'application de ce langage dans le domaine qui nous intéresse le plus : son intégration dans les pages internet.

Comme il a déjà été dit auparavant, il existe des différences entre le langage de **JavaScript**dans les différents navigateurs *(le site http://www.quirksmode.org/ référence beaucoup de différences d'interprêtations de commandes entre les navigateurs).*
Voici les concepts les plus importants de ces langages de programmation.

I. La version navigateur de JavaScript :

Dans les deux premiers livrets de cette formation, nous avons vu les propriétés du noyau **JavaScript** définit par la norme ECMA-262 : cette norme définit la syntaxe, les objets "*de base*" du langage **JavaScript**...

Ce noyau est utilisé comme base des langages suivants :

- **JavaScript** pour les navigateurs Webs ;
- **JavaScript** pour les fichiers pdf d'Acrobat Reader ;
- **actionScript** qui est le langage de programmation des animations Flash.

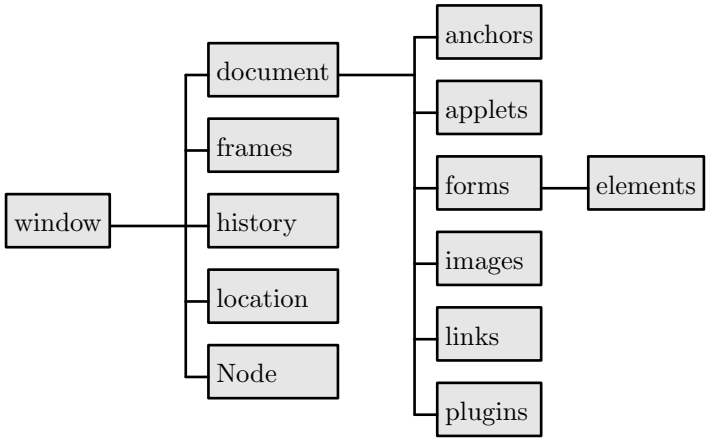
Evidemment, ce ne sont pas les mêmes langages ; les objets manipulés sont différents d'un langage à l'autre : des objets HTML pour les navigateurs Web, contrôler des séquences d'animations pour Flash, les éléments d'un fichier pdf dans le cas de **JavaScript** pour Acrobat.

Cette partie de la formation s'intéresse à l'étude des objets que **JavaScript** va manipuler, contrôler, créer dans un navigateur Web.

Dans la programmation **JavaScript**, on se contente de manipuler des objets. Ainsi, pour se familiariser à **JavaScript** pour les navigateurs, il faudra apprendre les nouveaux objets propre aux pages Web.
On a déjà vu que chaque objet possède des caractéristiques *(appelées propriétés)* et des fonctions permettant de le manipuler *(appelées méthodes)* : **JavaScript** est un langage de programmation orientée objet *(POO)* même si dans la pratique seul l'aspect objet est utilisée *(les classes ne le sont que très peu).*

Dans la norme ECMA-262, il existe un objet contenant tous les autres objets : l'objet `Global`. Pour **JavaScript**, l'objet `Global` prendra le nom de `window`.

Voici un schéma mettant en évidence, la hierarchie des objets **JavaScript** *(on commence par l'objet global* `window`*)*



Ce schéma ne présente pas l'ensemble des objets **JavaScript**.

Voici quelques précisions sur les objets précentés dans le diagramme précédent :

- `frames` : cet objet était utilisé lorsque les pages Web contenées différents cadres. Cette technique est en train d'être abandonnée.
- `history` : cet objet permet de gérer l'historique de navigation du client *(connaître la page précédente,...)*
- `location` : cet objet permet d'obtenir et de modifier l'URL de la page courante *(permet ainsi de rediriger la page courante)*
- `Node` : cet objet est plus difficile à comprendre, il sert de modèle à tous les objets **JavaScript** représentant un objet HTML. Bien qu'il ne soit pas utilisé directement, il est à la base de la spécification DOM *(voir dans un chapitre ultérieur)*
- `document` : c'est l'objet central de la programmation **JavaScript**. Il représente l'ensemble de l'affichage du navigateur : la plupart des manipulations passera par cet objet pour affecter les éléments HTML qu'il contient. Voici quelques uns de ses sous-objets :
  - `anchors` est un tableau contenant toutes les ancres *(lien d'une page sur une partie d'elle-même)* présentes dans le document.
  - `applets` permet d'accéder à tous les applets Java définis dans le document.
  - `forms` contient tous les formulaires présent dans la page courante. Il permet à accéder aux éléments d'un formulaire : boutton, case à cocher...et ainsi de modifier l'exécution d'un formulaire.
  - `images` est un tableau contenant toutes les images de la page courante.
  - `links` est un tableau contenant tous les liens *(lien hypertexte)* de la page courante.
  - `plugins` est un tableau référençant tous les plugins installés dans le navigateur utilisé.

Seul le noyau de **JavaScript** fait parti d'une norme, d'une standardisation *(standart EMCA-262)*, ainsi il existe des différences entre les navigateurs sur les objets présents et leurs caractéristiques : ces différences ont pour origine le développement du langage HTML et **JavaScript** par les deux concurrents Netscape et Microsoft.

Actuellement, des différences subsistent, obligeant le programmeur à de nombreuses péripéties dans l'écriture du code. Mais chaque constructeur actuel de navigateurs voulant apporter ses nouveautés technologique, il est à parier que ces différences ne s'estomperont pas dans l'avenir.

Les collections, comme les tableaux, permettent de contenir d'autres objets mais il ne possède que la propriété et la méthode suivantes :

- la propriété `length` : renvoyant le nombre d'objet contenu dans la collection.
- la méthode `item(i)` : qui renverra l'objet d'index `i`.
- la méthode `namedItem(str)` désigne, s'il existe, l'élément de la collection dont l'attribut *`id`* vaut `str`, dans le cas contraire essaye avec l'attribut *`name`*.

  Pour une collection ayant pour identifiant `collect`, il est possible d'appeler ses éléments en utilisant la syntaxe *(identique à celle des tableaux)* :

  ```
  collect[i]
  ```

  Pour l'objet de rang `i`.

II. L'insertion du code :

A. Le principe :

Il existe deux manières d'insérer du code **JavaScript** dans une page Web :

- le code est inscrit directement dans le code HTML ;
- le code est tapé dans un fichier extérieur ; on fait alors référence à ce fichier à l'intérieur du code HTML pour que ce fichier soit chargé.

Dans les deux cas, on utilise l'élément HTML script. Voici comment :

```
1 Pour insérer le code directement dans la page :
2   <script language="javascript">
3     ...Mettre le code ici
4   </script>
5
6 Pour appeler un fichier externe de script :
7   <script type="text/javascript" src="urlFichier">
8   </script>
```

Le second exemple montre comment appeler le fichier externe indiqué par l'URL `urlFichier` à l'aide de l'attribut *`src`*.

En cas de présence de l'attribut `src`, le code se trouvant entre les balises <`script`> et </`script`> n'est pas exécuté : seul le code se trouvant dans le fichier externe sera pris en compte.

On peut insérer notre code **JavaScript** en deux endroits dans une page HTML modifiant ainsi le moment d'exécution du code :

- A l'intérieur de l'élément head : le code sera chargé et exécuté en même temps que l'entête du document. A ce moment là, l'élément body et tous ces descendants n'existent pas encore.
  Généralement, on place à cet endroit l'ensemble des définitions et fonctions utilisées dans le reste du document.

- A l'intérieur de l'élément body : le navigateur recevant au fur et à mesure le code HTML, le code **JavaScript** sera exécuté au moment de sa reception.

Tel qu'il est présenté, on a l'impression que le code **JavaScript** s'exécute une seule fois. En fait, il se charge une seule fois, mais peut par la suite s'exécuter à l'aide des événements : mouvement de souris, frappe du clavier, soumission d'un formulaire...

III. L'objet `window` :

L'objet `window` est l'objet global de **JavaScript** : il contient tous les autres objets.

Il n'est pas, dans la plupart des cas, nécessaire de le citer explicitement :

`window.moveBy(3,10)` `moveBy(3,10)`

Par contre

Propriété :

string `window.defaultStatus`

Cette propriété contient le texte de la barre d'état. Elle peut être modifiée.

Attention FireFox cache, par défaut, la barre d'état : le client doit modifier les droits de **JavaScript**.

number `window.innerHeight`

Cette propriété renvoie, en nombre de pixels, la hauteur d'affichage de la page courante. Cette propriété est en lecture seule : aucune modification possible.
Cette propriété n'existe pas sous Internet Explorer.

number `window.innerWidth`

Cette propriété renvoie, en nombre de pixels, la largeur d'affichage de la page courante.
Cette propriété est en lecture seule : aucune modification possible.
Cette propriété n'existe pas sous Internet Explorer.

number `window.pageXOffset`

Cette propriété renvoie, en nombre de pixels, la position de la barre de défilement horizontale et elle est en lecture seule : aucune modification possible.
Cette propriété n'existe pas sous Internet Explorer.

number `window.pageYOffset`

Cette propriété renvoie, en nombre de pixels, la position de la barre de défilement verticale et elle est en lecture seule : aucune modification possible.
Cette propriété n'existe pas sous Internet Explorer.

Dans le cas d'une page Web constituée de cadres *(frames)*, les propriétés suivantes permettent à **JavaScript** de manipuler le jeu de cadres présent :

`frames`, `length`, `parent`, `self`, `top`

Les cadres étant de moins en moins utilisées, ces propriétés ne seront pas développées.

Méthode :

void `window.moveBy(`*int* `x,` *int* `y)`

Cette méthode déplace la fenêtre de sa position actuelle de `x` pixels horizontalement et de `y` pixels verticalement.

void `window.moveTo(`*int* `x,` *int* `y)`

Cette méthode place le coin supérieur gauche de la fenêtre à l'emplacement de coordonnées (`x` ; `y`)

void `window.resizeBy(`*int* `x,` *int* `y)`

Cette méthode modifie la largeur de `x` pixels *(attention* `x` *peut être négatif)* et la hauteur de `y` pixels.

void `window.resizeTo(`*int* `x,` *int* `y)`

Cette méthode fixe la largeur et la hauteur de la fenêtre respectivement à `x` et `y` pixels.

void `window.scrollBy(`*int* `x,` *int* `y)`

Dans le cas où le contenu de la page est supérieur à la surface d'affichage, le navigateur fera apparaître des barres de défilement.
Cette méthode permet, dans ce cas, de modifier l'emplacement de la barre horizontale de défilement de `x` pixels et la barre verticale de `y` pixels.

void `window.scrollTo(`*int* `x, `*int* `y)`

Dans le cas où la fenêtre affiche des barres de défilement, cette méthode va placer ces barres à la position `x` horizontalement et `y` verticalement.

void `window.alert(`*string* `msg)`

Cette méthode affiche une boîte de dialogue où apparait le message `msg` afin d'informer le client : cette boîte ne contient que le bouton "Ok".

boolean `window.confirm(`*string* `msg)`

Cette méthode affiche une boîte à dialogue avec le message `msg`. Le client dispose des deux boutons "Ok" et "Annuler" pour indiquer son choix.
Cette méthode renvoit `true` en cas d'appui sur "OK", `false` dans le cas contraire.

string `window.prompt(`*string* `msg, `*string* `valDefaut)`

Cette méthode affiche une boîte de dialogue avec pour message `msg` et un champ de texte est laissé à la disposition de l'utilisateur qui peut ainsi passer une chaîne de caractère au script.
Par défaut, le champ texte contient la valeur `valDefault`.
En cas d'appui sur la touche "Ok", la méthode renvoit la valeur du champ texte ; en cas d'appui sur la touche "Annuler", elle retoune la valeur `null`.

object `window.open(`*string* `url, `*string* `nom, `*string* `options)`

Le premier argument est obligatoire, les deux derniers sont optionnels.
Cette méthode permet d'ouvrir une autre fenêtre du navigateur se dirigeant vers l'adresse `url`.

L'argument `nom` permet de repérer la nouvelle fenêtre ouverte : si la méthode `open()` est exécutée plusieurs fois avec la même valeur `nom` alors les pages ouvertes s'afficheront toutes successivement dans la même fenêtre.
Au contraire, choisissez la valeur `"_blank"` pour que tous vos scripts ouvrent indépendamment une nouvelle fenêtre.

Le dernier argument définit les caractéristiques de la fenêtre qui s'ouvrira : cette chaîne de caractères spécifie des couples `propriété=valeur` de propriétés séparés entre eux par des virgules. Voici la liste des propriétés utilisables :

- `left` et `top` prennent des entiers pour valeur et définissent la distance en pixel du bord supérieur gauche de l'écran relativement à celui de la nouvelle fenêtre.
- `width` et `height` définissent respectivement la largeur et la hauteur de l'affichage

de la nouvelle fenêtre. Leurs valeurs sont des entiers représentant la taille en pixels.

`outerWidth` et `outerHeight` définissent la largeur et la hauteur de la fenêtre, comprenant barre d'outils, de navigation...

Ces deux propriétés ne sont pas reconnues par Internet Explorer.

Les propriétés suivantes prennent chacune la valeur `true` et `false` équivalent par conversion de type à 1 ou 0 :

- `fullscreen` affiche la nouvelle fenêtre en mode plein écran *(utilisable qu'avec Internet Explorer)*.
- `location` affiche ou non la barre d'adresse dans la nouvelle fenêtre.
- `menubar` affiche ou non la barre des menus dans la nouvelle fenêtre
- `toolbar` affiche ou non la barre d'outils.
- `status` affiche ou non la barre d'état dans la nouvelle fenêtre.
- `resizable` autorise ou non le client à redimensionner la fenêtre.
- `scrollbars` affiche ou non les barres de défilement.
- `dialog` affiche ou non les icones "agrandir/réduire" de la barre des titres *(seulement avec FireFox)*.

void `window.close()`

Cette méthode permet de fermer les fenêtres ouvertes par le même script qui exécute cette méthode. FireFox refusera de fermer toute fenêtre ouverte par le client, Internet Explorer ouvrira une boîte à dialogue pour demander au client de confirmer la fermeture de la fenêtre.

void `window.print()`

Cette méthode appelle l'impression du document ; au client de refuser ou de l'accepter.

number `window.setInterval(`*string* `code,`*int* `duree)`

`code` contient soit du code **JavaScript**, soit le nom d'une fonction *(pas inscrit dans une chaîne)*.

Après exécution de cette méthode, `code` sera exécuter à intervalles réguliers de `duree` millisecondes.

Cette méthode renvoie un entier identifiant ce mécanisme de répétition ; on utilisera conjointement `clearInterval()` pour arrêter cette exécution cyclique de `code`.

D'autres arguments sont possibles mais FireFox et Internet Explorer n'en font pas la même interprétation.

void `window.clearInterval(`*int* `id)`

Cette méthode arrête l'exécution cyclique lancée par la méthode `setInterval()` et identifié par l'entier `id`.

number `window.setTimeout(`*string* `code,`*int* `duree)`

`code` contient soit du code **JavaScript**, soit le nom d'une fonction.
Cette méthode exécute `code` avec un différé de `duree` millisecondes.
Entre le temps d'appel à cette méthode et l'exécution de `code`, il est possible d'utiliser la méthode `clearTimeout` pour annuler la future exécution de `code`.

void `window.clearTimeout(`*int* `id)`

Cette méthode annule l'exécution à retardement enclenché par `setTimeout()` et caractérisé par par `id`.

void `window.stop()`

Cette méthode arrête le chargement de la page. L'effet est différent de FireFox à Internet Explorer.

Il a été dit que `window` représente l'objet global de **JavaScript** et que sa nomination explicite n'était pas obligé. Il reste un cas pour lequel, il est nécessaire de le citer : pour tester l'existence et l'affectation d'une variable :

```
1 <script language="javascript">
2 x="window.a "+(!window.a?"n'existe pas":"existe")+"<br>↩
    ";
3 document.write(x);
4
5 x="a "+(!a?"n'existe pas":"existe")+"<br>";
6 document.write(x);     /*Ceci ne s'affichera pas*/
7 </script>
```

A l'exécution, ce script n'affiche pas la seconde partie, car l'exécution de `!a` affiche une erreur, alors que `window.a` renvoie la valeur `undefined` commme `a` est une propriété non-définie de `window`.

IV. L'objet `document` :

L'objet `document` est l'un des objets les plus importants en **JavaScript**. Il représente tout le

contenu de votre page, alors que `window` représente la fenêtre du navigateur.

Propriété :

HTMLElement `document.body`

Cette propriété renvoie l'objet **JavaScript** représentant l'élément body de la page courante. Pour que cette propriété soit définie, il faut que les balises <`body`> et </`body`> aient été explicitement inscrite dans le code.

number `document.width` et `document.height`

Ces deux propriétés donnent, en lecture seule, les dimensions d'affichages du navigateur *(pas de la fenêtre)*.
Ne fonctionne que pour FireFox.

string `document.title`

Cette propriété contient le titre de la fenêtre du navigateur.
Elle peut être modifiée dans un script.

string `document.location`

Cette méthode contient l'adresse de la page courante.
Par modification dans un script, on redirige la page vers une autre adresse.

collection `document.images`

Cette propriété référence toutes les images présentes dans la page dans une collection d'objet.

collection `document.forms`

Cette propriété référence tous les formulaires de la page courante dans une collection d'objet.

collection `document.styleSheets`

Cette propriété renvoie une collection d'objets contenant toutes les feuilles de styles déclarées dans le document.

collection `document.links`

Cette propriété renvoie dans une collection tous les liens présents dans le document.

collection `document.anchors`

Cette propriété renvoie une collection contenant toutes les ancres présentes dans le document.

string `document.cookie`

Cette propriété a pour valeur une chaîne de caractère associée au cookie, s'il existe, rattaché à la page courante. Voici la forme possible d'une telle chaîne :

```
"path:...;domain=...;max-age:...;expires=...;secure"
```

- `path` représente le sous-dossier où sera stocké le cookie.
- `domain` représente le domaine utilisant ce cookie.
- `max-age` exprime en secondes la durée de vie du cookie.
- `expires` donne la date limite d'existence du cookie *(obsolète)*.
- `secure` le cookie ne sera utilisé si seulement la connexion du client se fait en mode sécurisé.

En plus des couples nom/valeur données par le créateur du cookie.

Un travail sur les chaînes de caractères est nécessaire pour récupérer l'ensemble des valeurs.

node `document.childNode`

Cette propriété contient tous les noeuds de l'élément *(voir spécification DOM)*

Les propriétés `alinkColor`, `bgColor`, `fgColor`, `linkColor`, `vlinkColor` ne sont actuellement plus conseillée à l'utilisation. En effet, il est actuellement préférable de modifier directement les caractéristiques de l'élément body à l'aide des feuilles de styles :

```
window.document.body.style.backgroundColor="#AAAAAA";
```

Il est à indiquer les propriétés suivantes qui n'ont pas été développées ici *(pas toutes présentes dans Internet Explorer)* :

`embeds` ; `domain` ; `URL` ; `applets` ; `characterSet` ; `contentType`

Méthode :

void `document.write(`*string* `str)`

Cette méthode affiche la chaîne `str` dans la page courante au moment de son exécution. `str` peut contenir des éléments HTML.

Si le document n'est plus en cours de chargement, l'utilisation de cette méthode aura pour effet de vider préalablement l'ensemble du document : on se retrouve alors avec

une page blanche.

Rappelez-vous qu'en **JavaScript**, une chaîne de caractère ne peut être contenu que sur une ligne ; utilisez `\n\r` pour l'affichage d'un retour à la ligne dans un fichier texte et `<br>` dans une page Web.

void `document.writeln(`*string* `str)`

Indentique à `document.write()` mais fait suivre l'insertion de `str` par un retour à la ligne.

HTMLElement `document.getElementById(`*string* `ident)`

Cette méthode renvoie l'élément du document *(sensé être unique)* ayant l'identifiant `ident`.
Dans le cas où aucun élément ne possède l'attribut *`id`* affecté de la valeur `ident`, cette méthode renvoie `null`.

nodeList `document.getElementsByName(`*string* `nom)`

Cette méthode renvoie la liste de tous les éléments ayant l'attribut *`name`* fixé à `nom`.

nodeList `document.getElementsByTagName(`*str* `nom)`

Cette méthode renvoie la liste de tous les éléments HTML présent dans le document définis par la balise `<nom>`.

Les deux méthodes suivantes ne sont ciées qu'à titre d'information car leur utilité n'est pas très importante.

void `document.open()`

Cette méthode indique au navigateur que le script va envoyer des informations aux flux de sortie pour l'affichage.
Elle prépare l'utilisation de la méthode `document.write()` *(cette dernière lance automatiquement la méthode* `open()` *si cela n'a pas été fait préalablement).*

void `document.close()`

Cette méthode indique au navigateur que le script arrête d'envoyer des informations aux flux de sortie.

Il est à noter également les propriétés et méthodes de `document` qui n'ont pas été développées ici :

- Les méthodes suivantes permettent de créer au sens du modèle *(voir plus loin)* des éléments

afin de les adjoindre à un noeud du document :

`createAttribute()`, `createElement()`, `createComment()`, `createTextNode()`

`createAttribute()`

Cet aspect là ne sera pas discuter dans cette formation.

- Les propriétés suivantes font parties de tous éléments HTML affichables *(ou presque)* et seront discutés plus loin :

  `onmouseover`, `onclick`, `onfocus`...

V. L'objet `forms` :

L'objet `form` est une collection contenant les références à tous les formulaires HTML présents dans la page courante.

Ainsi, `forms[i]` permet d'accéder au formulaire d'index `i` de la page et ainsi à tous ses contrôles.

Pour comprendre ce paragraphe, il est nécessaire d'avoir une bonne connaissance des formulaires HTML.

Propriété :

number `document.forms.length`

En tant que collection, l'objet `forms` possède cette propriété indiquant le nombre d'éléments contenus dans cette collection : plus précisemment, cette propriété contient le nombre de formulaire présent dans le document courant.

collection `document.forms[i].elements`

Cette propriété retourne la collection de tous les contrôles composant le formulaire d'index `i`.

int `document.forms[i].length`

Cette propriété contient le nombre de contrôle présent dans le formulaire d'index `i`

string `document.forms[i].name`

Cette propriété contient le nom du formulaire d'index `i`. Sa valeur peut être modifiée.

string `document.forms[i].acceptCharset`

Cette propriété contient une chaîne de caractère représentant la liste des encodages de caractères acceptés *(dans les contrôles de champs de texte)* par le serveur : sa valeur

peut être `ISO-8859-2`, `UTF-8`...
Sa valeur peut être modifiée.

string `document.forms[i].action`

Cette propriété contient l'action *(l'URL de reception du formulaire)* à effectuer lors de la soumission du formulaire `i`. Sa valeur peut être modifiée.

string `document.forms[i].encoding`

Cette propriété contient la manière dont les données du formulaire `i`. Sa valeur par défaut est `application/x-www-form-urlencoded`.
La valeur `multipart/form-data` permet d'insérer des fichiers lors de la soumission du formulaire.

string `document.forms[i].method`

Cette propriété contient la méthode HTTP utilisée pour la soumission du formulaire : c'est à dire `"get"` ou `"post"`. Sa valeur peut être modifiée.

string `document.forms[i].target`

Cette méthode contient la cible du formulaire d'index `i` : le choix de la fenêtre pour l'exécution du formulaire. Sa valeur peut être modifiée.

Méthode :

void `document.forms[i].submit()`

Cette méthode a pour effet de soumettre le formulaire d'index `i` : cet méthode a le même effet que lorsque le client appuye sur le bouton de soumission du formulaire.

void `document.forms[i].reset()`

Cette méthode rétablit les valeurs par défaut de l'ensemble des contrôles du formulaire d'index `i`.

Chaque formulaire possède également deux gestionnaires d'événement `onsubmit` et `onreset` permettant au script de prendre la main lorsque, par exemple, l'utilisateur souhaite soumettre ou réinitialiser les valeurs du formulaire.
Le gestionnaire d'événement `onsubmit` peut permettre de vérifier l'intégrité des informations du formulaire avant leurs transmissions.
Ces deux méthodes seront développées lors de la présentation de la gestion des événements par **JavaScript**.

VI. L'objet `forms[i].elements` :

Après avoir sélectionné un formulaire, on accède à la propriété `elements` de la manière suivante :

```
window.document.forms[i].elements
```

Cet objet de type HTMLCollection permet de balayer l'ensemble des contrôles du formulaires.

L'objet `elements` étant une collection d'objets, voici les propriétés et méthodes qu'il supporte :

- `document.forms[i].elements.length` contient le nombre d'éléments présent dans la collection.
- `document.forms[i].elements.item(j)` permet de sélectionner l'élément de rang `j` de cette collection.
- `document.forms[i].elements.namedItem(str)` sélectionne l'élément dont l'attribut *`id`* ou *`name`* a pour valeur `str` parmi la collection.

Un élément de `elements` peut être de nature différentes et les propriétés et méthodes associés dépendront donc de l'élément sélectionné.
Passons en revue, les différents types d'éléments dans `elements` :

- De type *HTMLSelectElement* :

string `elements[i].type`

Cette propriété représente le type du contrôle. La valeur sera `"select-one"` ou `"select-multiple"` dans le cas où ce contrôle possède l'attribut `multiple`.

number `elements[i].selectedIndex`

Renvoie le numéro d'index du choix actuellement sélectionné.
Le premier choix a un index de 0. Cette propriété renvoie -1 si aucun choix n'est actuellement sélectionné.

string `elements[i].value`

Renvoie la valeur du choix actuellement sélectionné.

number `elements[i].length`

Renvoie le nombre d'option de la liste déroulante représentée par `elements[i]`.

HTMLFormElement `elements[i].form`

Cette propriété contient la référence au formulaire contenant le contrôle

`elements[i]`.

collection `elements[i].options`

Cette propriété contient la collection de toutes les options de choix de la liste déroulante `elements[i]`.

boolean `elements[i].disabled`

Si cette propriété a pour valeur `true`, alors le contrôle est désactivé .

boolean `elements[i].multiple`

Si sa valeur est `true`, la liste déroulante est à choix multiple : le client pourra choisir plusieurs options parmi celles proposées *(équivalent à utiliser l'attribut* `multiple` *dans la balise HTML).*

string `elements[i].name`

Cette propriété contient le nom de la liste déroulante.

number `elements[i].size`

Renvoie le nombre d'options visibles de la liste déroulante.

void `elements[i].add(`*HTMLOptionElement* `nvlleOpt,`
HTMLOptionElement `avantOpt)`

Cette méthode permet d'insérer un nouveau choix dans la liste déroulante `elements[i]`.

Au préalable, on créera une nouvelle option à l'aide du constructeur suivant :

```
new Option(texte,valeur)
```

et on repérera l'emplacement où insérer la nouvelle option en repérant l'option de la liste se trouvant avant cet emplacement.

Dans le cas où `avantOpt` a la valeur `null` *(rendant ainsi cet argument facultatif)*, la nouvelle option sera placée en fin de la liste déroulante.

void `elements[i].remove(`*int* `ind)`

Cette méthode va retirer de la liste `elements[i]` le choix d'index `ind`

void `elements[i].focus()`

Cette méthode donne le focus à l'élément `elements[i]`.

Vous pouvez alors modifier la sélection de la liste à l'aide du clavier.

void `elements[i].blur()`

Enlève le focus à l'élément `elements[i]`.

- De type *HTMLOptGrouptElement* :

Les groupes d'options permettent de regrouper des choix de la liste pour une meilleure présentation de celle-ci.

boolean `elements[i].disabled`

Cette propriété permet de désactiver l'ensemble des contrôles du groupes.

string `elements[i].label`

Cette propriété contient le label du groupe.

- De type *HTMLOptionElement* :

HTMLFormElement `elements[i].option[j].form`

Renvoie la référence au formulaire contenant cette option de choix.

string `elements[i].option[j].text`

Renvoie le texte qui représentera le choix d'index `j` de la liste déroulante.
Cette propriété est en lecture seule, elle ne peut être modifié.

number `elements[i].option[j].index`

Renvoi l'index du choix d'index `j`.
Cette propriété ne peut être modifié.

boolean `elements[i].option[j].disabled`

Désactive le choix d'index `j` de la liste déroulante.

boolean `elements[i].option[j].defaultSelected`

Renvoie `true` si ce choix est le choix sélectionné par défaut *(au chargement du formulaire)*.

boolean `elements[i].option[j].selected`

Renvoie un booléen indiquant si l'option `j` est actuellement sélectionné. Sa modification permet de modifier la sélection du choix courant.

string `elements[i].option[j].value`

Retourne la valeur associée au choix `j`.

- De type *HTMLInputElement* :

  Ces propriétés s'appliquent aux élément HTML input : l'attribut ***type*** de cet élément détermine la nature même du contrôle. Les propriétés et méthodes applicables dépendront de la valeur de l'attribut ***type***.

string `elements[i].name`

Cette propriété contient la valeur de l'attribut ***name*** du contrôle `elements[i]`.

HTMLFormElement `elements[i].form`

Cette propriété est une référence au formulaire contenant l'élément.

string `elements[i].accept`

Pour la valeur *file* de l'attribut ***type***, cette propriété contient les différents types de fichiers *(Mime)* acceptés par l'upload.

boolean `elements[i].disabled`

Lorsque cette propriété a pour valeur `true`, le contrôle est désactivé.

string `elements[i].readOnly`

Pour les éléments de type *text* et *password*, cette propriété permet d'empêcher le client de modifier sa valeur, mais le contrôle n'est pas désactivé.

number `elements[i].tabIndex`

Cette propriété permet de modifier l'ordre de sélection des contrôles de la page par appuis successifs de la touche de tabulation.

string `elements[i].type`

Renvoie la valeur de l'attribut `type` HTML de l'élément input. Un script, par modification de cette propriété, est capable de changer dynamiquement le type d'un contrôle.

number `elements[i].size`

Pour les éléments de type *text* et *password*, cette propriété définit la longueur, en nombre de caractère, du contrôle. Ne pas confondre avec `maxlength`.

Pour les autres éléments, ce nombre représente la taille du contrôle en pixels *(n'affecte pas tous les éléments)*

boolean `elements[i].checked`

Pour les éléments de type *radio* et *checkbox*, cette propriété permet de savoir si le contrôle est coché ou non. Un changement de valeur affecte l'élément HTML en conséquence.

string `elements[i].defaultValue`

Pour les éléments de type *text*, *file* et *password*, cette propriété représente la valeur par défaut *(définie par l'attribut HTML* `value`*)* du contrôle.

boolean `elements[i].defaultChecked`

Pour les éléments de type *checked* et *radio*, cette propriété indique si le contrôle est coché par défaut *(utilisation de l'atribut HTML* `checked`*)*.

number `elements[i].maxLength`

Pour les éléments de type *text* et *password*, cette propriété indique le nombre maximal de caractère saisissables par le client.

string `elements[i].src`

Pour les éléments de type *image*, cette propriété indique l'image utilisée pour représenter le contrôle.

string `elements[i].useMap`

Pour les éléments de type *image*, cette propriété indique l'emplacement de l'image map à utiliser.

string `elements[i].value`

Pour les éléments de type *text*, *file* et *password*, cette propriété représente la valeur courante du champs texte correspondant mais ne change pas la valeur par défaut de ce contrôle en cas de reinitialisation du formulaire.

Pour les éléments de type *button*, *hidden*, *submit*, *reset*, *image*, *checkbox* et *radio*, cette propriété représente la valeur de l'attribut HTML `value`.

void `elements[i].blur()`

Cette méthode enlève le focus au contrôle. Le client ne peut plus modifier le contrôle à l'aide du clavier.

void `elements[i].focus()`

Cette propriété donne le focus au contrôle : le contrôle est sélectionné pour les actions à partir du clavier.

void `elements[i].select()`

Pour les éléments de type *text*, *file*, *password*, cette méthode sélectionne le champs de texte.

void `elements[i].click()`

Pour les éléments de type *button*, *hidden*, *submit*, *reset*, *checkbox* et *radio*, cette méthode simule un clic sur le contrôle.

- De type *HTMLTextAreatElement* :

string `elements[i].defaultValue`

Cette propriété renvoie la valeur par défaut du champ de texte : c'est à dire la valeur de l'attribut HTML *value*.

HTMLFormElement `elements[i].form`

Cette propriété renvoie une référence au formulaire contenant ce champ de texte.

string `elements[i].accessKey`

Cette propriété prend pour valeur un seul caractère, celui-ci permettra d'accéder plus rapidement au contrôle à l'aide du clavier.
Ne marche que pour Internet Explorer.

boolean `elements[i].disabled`

Si cette propriété a pour valeur `true`, le contrôle est désactivé.

string `elements[i].name`

Cette propriété renvoie le nom du champ de texte : la valeur de l'attribut HMTL *name*.

boolean `elements[i].readOnly`

Si cette propriété a pour valeur `true`, le champ de texte est en lecture seule pour le client mais n'est pas désactivé.

number `elements[i].cols`

Cette propriété contient le nombre de colonne composant le champ de texte. Sa modification entraîne celle du contrôle.

number `elements[i].rows`

Cette propriété contient le nombre de lignes composant le champ texte. Sa modification entraîne celle du contrôle.

number `elements[i].tabIndex`

Cette propriété contient le numéro d'ordre par lequel le bouton sera sélectionné par appui successif de la touche de tabulation.

string `elements[i].type`

Cette propriété renvoie la valeur `"textaera"` identifiant ainsi la nature du contrôle.

string `elements[i].value`

Cette propriété contient la valeur actuelle du contenu du champ texte, ne pas confondre avec la valeur par défaut du contrôle définie par l'attribut *`value`*.

void `elements[i].blur()`

Cette méthode entraîne la perte du focus à l'élément.

void `elements[i].focus()`

Cette méthode donne le focus à l'élément.

void `elements[i].select()`

Cette méthode permet de sélectionner l'ensemble du contenu du champ texte.

Il existe également les éléments de types *HTMLLabelElement*, *HTMLFieldSetElement*, *HTMLLegendElement*, *HTMLButtonElement* qui ne seront pas développés dans cette formation, leur utilité étant moindre que les contrôles présentés ci-dessus.

VII. Le Document Object Model :

A. Une introduction aux noeuds :

Le Document Object Model *(DOM)* est une interface de programmation pour les pages Web :

- Il propose une représentation de la structure d'une page Web sous forme d'arbre permet-

tant la mise en évidence de l'imbrication des éléments HTML.

- Il propose un certain nombre de propriétés et méthodes permettant de contrôler et de modifier l'ensemble de la page.

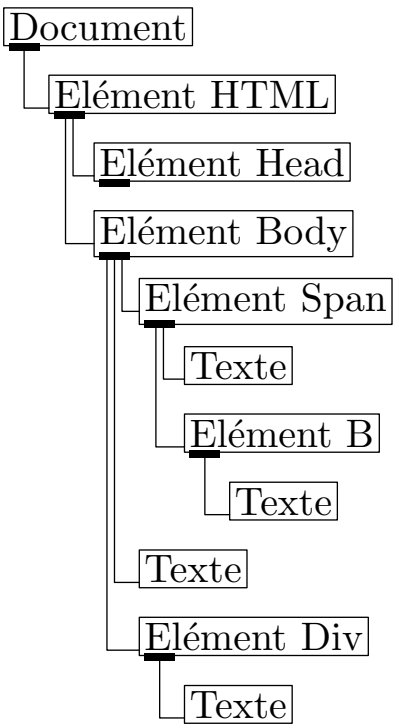
Cette interface est une norme du W3C depuis 1998, elle a connu trois versions successives pour connaître sa forme définitive au printemps 2004. Nous étudierons l'implémentation **JavaScript** de cette interface.

Mais il est bon de préciser qu'il existe une implémentation de cette interface pour presque tous les langages *(Visual Basic, C++, java...)* ; ainsi, connaissant la logique de cette interface et les propriétés/méthodes s'y rapportant, il est alors possible de contrôler une page Web à partir de n'importe quel langage de programmation.

Ainsi, sous la vision DOM, une page HTML peut être vu comme un arbre ordonné dont le premier élément est l'élément `document` qui renferme tous les autres documents.

Voici un exemple de conception de page Web, sous la vision du DOM :

```
1  <html>
2   <head>
3   </head>
4   <body>
5    <span font=5>
6     Bon<b>jour</b>
7    </span>
8    Thomas
9    <div>Quel jour est-on? </div>
10  </body>
11 </html>
```



Le code est représenté par une arbre où sont représentés chaque élément et leurs descendants respectifs : ce modèle met en évidence l'imbrication des éléments HTML.

Chaque élément est le départ d'un noeud *(en anglais "node")*, mis en évidence sur le graphique par un trait noir, représentant les autres éléments qu'il contient.

Le modèle DOM possède plusieurs types de noeud dont voici les trois principaux :

- Les noeuds-éléments : chaque élément HTML est un noeud-éléments possédant ou non des sous-noeuds appelés des noeuds enfants.
- Les noeuds-attributs : ils représentent les attributs HTML affectés à un élément *(ne peut avoir de noeud-enfants)*
- Les noeuds-texte : contenant l'ensemble des caractères du texte.

Le schéma ci-dessus représente les noeuds-éléments de notre page Web : c'est la représentation importante du modèle DOM.

Ainsi, dans l'exemple précédent, l'élément body est un noeud contenant trois noeud-enfants :

- le noeud définit par span ;
- le noeud texte contenant "Thomas"
- le noeud définit par l'élément div.

Le modèle DOM au travers des méthodes et propriétés qu'il dispense permet :

- de parcourir et d'inspecter l'ensemble d'un document ;
- de rajouter, d'effacer des éléments aux noeuds ;
- d'accéder aux éléments HTML et d'en modifier le contenu, les attributs, les définitions de styles...

B. L'accés aux noeuds :

Il existe deux manières d'accéder à un élément HTML en utilisant le modèle DOM :

- Les méthodes `getElementById()`, `getElementsByName()` et `getElementsByTagName()` de l'objet `document` permettent de repérer un ou un ensemble à partir d'informations sur ces éléments : la valeur de l'attribut *`id`*, la valeur de l'attribut *`name`*, le nom de la balise. Ces méthodes font parties de l'élément `document`.
- L'autre manière est de parcourir l'ensemble des noeuds de la page en partant du noeud de départ *(l'élément* `document`*)* et de parcourir ses noeud-enfants et suivant.

Voici les différentes propriétés et méthodes permettant de parcourir l'ensemble des éléments d'une page Web :

`elt` représente un élément de type `Node`.

Propriété :

NodeList `elt.childNodes`

Cette propriété renvoie la liste de tous les noeuds-enfants contenus dans l'élément `elt`.

Le nombre de noeuds-enfants est connu à l'aide de la propriété `length` :

```
elt.childNodes.length
```

On accède au noeud-enfant de rang 2 par l'une des deux commandes suivantes :

```
elt.childNodes[2]   ;  elt.childNodes.item(2)
```

Node `elt.firstChild`

Cette propriété renvoie le premier noeud-enfant contenu dans l'élément `elt`.

Node `elt.lastChild`

Cette propriété renvoie le dernier noeud-enfant de l'élément `elt`.

Node `elt.nextSibling`

Cette propriété renvoie le noeud-frère suivant directement l'élément `elt`.

Node `elt.previousSibling`

Cette propriété renvoie le noeud-frère précédent directement l'élément `elt`.

Node `elt.parentNode`

Cette propriété renvoie le noedu-parent de l'élément `elt`.

Exemple pratique : Etude de l'exemple donné en début de paragraphe.

- `document` est la racine de l'arbre DOM.
  - Il ne contient que l'élément HTML. Ainsi :
    `document.childNodes.length` renvoie 1.
    L'élément HTML est récupérable par :
    `document.firstChild`
  - Pour s'avoir combien d'éléments contient HTML, on fait :
    `document.firstChild.childNodes.length`
- Intéressons-nous maintenant à l'élément body
  - On récupère cet élément à l'aide de l'une des deux instructions suivantes :
    `a=document.getElementsByTagName("body")[0]`
    `a=document.body`
  - Le nombre de noeud-enfants de body s'obtient par :
    `a.childNodes.length`
  - On accèdera à l'élément span par :
    `a.firstChild`
    Le nombre de noeud-enfants de l'élément span est :
    `a.firstChild.childNodes.length` renvoie 2.

C. Propriétés et méthodes des noeuds :

Dans ce paragraphe, seront exposées les propriétés et les méthodes de **JavaScript**des noeuds *(c'est à dire des éléments HTML??)*. Ces propriétés et méthodes permettrons d'agir directe-

ment sur les éléments HTML de la page.

`elt` représentera un élément HTML quelconque vu comme un noeud *(de nature* `Node`*)* :

Propriété :

NodeList `elt.attributes`

Cet attribut représente la collection des attributs définits pour l'élément `elt`. Ainsi :

- `elt.attributes.length` renvoie le nombre d'attributs de `elt`.
- `elt.attributes[0].name` contient le nom du premier attribut et `elt.attributes[0].value` sa valeur.

String `elt.className`

Cette attribut contient le nom de la classe de l'élément s'il est défini : défini par l'attribut *`class`*.

String `elt.id`

Cette attribut contient la valeur de l'attribut HTML *`id`* de l'élément `elt`.

`elt.name`

Seulement applicable aux éléments anchor, applet, form, frame, iframe, image, input, map, meta, object, option, param, select and textarea. dans firefox

String `elt.nodeName`

Cette propriété contient le nom de l'élément `elt`. Cela peut être :

- le nom de la balise dans le cas où `elt` est un élément HTML
- `"#text"` si `elt` est un noeud-texte

Number `elt.nodeType`

Cette propriété contient un nombre représentant le type de `elt` au sens du DOM. On retiendra que les valeurs suivantes *(voir en annexe pour de plus amples informations)* :

- Un noeud élément a la valeur 1 *(Node.ELEMENT_NODE)* ;
- Un noeud attribut a la valeur 2 *(Node.ATTRIBUTE_NODE)* ;
- Un noeud texte a la valeur 3 *(Node.TEXT_NODE)* ;

String `elt.nodeValue`

Pour des noeuds de type texte ou attribut, cette propriété renvoie la valeur de `elt` *(le texte ou la valeur de l'attribut)*.

Object `elt.style`

Cette propriété permet d'accéder aux définitions de style CSS de lélément `elt`.
Cette propriété, très utilisée, permet de modifier dynamiquement le style d'un élément *(elle sera étudiée dans un prochain chapitre)*.

Number `elt.tabIndex`

Cette propriété définit le numéro d'accès à l'élément `elt` lors de frappes successives sur la touche tabulation du clavier.

String `elt.tagName`

Identique à `nodeName`. Il renvoie le nom de la balise définissant l'élément `elt` ; si `elt` n'est pas un élément HTML, la valeur `undefined` est renvoyée.

Les propriétés suivantes ne font pas parties du modèle DOM, mais permettent de manipuler presque tous les éléments HTML :

String `elt.innerHTML`

Cette propriété contient l'ensemble du texte et des balises contenus dans l'élément `elt`.
Cette propriété est très utilisée pour modifier dynamiquement le contenu d'un élément HTML.
Cette propriété ne fait pas partie du DOM ; actuellement, Internet Explorer ne supporte cette propriété que pour des objects de type "*block*" *(ne marche pas les éléments table)*.

Number `elt.offsetHeight`

Cette propriété contient la hauteur de l'élélement `elt`.

Number `elt.offsetWidth`

Cette propriété contient la largeur de l'élément `elt`.

Number `elt.offsetLeft`

cette propriété représente la longueur en pixels séparant le bord gauche de l'élément `elt` du bord gauche de son élément parent.

Number `elt.offsetTop`

cette propriété représente la longueur en pixels séparant le bord supérieur de l'élément `elt` du bord supérieur de son élément parent.

Node `elt.offsetParent`

Cette propriété rentourne une référence à l'élément parent de `elt`.

Les quatres propriétés suivantes sont utiles pour des éléments HTML utilisant les barres de défilement. Elles sont utiles pour des éléments de tailles fixés et utilisant la définition de style "***overflow***:*scroll*"

Number `elt.scrollHeight`

Cette propriété représente la hauteur du contenu qui sera défilé dans l'élément `elt`.

Number `elt.scrollWidth`

Cette propriété représente la largeur du contenu qui sera défilée dans l'élément `elt`.

Number `elt.scrollTop`

Cette propriété représente la position actuelle de la barre de défilement vertical de l'élément.
Elle renvoit 0 si cette barre de défilement n'existe pas.

Number `elt.scrollLeft`

Cette propriété renvoie la position actuelle de la barre de défilement horizontal de l'élément.
Elle renvoit 0 si cette barre de défilement n'existe pas.

Méthode :

String `elt.getAttribute(`*string* `attr)`

Cette méthode renvoie la valeur de l'attribut `attr` de l'élément `elt` dans le cas où celui-ci est défini.
Dans le cas contraire, elle renvoie une chaîne vide ; il préférable d'utiliser d'abord la méthode `elt.hasAttribute()` pour vérifier l'existence de cet attribut.

node `elt.getAttributeNode(`*string* `attr)`

Identique à la méthode `getAttribute`, mais renvoie le résultat sous forme d'un noeud d'attribut.

boolean `elt.hasAttribute(`*string* `attr)`

Cette méthode renvoie `true` dans le cas où l'élément `elt` possède bien l'attribut `attr` défini.
Evidemment la valeur `false` dans le cas contraire.

Boolean `elt.hasAttributes()`

Cette méthode renvoie `true` si l'élément `elt` possède au moins un attribut.

void `elt.normalize()`

Cette méthode normalise l'élément `elt` et toute sa descendance *(au sens de DOM)* : cette méthode efface tous les noeuds-textes vides et elle concaténe tous les noeuds-textes adjacents

void `elt.removeAttribute(`*str* `attr)`

Cette méthode retire l'attribut `attr` de l'élément `elt`.

void `elt.setAttribute(attr,val)`

`attr` et `val` sont des chaînes de caractères.
Cette méthode va fixer la valeur de l'attribut `attr` à la valeur `val`.

nodeList `elt.getElementsByTagName(`*string* `nom)`

Cette méthode retourne la collection des noeuds-enfants de `elt` ayant pour nom de balise `nom`.

On citera au passage les méthodes `elt.setAttributeNode()` et `elt.removeAttributeNode()`, peu utilisées et eqúivalente aux méthodes `elt.setAttribute()` et `elt.removeAttribute()` mais prennant pour argument un noeud d'attribut.

D. Modification dynamique des éléments d'une page :

Les méthodes suivantes permettent de modifier dynamiquement la structure DOM d'une page HTML : on peut ajouter ou supprimer des éléments en modifiant les noeuds-élélements.

node `document.createElement(`*string* `tagName)`

Cette méthode permet de créer un nouvel élément HTML dont le type est `tagName`.
Cette méthode est utilisée pour créer un élément avant de l'utiliser avec les méthodes

présentées à la suite.

node `elt.appendChild(`*node* `noeud)`

Cette méthode place le noeud-élément `noeud` à la fin de la liste des noeuds-enfants de l'élément `elt`.
Si `noeudEnfant` existe déjà alors il sera uniquement déplacé ; la méthode `elt.cloneNode()` peut être utilisée pour copier au préalable un noeud.
Cette méthode renvoie l'élément rajouté.

node `elt.cloneNode(`*boolean* `profondeur)`

Si `profondeur` vaut `true`, cette méthode créé une copie intégrale de `elt` et de toute sa descendance.
Si `profondeur` vaut `false`, seul l'élément et ses attributs seront copiés.

Cette méthode renvoie comme valeur, le noeud copié.

node `elt.insertBefore(`*node* `nvElem,` *node* `emplElem)`

Cette méthode insére l'élément `nvElem` dans l'élément `elt` juste avant `emplElem` : `emplElem` doit être un noeud-enfant de `elt` sous peine de provoquer une erreur.
Cette méthode renverra l'élément inséré *(c'est à dire* `nvElem`*)*.

Si `nvElem` fait déjà parti de l'arbre DOM, il sera alors déplacé.
Si `emplacementElem` vaut `null`, le nouvel élément est inséré à la fin de la liste des noeuds-enfants de `elt`.

node `elt.removeChild(`*node* `enfant)`

Cette méthode retire le noeud-enfant `enfant` de l'élément `elt`.
Cette méthode retourne le noeud retiré.

node `elt.replaceChild(`*node* `nvEnfant,` *node* `enfantRemplace)`

Cette méthode remplace le noeud `enfantRemplace` par le noeud `nvEnfant`.
Cette méthode retournera comme valeur le noeud qui a été retiré de `elt`.

Si `nvEnfant` appartenient à l'arbre DOM, il sera alors déplacé.
Si `enfantRemplace` n'est pas un noeud-enfant de `elt`, une erreur sera générée.

VIII. CSS et feuille de style :

A. Introduction :

Commençons par rappeler qu'une définition de style CSS affectant un élément peut être :

- incluse directement dans l'élément à l'aide de l'attribut HTML *style*.
- définie dans une feuille de style :
  - interne qui est placé dans l'entête du document,
  - externe dans un fichier externe, elle sera invoquée à l'aide de l'élément link.

Pour gérer, les définitions apportées directement à un élément `elt`, on utilise la propriété `elt.style` : en annexe est donné un tableau de référence entre le nom d'une règle de style et le nom de la propriété **JavaScript** correspondante.
L'interface DOM et son implémentation en **JavaScript** nous donne un certain nombre de méthode pour contrôler les feuilles de styles associées au document.

B. La collection des feuilles de styles :

La propriété `document.styleSheets` permet d'accéder à la collection des feuilles de styles associées au document.

Propriété :

number `document.styleSheets.length`

Cette propriété contient le nombre de feuilles de styles liées à la page courante.

Méthode :

styleSheet `document.styleSheets.item(`*int* `num)`

Cette méthode renvoye une référence de la feuille de style de rang `num` de la page courante.
Ceci est est équivalent, comme toute collection, à `document.styleSheets[i]`.

Nous allons voir dans le prochain paragraphe, comment récupérer une feuille de style est la modifier à l'aide de la propriété `item()`.

Il est à noter également qu'on récupère d'un élément `elt` de type link relié à une feuille de style ou de type style la feuille de style qu'il définit à l'aide de la propriété :

```
elt.sheet
```

C. La gestion des feuilles de styles :

Ici `document.styleSheets[i]` représentera un feuille de style valide du document.

Propriété :

string `document.styleSheets[i].type`

Cette propriété contient le type de la feuille de style indiquée. Par défaut, la valeur renvoyée est :

```
"text/css"
```

boolean `document.styleSheets[i].disabled`

Cette propriété indique si la feuille de style indiquée est désactivée ou non.
JavaScript peut modifier cette valeur.

Pour qu'une feuille soit active, il faut que la valeur de l'attribut `media` corresponde à l'agent utilisateur utilisé *(navigateur, imprimante, vidéo-projecteur...)*

node `document.styleSheets[i].ownerNode`

Cette propriété fait référence au noeud-élément définissant la feuille de style.
En HTML, ce noeud est un élément link ou style.

string `document.styleSheets[i].href`

Cette méthode renvoie, dans le cas d'une feuille externe, l'adresse relative du fichier contenant la feuille de style.

Si la feuille de style est interne, la valeur retourée sera `null` *(attention, IE renvoie une chaîne vide).*

string `document.styleSheets[i].title`

Cette propriété donne le titre de la feuille de style, si celui-ci a été déclaré.

mediaList `document.styleSheets[i].media`

Cette propriété indique, si elle a été définie, le média de destination prévu pour cette feuille de style.

D. La gestion des règles de styles dans une feuille :

Dans le modèle DOM, on accède aux règles de styles présentes dans une feuille de styles à l'aide de la propriété `cssRules`. Il en sera ainsi avec FireFox.
Par contre, pour Internet Explorer, la propriété `rules` sera utilisée.

Propriété :

CSSRuleList `document.styleSheets[i].cssRules`

Cette propriété contient la collection des règles CSS définies dans la feuille de style `styleSheets[i]`.

Le nombre de règles définies dans cette feuille est accessible via :

```
document.styleSheets[i].cssRules.length
```

La règle de style de rang `j` est récupérable avec :

```
document.styleSheets[i].cssRules[j]
document.styleSheets[i].cssRules.item(j)
```

Méthode :

number `document.styleSheets[i].insertRule(`*string* `regle,`*int* `j)`

Cette méthode va insérer la règle définie par la chaîne de caractère `regle` *(contenant le sélecteur et la déclaration de style).* Pour rajouter cette règle en fin feuille, `j` doit avoir la valeur :

```
document.styleSheets[i].cssRules.length
```

Cette méthode renvoie l'index de la règle ainsi ajoutée.

void `document.styleSheets[i].deleteRule(`*int* `j)`

Cette méthode efface la règle de rang `j` de la feuille de styles considérée.

A noter que Internet Explorer préfère les méthodes `addRule()` et `removeRule`.

E. Les règles de styles :

En ayant récupéré une règle de style d'une feuille de style, à partir du code suivant :

```
regle=document.styleSheets[i].cssRules.item(j)
```

string `regle.cssText`

Cette propriété retourne la chaîne de caractères représentant le texte complet représentant la règle *(sélecteur et déclaration de la règle).*

Pour une bonne compatibilité entre IE et FireFox, il est préférable d'utiliser :

```
regle.style.cssText
```

mais seul la chaîne de caractères correspondant à la déclaration correspond à cette propriété

number `regle.type`

Cette propriété renvoie le type de règle. Voici les différents types reconnues dans

DOM :

UNKNOWN_RULE : 0	MEDIA_RULE : 4
STYLE_RULE : 1	FONT_FACE_RULE : 5
CHARSET_RULE : 2	PAGE_RULE : 6
IMPORT_RULE : 3	

string `regle.selectorText`

Cette propriété contient le sélecteur associé à la règle *(.maClasse, #monId, element)*.

SSStyleDeclaration `regle.style`

Cette propriété permet de modifier les propriétés de styles définies dans `regle`. Elle sera étudiée en profondeur dans le prochain paragraphe.
C'est la voie à adopter pour une plus grande compatibilité entre FireFox et IE.

F. Les déclarations de styles :

La propriété `style` permet d'atteindre la déclaration de style :

- d'un noeud-élément HTML. Par exemple :

```
document.getElementById("tableau")
```

- d'une règle d'une feuille de style. Par exemple :

```
document.styleSheets[2].cssRules[1]
```

On notera dans le reste de ce paragraphe `decl` une déclaration de style.

En annexe, se trouve un tableau présentant les différentes propriétés de `elt.style` et leur correspondance en CSS. Toutes ces propriétés sont modifiables par **JavaScript**.

Propriété :

Hormis les propriétés applicables à l'objet `elt.style` ayant un lien avec une définition de style *(que vous retrouverez en annexe)*, voici les autres propriétés disponibles :

string `decl.cssText`

Cette propriété contient la chaîne de caractère représentant la déclaration de style `decl`.
Attention, cette chaîne a été traitée par le navigateur et peut connaître des différences relativement à la page de l'auteur *(notamment un passage en lettres majuscules pour IE)*.

number `decl.length`

Cette propriété renvoie le nombre de propriétés de style présentes dans la déclaration `decl`.

Attention à l'interprêtation par les navigateurs de définition multiple :

```
border:3px red solid
```

Internet Explorer n'interprête pas encore cette commande.

Méthode :

Les méthodes suivantes permettant de gérer les déclarations de styles ne sont pas prise en compte avec Internet Explorer *(FireFox les prend en compte)*.

string `decl.getPropertyPriority(`*string* `nom)`

Cette méthode renvoie la priorité de la propriété de style `nom` : c'est à dire `"important"` si elle est définie, une chaîne vide dans le cas contraire.

string `decl.getPropertyValue(`*string* `nom)`

Cette propriété renvoie la valeur de la propriété `nom` définie directement dans la déclaration *(non pas par héritage)*.

Elle renverra unechaîne vide dans le cas où la propriété n'a pas été définie dans la déclaration `decl` de style.

string `decl.item(`*int* `num)`

Cette méthode renvoie le nom de la propriété de rang `num` si elle existe, sinon elle renvoie une chaîe vide.

string `decl.removeProperty(`*string* `nom)`

Cette méthode est utilisée pour supprimer la propriété `nom` de style de la déclaration `decl` de styles.

Cette méthode retourne la valeur de la propriété supprimée ou une chaîne vide dans le cas où cette propriété n'a pas été trouvée dans `decl`.

void `decl.setProperty(`*string* `nom,` *string* `val,` *string* `priorit)`

Cette méthode définit dans la déclaration `decl` de styles, la propriété `nom` avec pour valeur `val` et munie d'une priorité `priorit` *(ayant pour valeur la chaîne* `"important"` *ou la chaîne vide)*.

IX. Les événements :

A. Introduction :

JavaScript est un langage de programmation :

- *séquentiel* : les informations se trouvant à l'intérieur des balises <`script`> et </`script`> s'exécute les unes après les autres dans l'ordre de leurs apparitions.

- *procédural* : des procédures *(à mauvais titre appelés des fonctions)* rassemblent des bout de code à exécuter plusieurs fois dans le script. Ce côté présente l'avantage de donner un code plus compact, plus lisible et rendant plus facile la maintenance.

- *orienté objet* : la programmation **JavaScript** consiste à manipuler des objets au travers de propriétés et méthodes de chacun d'eux. Plus précisemment, le programmeur peut également créer ses propres objets afin de modéliser son objectif.

Le langage **JavaScript** est également un langage évenementiel : le script reçoit des informations de la page Internet et pourra lancer des actions en conséquence. Ainsi, il y a interaction entre le script et la page Web.
Des parties du code pourront être exécuter si :

- le client passe sa souris sur un élément de la page ;

- le client appui sur une touche ;

- le client quitte la page...

Ce passage est celui qui présente le plus de différence entre le standart préconisé par le W3C et la manière dont Internet Explorer gère les évéenements.

B. Liste des événements :

Voici une liste non-exhaustive des événements pris en charge par **JavaScript** :

Nom	Actionné par	Applicable sur
onclick	un clic du clien	Presque tous les éléments HTML
ondblclick	un double clic	
onmouseover	la souris passe par dessus un élément	
onmouseout	la soutis quitte un élément	
onmousedown	un bouton de la souris est enfoncé	
onmouseup	un bouton de souris est relaché	
onmousemove	la souris est en déplacement au dessus d'un élément	
onkeypress	une touche est enfoncée puis relachée	
onkeydown	une touche est enfoncée	
onkeyup	une touche est relâchée	
onload	la page vient de se finir de charger	body, frameset
onunload	la page est changée ou le navigateur est fermé	
onsubmit	le formulaire est en train d'être soumi	form
onunload	la page est changée ou le navigateur est fermé	
onselect	le client sélectionnant un champ texte	input, textarea
onchange	la valeur du champ de saisie est modifiée	input, select, textarea
onfocus	activation de l'élément soit par la souris soit par la tabulation	a, area, input, select textarea, button
onblur	désactivation de l'élément soit par la souris soit par la tabulation	

C. La prise en charge des événements :

Au fur et à mesure de l'évolution du langage HTML et **JavaScript**, différents types de déclarer la gestion des événements sont apparus : trois manière de déclarer et de gérer l'apparation des événements se sont imposés.

Voici ces trois manières présentées suivant leur apparation chronologiques mais également suivant leurs niveaux de complexités :

1. On ajoute à l'élément HTML devant recevoir l'événement, l'attribut HTML correspondant à l' événement. Pour l'événement `clik`, l'attribut est `onclick` *(on rajoute "on" en préfixe au nom de l'événement)* ; sa valeur sera le code à exécuter :

```
1 Inscription directe du code
2 <div onclick="alert('Bonjour');">...</div>
3
```

```
Appel d'une fonction
<div onclick="affiche(event)">...</div>
<script>
function affiche(evt){
 alert("Evénement de type : "+evt.type);
}
</script>
```

Le mot-clef `event` représente ici, l'objet événement que nous étudierons plus tard.

2. Dans le modèle DOM, on ajoute au noeud-élément devant recevoir l'événement la propriété correspondant à l'événement *(on rajoute "on" en début du nom de l'événement).* La valeur de cette propriété sera une fonction *(sans argument)* ; lors de l'avénement de lévénement, cette fonction sera appelée et l'objet `event` sera passé en argument.

   Voici deux exemples d'assignation d'un événement par cette méthode :

```
<script>
Repérons d'abord l'élément
a=document.getElementById("cadre");

Premier exemple
function averti(evt){
 alert("Vous venez de passer sur l'élément");
}
a.onmouseover=averti;

Deuxieme exemple
a.getElementById("cadre").onmouseover=function(evt){
 alert("Vous venez de passer sur l'élément");
}
</script>
```

3. On peut adjoindre à un noeud-élément un gestionnaire d'écoute à l'aide de la méthode `addEventListener()` *(pour IE,* `attachEvent(detachEvent)` *)* et on enlève un gestionnaire d'écoute avec `removeEventListener()` *(pour IE,).*

   Voici un exemple d'implémentation d'une écoute par cette méthode :

```
<div id="boite">Bonjour</div>

```

```
<script>
function affiche(evt){
 alert("Evenement de type : "+evt.type);
}

a=document.getElementById("boite");
a.addEventListener("click",affiche,true);
</script>
```

Dans chacun des cas *(dans la valeur de l'attribut de l'événement ou dans le corps de la fonction appelé)*, le mot-clef `this` fait référence à l'objet recevant l'événement.

Discutons maintenant des avantages et différences de chacune de ces méthodes :

- La méthode 1. est facilement mis en oeuvre.
- Les méthodes 2. et 3. affectent les éléments de gestionnaire d'événements directement á partir du code. Elles permettent de séparer la partie HTML de la partie programmation. Dans le cas où tous les éléments d'un même type doivent recevoir le même écouteur, il sera plus facile de le faire à l'aide d'une boucle et de la méthode `getElementsByTagName()`
- Dans le cas de l'utilisation d'AJAX, la méthode 1. pose des problèmes, il est alors nécessaire d'utiliser l'une des deux autres méthodes.
- La méthode 3. est la plus récente et la plus avancée.
  Elle propose l'avantage de pouvoir affecter plusieurs écouteurs d'un même {evénement à un même élément. Elle contrôle le déclenchement de l'écouteur en fonction du sens de propagation de l'événement le long de l'arbre DOM *(cette question sera discutée plus tard)*.
  Elle permet également de pouvoir plus facilement gérer les écouteurs reliés à un élément : rajout ou effacement de l'un d'eux.

D. L'objet `Event` *:*

A chaque événement, **JavaScript** crée un objet `event`. Cet objet nous permettra de connaître des informations précises sur l'événement déclenché, ainsi qu'à contrôler certains aspects de celui-ci et sa possible propagation aux éléments de la page.

- dans le cas de la méthode 1., le mot-clef `event` permet de récupérer cet objet.
- Dans le cas des méthodes 2. et 3., la gestion de l'objet de type `Event` va différer d'Internet

Explorer et du modèle DOM :

- Dans le modèle DOM :
  La fonction déclenché par l'événement recevra l'élément de type `Event` comme son unique argument.

- Pour Internet Explorer :
  Pour récupérer l'objet `event`, il faut le rappler au travers de l'objet `window` :

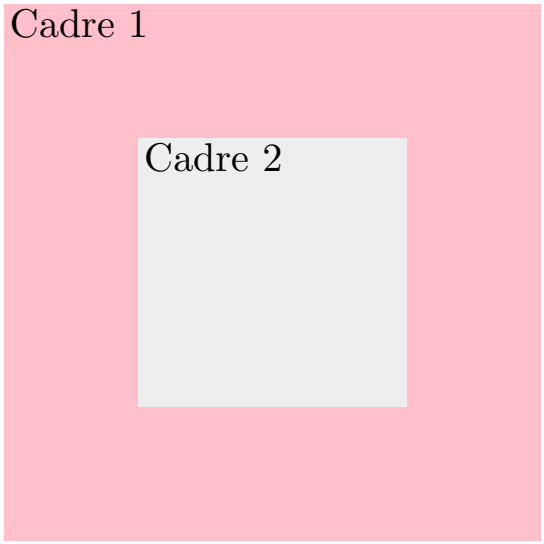
```
window.event
```

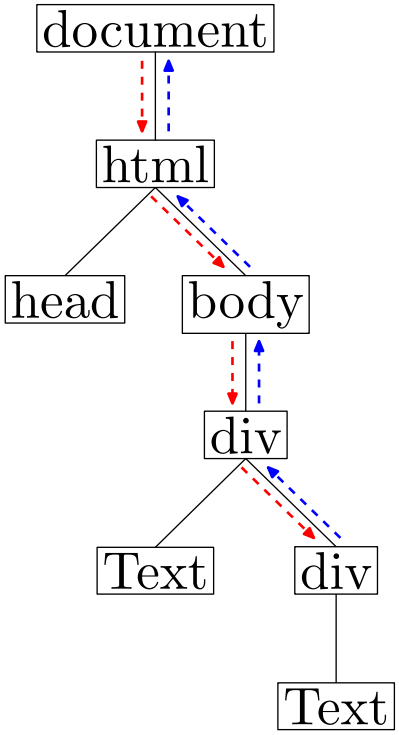
E. La propagation d'événement :

Nous allons au cours de ce paragraphe utiliser le code HTML suivant :

```
1 <body>
2  <div id="div1" style="position:absolute;height:200px;width:200px;background-color:pink">
3   Premier Cadre
4   <div id="div2" style="position:absolute;top:50px;left:50px;width:100px;height:100px;background-color:#EEEEEE">
5    Second Cadre
6   </div>
7  </div>
8 </body>
```

Ci-dessous est représenté, à droite le rendu dans un navigateur et à gauche l'arbre DOM représentant cette page :





Lorsque le client clique à l'emplacement du cadre gris, a-t-il voulu cliquer sur *div1*, *div2*, sur `body` ou sur l'ensemble du document `document`.

Ainsi, une différence de conception apparue dès l'apparition des premiers navigateurs. Prenons l'exemple précédent d'un clic dans la partie grise :

- Pour Microsoft, le navigateur part de la racine de la page *(l'élément* `document`*)* et parcourt l'ensemble de l'arbre jusqu'à arriver jusqu'au cadre gris.
  Tout noeud-élément possédant un écouteur d'événement `click` aura son code exécuté : dans l'ordre des flèches bleus.
  Cette phase s'appelle la phase de "*capture*" *(en anglais capture)*.

- Pour Netscape, le navigateur part à la recherche des éléments ayant la gestion de l'événement `click` dans le sens des flèches bleus.
  Ainsi, Netscape part à la recherche des éléments réagissant au `click`, en partant du cadre gris jusqu'à la racine de l'arbre.
  Cette phase s'appelle la phase de "*bouillonnement*" *(en anglais bubbling)*.

Ainsi, le rajout du code suivant :

```
1 <script>
2 function a(){alert("un");}
3 function b(){alert("deux");}
4
5 document.getElementById("div1").onclick=a;
```

```
6 document.getElementById("div2").onclick=b;
7 </script>
```

aura l'effet suivant :

- Avec Netscape, la fenêtre avec "un" s'affichera d'abord puis celle avec "deux".

- Actuellement, tous les autres navigateurs sont en mode "*bubbling*" ainsi l'événement affectera d'abord l'élément `div2` puis l'élément `div2`.

Actuellement le modèle DOM mélange ces deux conceptions : l'év'enement part de la racine du document jusqu'à la cible pour retourner à la racine du document.

Les méthodes 1. et 2. n'agissent que sur la phase ascendante *(bubbling)*. Pour intercepter, l'événement dans la phase de "*capture*", il faudra utiliser la méthode 3. de déclaration des écouteurs d'événements à l'aide de la méthode `addEventListener()`.

Ainsi le code suivant rajouté à notre page :

```
1 <script>
2 function a(e){alert("Evenement : "+e.type+"\r\nAction ↩
    actuelle sur : "+e.currentTarget.id+"\r\nCible : "+e.↩
    target.id);}
3
4 function b(e){alert("Evenement : "+e.type+"\r\nAction ↩
    actuelle sur : "+e.currentTarget.id+"\r\nCible : "+e.↩
    target.id);}
5
6 document.getElementById("div1").addEventListener('click↩
    ',a,true);
7 document.getElementById("div2").addEventListener('click↩
    ',a,true);
8 document.getElementById("div1").addEventListener('click↩
    ',b,false);
9 </script>
```

et un click sur la partie grise donnera dans l'ordre :

- l'exécution de "*div1*" en phase capture ;

- l'exécution de "*div2*" en tant que cible en début de phase "*bubbling*" ;
- l'exécution de "*div1*" en phase "*bubbling*".

F. Propriétés et méthodes des événements :

On expose ici les propriétés et les méthodes utilisées pour la gestion des événements dans le modèle DOM. Les recommendations pour Internet Explorer seront données dans un autre chapitre.

- Pour les noeuds-éléments :

void `elt.addEventListener(`*string* `event,` *function* `action,`*boolean* `propag)`

`evt` représente l'événement considéré dont l'écouteur sera rajouté à l'élément `elt`. Lors de la venue de l'événement, la fonction `action` est exécuté.
Si `propag` a pour valeur `true`, alors l'événement n'aura lieu que dans la phase de capture ; avec la valeur `false` seul la phase "*bubbling*".

void `elt.removeEventListener(`*string* `event,`*function* `action,` *boolean* `propag)`

Cette méthode effacera de l'élément `elt`, l'écouteur s'il existe de l'événement `event` appelant la fonction `action` et défini pour le mode de propagation défini par le booléen `propag` *(*`true` *pour cascade et* `false` *pour bubbling).*

boolean `elt.dispatchEvent( evt`

Cette méthode enclenche sur l'élément `elt` l'évément `evt`.
Cet événement aura pû être créé au préalable par :

```
document.createEvent("click");
```

Cette méthode renvoye si au moins un des gestionnaires d'événement a exécuté :

```
evt.preventDefault()
```

- Pour les événements :

boolean `evt.bubbles`

Ce booléen indique si l'événement `evt` accepte la phase "*bubbling*" ; certains événement n'accepte qu'une phase de capture.

boolean `evt.cancelable`

Cette propriété indique si l'événement `evt` est annulable ou non.

node `evt.currentTarget`

Cette propriété représente la cible de l'événement *(ce n'est pas forcément* `elt`*).*

int `evt.eventPhase`

Cette propriété représente la phase dans laquelle se trouve l'événement `evt` lors de sa capture :

1 en phase de capture ; 2 lors de la présence sur la cible ; 3 en phase de bubbling

node `evt.target`

Cette propriété donne une référence du noeud où se trouve l'élément sur lequel à été capturé l'événement `evt`.

timeStamp `evt.timeStamp`

Cette propriété contient le timeStamp en milliseconde représentant le temps de création de l'événement `evt`.

string `evt.type`

Cette propriété contient le type d'événement de `evt` *(click, onmouseover, onload...).*

void `evt.preventDefault()`

Cette méthode permet d'annuler l'action normale de l'événement sur l'élément considéré.

Par exemple, on peut annuler le fait de cocher une case en récupérant l'événement `click` et en exécutant cette méthode.

void `evt.stopPropagation()`

Cette méthode arrête la propagation de l'événement le long de l'arbre *(sans arrêter l'exécution en cours)* quelque soit la phase de parcours *(capture ou bubbling).*

G. Pour Internet Explorer :

Ne seront présenté ici que les différences fondamentales de la conception d'Internet Explorer avec le modèle DOM utilisé par les autres navigateurs.

- La méthode `addEventListener()` n'existe pas, elle est remplacée par :

`elt.attachEvent(`*string* `evt,` *function* `action)`

Cette méthode rajoute à l'élément `elt` un écouteur pour l'événement `evt` *(son nom doit être précédé de "on")* et `action` représente la fonction à exécuter lors de la survenue de l'événement correspondant.

On remarque qu'il n'y pas sous Internet Explorer de gestion de la propagation de l'événement, en effet, le seul mode de propagation bubbling existe sous IE.

La méthode `removeEventListener` n'existe pas, elle est remplacée par :

`elt.detachEvent(`*string* `evt,` *function* `action)`

Cette méthode supprime l'écouteur de l'évément `evt` pour l'élément `elt` utilisant la fonction `action`.

- La méthode `currentTarget` du modèle DOM n'a pas d'équivalent avec Internet Explorer. On peut utiliser le mot-clef `this` pour récupérer à l'intérieur de la fonction s'exécutant la référence à l'élément courant.

  Mais ceci n'est possible que pour les méthodes 1. et 2. :
  C'est lélément qui appelle directement la fonction.
  Dans le cas de la méthode 3., c'est l'objet `window` qui appellera la fonction pour l'objet : `this` se référera à l'objet `window`.

- La propriété `evt.target` du DOM est remplacée par :
  `elt.srcElement`

- La méthode `elt.preventDefault()` du DOM est remplacée par la propriété :
  `elt.returnValue`
  Cette propriété doit être mise à `false` pour stopper l'action normale de l'événement.

- La méthode `evt.stopPropagation()` du DOM est remplacée par la propriété :
  `evt.cancelBubble`
  Cette propriété arrête la propagation de l'événement si sa valeur est mise à `true`.

X. Annexe :

H. Les propriétés de styles et **JavaScript** *:*

background⟺ background	counterReset⟺ counter-reset
backgroundAttachment⟺ background-attachment	cssFloat⟺
backgroundColor⟺ background-color	cssText⟺
backgroundImage⟺ background-image	cursor⟺ cursor
backgroundPosition⟺ background-position	direction⟺
backgroundRepeat⟺ background-repeat	display⟺ display
border⟺ border	emptyCells⟺ empty-cells
borderBottom⟺ border-bottom	font⟺ font
borderBottomColor⟺ border-bottom-color	fontFamily⟺ font-family
borderBottomStyle⟺ border-bottom-style	fontSize⟺ font-size
borderBottomWidth⟺ border-bottom-width	fontSizeAdjust⟺
borderCollapse⟺ border-collapse	fontStretch⟺ font-stretch
borderColor⟺ border-color	fontStyle⟺ font-style
borderLeft⟺ border-left	fontVariant⟺ font-variant
borderLeftColor⟺ border-left-color	fontWeight⟺ font-weight
borderLeftStyle⟺ border-left-style	height⟺ height
borderLeftWidth⟺ border-left-width	left⟺ left
borderRight⟺ border-right	length⟺ ??
borderRightColor⟺ border-right-color	letterSpacing⟺ letter-spacing
borderRightStyle⟺ border-right-style	lineHeight⟺ line-height
borderRightWidth⟺ border-right-width	listStyle⟺ list-style
borderSpacing⟺ border-spacing	listStyleImage⟺ list-style-image
borderStyle⟺ border-style	listStylePosition⟺ list-style-position
borderTop⟺ border-top	listStyleType⟺ list-style-type
borderTopColor⟺ border-top-color	margin⟺ margin
borderTopStyle⟺ border-top-style	marginBottom⟺ margin-bottom
borderTopWidth⟺ border-top-width	marginLeft⟺ margin-left
borderWidth⟺ border-width	marginRight⟺ margin-right
bottom⟺ bottom	marginTop⟺ margin-top
captionSide⟺ caption-side	markerOffset⟺
clear⟺ clear	marks⟺
clip⟺ clip	maxHeight⟺ max-height
color⟺ color	maxWidth⟺ max-width
content⟺ content	minHeight⟺ min-height
counterIncrement⟺ counter-increment	minWidth⟺ min-width

opacity⟺ opacity	position⟺ position
outline⟺ outline	quotes⟺
outlineColor⟺ outline	right⟺ right
outlineOffset⟺ outline-offset	size⟺
outlineStyle⟺ outline-style	tableLayout⟺ table-layout
outlineWidth⟺ outline-width	textAlign⟺ text-align
overflow⟺ overflow	textDecoration⟺ text-decoration
overflowX⟺	textIndent⟺ text-indent
overflowY⟺	textShadow⟺
padding⟺ padding	textTransform⟺ text-transform
paddingBottom⟺ padding-bottom	top⟺ top
paddingLeft⟺ padding-left	unicodeBidi⟺
paddingRight⟺ padding-right	verticalAlign⟺ vertical-align
paddingTop⟺ padding-top	visibility⟺ visibility
page⟺	whiteSpace⟺ white-space
pageBreakAfter⟺	width⟺ width
pageBreakBefore⟺	wordSpacing⟺ word-spacing
pageBreakInside⟺	zIndex⟺ z-index
readonly⟺ CSSRule parentRule	⟺

Index