

3I017 - TECHNOLOGIES DU WEB

Les bases en Javascript

Thursday 14th March, 2019

Laure Soulier



Le web dynamique

Scripts Web

- HTML / CSS permettent de produire des pages Web avec dispositions graphiques évoluées mais
 - Pages peu dynamiques
 - Interactions utilisateur limitées
- Langages de script pour le Web
 - Permettent de combler ce manque en donnant la possibilité de définir des fonctions
 - De modification des éléments affichés (modification éléments / attributs / propriétés de l'arbre DOM)
 - De réaction à des actions utilisateur
 - De communication client / serveur
 - Exécution de traitements côté client
 - Fortement dépendants du navigateur appelant la page web

- Statique

```
<html>
<head>
<title>Page statique</title>
</head>
<body>
<p> La date du jour est le 25 fevrier 2019 </p>
</body>
</html>
```

- Dynamique

```
<html>
<head>
<title>Page dynamique</title>
</head>
<body>
<script type = "text/javascript">
    date = new Date();
    document.writeln("La date du jour est le ", date);
</script>
</body>
</html>
```

- Différents langages
 - Javascript (Netscape & Sun)
 - VBscript (Microsoft)
 - XUL (Mozilla)
 - XSLT (W3C)
- European Computer Manufactures Association (ECMA)
 - ⇒ Standard ECMA 262

- Java : Langage objet compilé, qui peut être exécuté côté client par le biais d'une applet
- Javascript : Langage de scripts moins évolué, intégré aux pages Web, qui s'inspire de différents langages (dont Java)

JavaScript	Applet Java
Langage interprété	Langage compilé
Interprétation par le navigateur	Chargement d'une machine virtuelle
Code intégré au HTML	Code appelé à partir de la page
Langage peu typé	Langage fortement typé
Accessibilité du code	Confidentialité du code

Insérer du code Javascript

Javascript dans balises

```
<!-- Pour differentes balises
      (avec eventHandler le nom d'un gestionnaire d'evenement) -->
<balise eventHandler="javascript:(function(){...})">

<!-- ou en tant qu'action d'un formulaire -->
<form action="javascript:(function(){...})">
```

Javascript dans `<script> ... </script>`

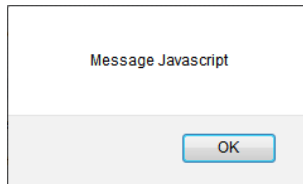
```
<script type="text/javascript"> Code javascript </script>
```

Javascript dans un ou plusieurs fichiers séparés

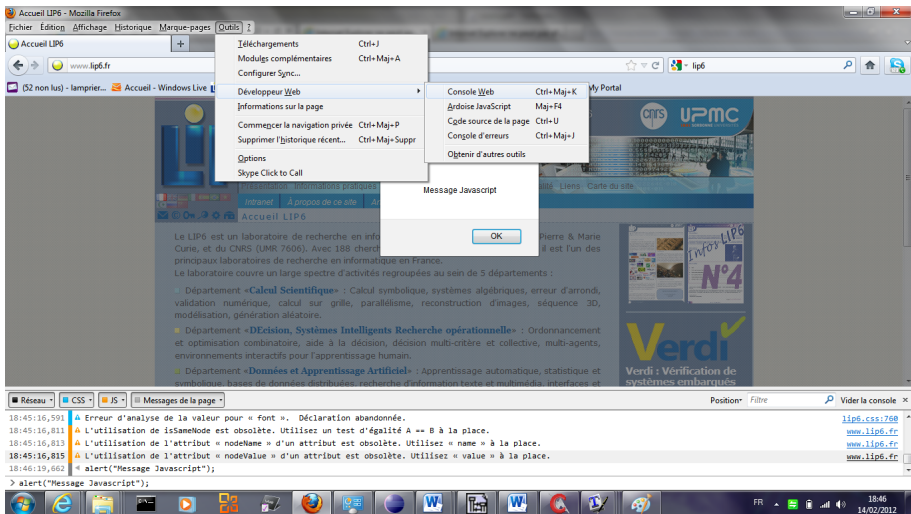
```
<script type="text/javascript" src="scripts/code.js"></script>
```


Exemple premier code Javascript

```
<HTML>
<HEAD>
<TITLE>Exemple Javascript </TITLE>
</HEAD>
<BODY>
<SCRIPT type="text/javascript">
<!--
alert(" Message Javascript ");
// -->
</SCRIPT>
</BODY>
</HTML>
```



Debugger du Javascript : Console Web



The screenshot shows a Mozilla Firefox browser window displaying the website www.lip6.fr. The browser's developer tools are open, with the 'Console' tab selected. The console displays several error messages and a JavaScript alert.

Developer Tools - Console:

- 18:45:16,591 Erreur d'analyse de la valeur pour « font ». Déclaration abandonnée.
- 18:45:16,811 L'utilisation de isSameNode est obsolète. Utilisez un test d'égalité A == B à la place.
- 18:45:16,813 L'utilisation de l'attribut « nodeName » d'un attribut est obsolète. Utilisez « name » à la place.
- 18:45:16,815 L'utilisation de l'attribut « nodeValue » d'un attribut est obsolète. Utilisez « value » à la place.
- 18:46:19,662 alert("Message Javascript");

JavaScript Message:

Message Javascript

OK

The browser window shows the website's navigation menu, including 'Accueil LIP6', 'Intranet', and 'À propos de ce site'. The main content area features a banner for 'Verdi' (Vérification de systèmes embarqués) and a section titled 'Le LIP6 est un laboratoire de recherche en informatique en France'.

Langage Javascript

Javascript : Les variables

Variables locales / Variables globales

```
i = 0; // Variable globale
var j = 'Technos Web'; // Variable globale

function f(){
  var k = 'Javascript'; // Variable locale
  l = 20; // Variable globale (car sans le mot-cle var)
  return i*l; // x est accessible ici car il s'agit d'une variable
}
// k n'est plus accessible ici (car locale)
```

Javascript : Les Fonctions

- Une fonction est un bloc d'instructions acceptant une liste de paramètres
 - Définie par le mot-clé function
 - Peut posséder un nom
 - Retourne éventuellement une valeur (grâce à return)

Fonction Javascript

```
function nom_fonction(parametre_1, ... , parametre_n) {  
    instructions;  
    return expression;  
}
```

- On peut stocker une fonction dans une variable

Fonction anonyme

```
var f = function(parametre_1, ... , parametre_n) {  
    instructions;  
    return expression;  
};
```

Javascript : Les Fonctions

- Paramètres de fonctions
 - Pas obligatoirement même nombre de paramètres à l'appel et dans la définition
 - Affectation des paramètres dans l'ordre
 - Les paramètres peuvent être accédés par l'objet arguments \Rightarrow arguments[n-1] correspond à la n-ième valeur passée en paramètre
 - Si nombre de paramètres à l'appel $<$ nombre de paramètres attendus \Rightarrow paramètres supplémentaires = undefined
 - Si nombre de paramètres à l'appel $>$ nombre de paramètres attendus \Rightarrow paramètres additionnels peuvent être récupérés par l'objet arguments
 - Les paramètres sont passés par référence sauf les types de bases (nombres, caractères,...) qui sont passés par valeur (copie)

Paramètres

```
function fonc(x) {
    alert("x = "+x+" y= "+arguments[0]+"
        , z =" +arguments[1]+" , w = "+arguments[2]);
}
fonc(1,2); // Affiche x=1, y=1, z=2, w=undefined
fonc(1); // Affiche x=1, y=1, z=undefined , w=undefined
fonc(); // x=undefined , y=undefined , z=undefined , w=undefined
```


Javascript : Les Fonctions

- Une fonction peut être définie à l'intérieur d'une autre
 - La fonction interne peut accéder aux variables locales de la fonction externe...
 - ... Et s'en souvient même après en être sortie
- ⇒ On parle de fermeture

Fermeture

```
var x = 1;
function fonc() {
  var x=2;
  y = function() {alert(x);}
  alert(x);
}
alert(x);           // affiche "1" (car le x de fonc est local)
fonc();             // affiche "2" (normal, on est dans fonc)
y();                // affiche "2" et pas "1" (bien que hors de fonc)
```

Javascript : Les Tableaux

- Un tableau est un ensemble d'éléments repérés par leur indice (entier commençant à 0)
 - Création
 - Création littérale
Exemple : var tab = [0,1,2,3,4,5];
 - Création grâce au constructeur Array
Exemple : var tab=new Array(10);
 - Accès aux éléments par la notation tableau[indice]
Exemple : tab[0]=10; var x=tab[0];
 - Propriété length qui représente la longueur du tableau
 - L'ajout d'un élément à un indice supérieur à cette taille, augmente automatiquement length
 - Décrémenter length revient à supprimer le dernier élément
 - Méthodes spécifiques comme join, slice, et push

- Contrairement aux langages objets classiques (Java, C++, ...)
 - La notion de classe n'existe pas
 - Le langage n'est pas typé
- En JavaScript, un objet correspond à une sorte de tableau associatif
 - Chaque élément d'un objet correspond à une "entrée" (identifiée par un nom)
 - Un attribut correspond à une entrée avec un type quelconque
 - Une méthode correspond à une entrée dont le type attendu est fonction
 - Manipulation dynamique
 - Possibilité d'ajouter, modifier ou supprimer les entrées de l'objet tout au long de sa vie
 - Mais pas de length, ni fonctions join, splice, push, etc...

Javascript : Les objets

- Création d'objets
 - Deux modes de création
 - Création à l'aide de constructeur
 - Construction littérale
- Création par constructeur
 - Un constructeur est une fonction qui associe des valeurs (pouvant être des fonctions) à des attributs
 - Un constructeur est appelé par le mot-clé new
 - Constructeur prédéfini Object
Exemple : var obj=new Object();
- Construction littérale
 - Déclaration entre {...} d'associations attributs-valeurs
Exemple : var obj={x : 1, y : 'chaine'};
 - Les valeurs associées peuvent elles-mêmes être des objets
⇒ Base du format JSON

- Les propriétés des objets peuvent être créées, lues et écrites
 - Soit avec la notation "point" objet.propriété :
Exemple : var x=obj.couleur;
 - Soit avec la syntaxe utilisée pour les éléments de tableau :
Exemple : var x=obj['couleur'];

Javascript : Les objets

- Constructeurs
 - Simples fonctions d'association propriété-valeur
 - Pas de notions de classes comme dans autres langages
 - Par convention, noms de constructeurs commencent par une majuscule
- Possibilité d'écrire ses propres constructeurs d'objets
 - Utilisation du mot-clé this

this se rapporte à l'objet sur lequel a été appelé la fonction

Exemple : l'objet x dans x.a()

Dans le cas d'un constructeur, appel avec new

⇒ this se rapporte alors à l'objet nouvellement créé
 - Affectation de valeurs à différentes propriétés de l'objet

Exemple : this.couleur=2;

Javascript : Les objets

Constructeur d'objets

```
function MonObjet(param1, param2) {
    this.attribut1 = param1;
    this.attribut2 = param2;
}

objet = new MonObjet(5, 'bleu ');

alert(objet.attribut1);           // affiche 5
alert(objet["attribut2"]);        // affiche "bleu"

objet.attribut3 = new Date();     // ajoute une nouvelle propriete a l'objet
alert(objet.attribut3);           // affiche la date d'ajout de attribut3

MonObjet.statique = "serif";      // ajoute une propriete statique
alert(MonObjet.statique);         // affiche "serif"
alert(objet.statique);            // affiche undefined

delete objet.attribut2;           // enleve une propriete a l'objet
alert(objet.attribut2);           // affiche undefined

delete objet;                    // supprime l'objet entier (rarement utilise)
alert(objet.attribut1);           // declenche une exception
```


Javascript : Les objets

- this permet d'affecter des attributs / méthodes à un objet
 - On peut y accéder de l'extérieur du constructeur
 - ⇒ Ils sont donc publics
- Grâce au mécanisme de fermeture, on peut définir des attributs / méthodes privés :

Constructeur d'objets

```
function MonObjet(val) {
  var a = val;
  this.getA = function() {
    return(a);
  }
  this.setA = function(newVal) {
    a = newVal;
  }
}

var obj = new MonObjet(1);
console.log("A: " + obj.a); // Affiche undefined (a non d fini)

console.log("A: " + obj.getA()); // Affiche "A: 1" (getA est un accesseur)

obj.a=2;
console.log("A: " + obj.getA()); // Affiche "A: 1" (la valeur n'a pas change)

obj.setA(2);
console.log("A: " + obj.getA()); // Affiche "A: 2" (setA est un modificateur)
```

Javascript : Les objets

- On peut faire la même chose avec les méthodes :

Constructeur d'objets

```
function MonObjet(parametre1, parametre2) {
    var attribut1 = parametre1;
    var attribut2 = parametre2;
    var methode_privée = function() {
        console.log("Attributs: " + attribut1 + ", " + attribut2);
    }
    this.methode_publique = function() {
        methode_privée();
    }
}

var obj = new MonObjet(1, 2);
alert("Attribut1: " + obj.attribut1); // Affiche undefined
// (attribut1 non défini)

obj.methode_privée(); // "TypeError: obj.methode_privée not a function"
obj.methode_publique(); // Affiche "Attributs: 1, 2"
```


Javascript : Construction littérale

- Comme on l'a vu précédemment :
 - Un objet peut être instancié par un constructeur...
 - ... ou formé par une déclaration littérale
- Construction littérale :
 - Un objet se construit par `{ }`
 - Un tableau se construit par `[]`
 - Possibilité de former des imbrications d'objets / tableaux

Exemple de construction littérale

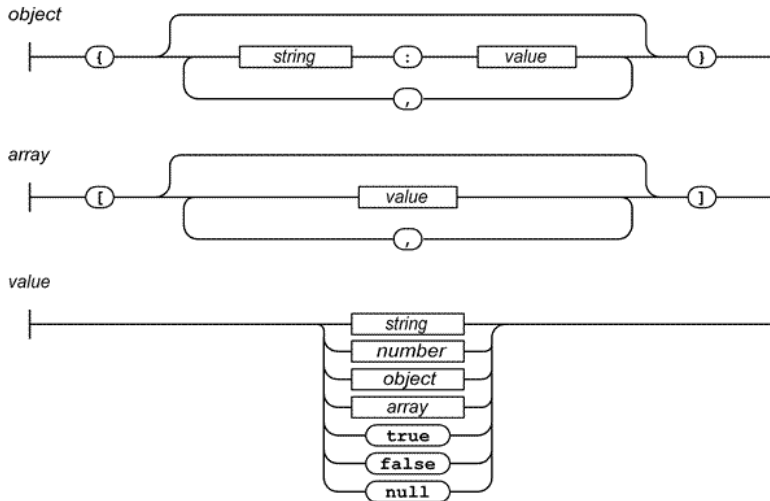
```
var obj={"menu": {
  "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      {"value": "New", "onclick": function(){alert("Create New Doc");}},
      {"value": "Open", "onclick": function(){alert("Open Doc");}},
      {"value": "Close", "onclick": function(){alert("Close Doc");}}
    ]
  }
}}

obj.menu.popup.menuitem[1].onclick(); // Affiche "Open Doc"
```

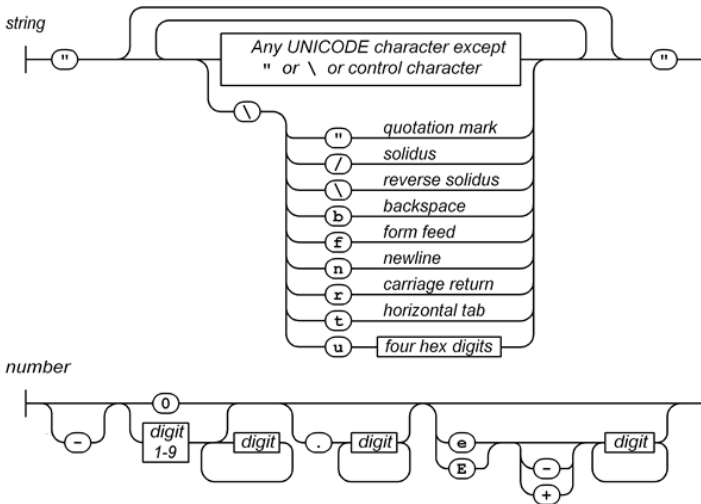
Javascript : Le format JSON

- JSON (JavaScript Object Notation)
 - Format de données textuel
 - Dérivée de la construction littérale d'objets
- Format JSON est composé :
 - d'ensembles de paires nom / valeur \Rightarrow les objets
 - de listes ordonnées de valeurs \Rightarrow les tableaux

Javascript : Le format JSON



Javascript : Le format JSON



- Le format JSON = chaîne de caractères correspondant à la formation littérale d'un objet
 - ⇒ Nécessite de disposer :
 - D'un parser : texte JSON ⇒ objet
 - D'un serializer : objet ⇒ texte JSON

Javascript : Parser du JSON

- Parser : la fonction `eval(string)`
 - Permet d'interpréter une chaîne de caractères
 - Puisque le format JSON = chaîne de construction littérale, `eval('('+json_text+')')` construit l'objet correspondant au texte contenu dans `json_text`
 - Mais :
 - `eval` est une fonction générique permettant d'évaluer n'importe quelle chaîne représentant du code Javascript
- ⇒ Problèmes de sécurité car du code nuisible peut être exécuté lors de la transformation du texte JSON

Faille de sécurité

```
// JSON transmis par le serveur :
json_texte="{\"g\":1,\"f\":\"json\"}";

var obj=eval("("+json_texte+")"); // construction de l'objet

// JSON tronqu  lors du transfert :
json_texte="function(){alert('Hack!')}(";

var obj=eval("("+json_texte+")"); // Affiche "Hack!"
```

Javascript : Parser du JSON

- Depuis 2009, les navigateurs intègrent un support JSON comportant une fonction parse(json_text, revive)
 - json_text : chaîne JSON à transformer
 - revive (facultatif) : méthode appelée sur chaque couple (clé,valeur) à chaque niveau de la construction de l'objet⇒ Méthode spécifique n'interprétant pas d'autre code qu'une chaîne de construction JSON

JSON.parse

```
// JSON transmis par le serveur :
json_texte="{\"g\":1,\"f\": \"json\"}";

// construction de l'objet
var obj=JSON.parse(json_texte);

Pour un parsing fonctionnant sur n'importe quel navigateur :
var obj = typeof JSON !=='undefined' ?
    JSON.parse(json_texte) : eval('(' + json_texte + ')');

// JSON tronqué lors du transfert :
json_texte="function(){alert('Hack!')})();";

// Affiche "SyntaxError: JSON.parse: unexpected keyword"
var obj=JSON.parse(json_texte);
```

Javascript : Serializer en JSON

- Le support JSON comporte également une fonction stringify(objet, replacer);
 - objet : objet à transformer en chaîne JSON
 - replacer (facultatif) : méthode appelée sur chaque couple (clé,valeur) à chaque niveau de la structure de l'objet pour spécifier un traitement spécial

JSON.stringify

```
//Objet      serializer en JSON
var obj=new Object();
obj.g="1";
obj.f="json ";

var json_text=JSON.stringify(obj);
// json_text contient "{ \"g\": \"1\", \"f\": \"json\" }"
```

⇒ **Attention** : `JSON.stringify` ne serialise pas ce qui est dans le prototype

Javascript : JSON

Exemples utilisation `replacer` et `revival` : objet `Date()`

```
var obj={g:1, r:new Date()};
obj; // Affiche "({g:1, r:(new Date(1329482734849))})" sur la console Web

json_text = JSON.stringify(obj, function (key, value) {
    return this[key] instanceof Date ? 'Date(' + this[key] + ')' :
});

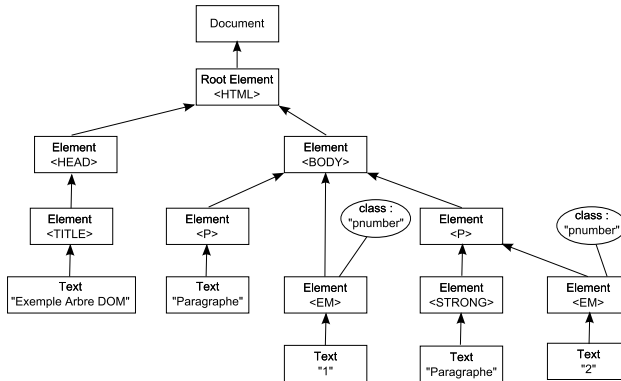
obj2=JSON.parse(json_text, function (key, value) {
    var d;
    if (typeof value === 'string' &&
        value.slice(0, 5) === 'Date(' &&
        value.slice(-1) === ')') {
        d = new Date(value.slice(5, -1));
        if (d) {
            return d;
        }
    }
    return value;
});

obj2; Affiche "({g:1, r:(new Date(1329482734000))})" sur la console Web
```

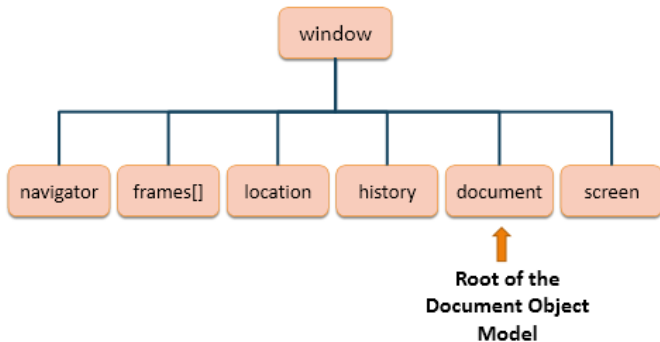

Arbre DOM

- Arbre DOM (Document Object Model)
 - Standard du W3C
 - Décrit une interface indépendante de tout langage de programmation
 - ⇒ Accéder ou de mettre à jour le contenu, la structure ou le style de documents XML et HTML
- Arbre BOM (Browser Object Model)
 - Pas de standard sur la manière de gérer les éléments du navigateur
 - Nécessité de produire du code javascript adapté à tout navigateur / système d'exploitation
 - ⇒ BOM sert d'interface standard entre le navigateur et javascript

Document Object Model (DOM)



Browser Object Model (BOM)



- Lors de l'ouverture d'une page Web, le navigateur crée différents objets. Les principaux sont :
 - window : fenêtre d'affichage de la page (contient des propriétés sur la fenêtre mais aussi les objets de la page chargée)
 - navigator : qui contient des informations sur le navigateur
 - location : contient des informations relatives à l'adresse de la page à l'écran
 - history: historique de navigation (liste de liens visités précédemment)
 - document : contient les propriétés sur le contenu de la page Web chargée

Javascript : Objet Navigator

- **Objet Navigator** : Permet d'avoir des informations sur le navigateur utilisé
 - Infos générales sur le navigateur : navigator.userAgent;
 - Nom du navigateur : navigator.appName;
 - Version du navigateur : navigator.appVersion;
 - Méthodes de test sur le nom du navigateur : isIE(), isFirefox, isSafari(), ...
 - Système d'exploitation du poste client : navigator.platform;
 - Langue utilisée par le navigateur : navigator.language;
 - Types de données supportées par le navigateur : navigator.mimeTypes;
 - Liste de plugins installés : navigator.plugins;
 - Méthode pour savoir si le Java est autorisé : navigator.javaEnabled();
 - Pour savoir si les cookies sont autorisés : navigator.cookiesEnabled;

- Objet History : Contient des infos sur l'historique
 - La propriété history.length permet de connaître le nombre d'objets dans l'historique
 - La méthode history.back() permet d'aller à l'URL précédent dans l'historique
 - La méthode history.forward() permet d'aller à l'URL suivant dans l'historique
 - La méthode history.go(variable) permet d'aller à un des URL de l'historique (variable peut être un index de page ou une chaîne proche de la page désirée).

Javascript : Objet Document

- Objet Document : Contenu de la page
- Propriétés
 - `alinkColor` : couleur des liens lorsqu'ils sont cliqués
 - `bgColor` : couleur d'arrière plan
 - `charset` : jeu de caractères utilisés
 - `cookie` : chaîne de caractères pouvant être sauvegardée chez l'utilisateur
 - `fgColor` : couleur du texte
 - `lastModified` : date de dernière modification
 - `linkColor` : couleur des liens
 - `referrer` : pages déjà visitées
 - `title` : titre de la page

- Un cookie est une paire attribut/valeur
 - Stocké dans le navigateur du client
 - Possède éventuellement une date péremption
 - Contient des informations envoyées par un serveur à un client HTTP
 - Envoyé par le client en entête de chaque communication HTTP avec le serveur qui l'a créé
- Sert à enregistrer chez le client
 - Des données d'authentification
 - L'identifiant de la session en cours sur le serveur
 - Des préférences de l'utilisateur
 - Le contenu d'un panier d'achat électronique
 - ...

Javascript: Objet Document

Gestion de cookies

```
function createCookie(name, value, days) {
    if (days) {
        var date = new Date();
        date.setTime(date.getTime()+(days*24*60*60*1000));
        var expires = ""; expires="+date.toGMTString()";
    }
    else var expires = "";
    document.cookie = name+"="+value+expires+"; path=/";
}

function readCookie(name) {
    var nameEQ = name + "=";
    var ca = document.cookie.split(';');
    for(var i=0;i < ca.length;i++) {
        var c = ca[i];
        while (c.charAt(0)==' ') c = c.substring(1,c.length);
        if (c.indexOf(nameEQ) == 0)
            return c.substring(nameEQ.length, c.length);
    }
    return null;
}

function eraseCookie(name) {
    createCookie(name, "", -1);
}

createCookie("a",1,10); // cr e un cookie a=1 expirant dans 10 jours
var val=readCookie("a"); // lit un cookie nomm a
```

- createAttribute(nom_attribut) : Crée un noeud attribut
- createElement(nom_balise) : Crée un noeud élément
- createTextNode(texte) : Crée un noeud de texte
- getElementById(id) : Retourne le noeud correspondant à l'identifiant passé en paramètre
- getElementsByTagName(nom) : Retourne un tableau contenant les éléments possédant le nom passé en paramètre
- getElementsByTagName(nom_balise) : Retourne un tableau contenant les éléments du type passé en paramètre

⇒ **Attention** : manipuler arbre DOM suppose d'attendre la fin du chargement de la page. Les traitements devant être faits lors de de l'affichage doivent être appelés par le gestionnaire d'évènement document.onload.

Javascript : Node

- Node = Noeud de l'arbre DOM
 - Différents types de noeud

Numéro	Type de nœuds
1	Nœud élément
2	Nœud attribut
3	Nœud texte
4	Nœud pour passage CDATA
5	Nœud pour référence d'entité
6	Nœud pour entité
7	Nœud pour instruction de traitement
8	Nœud pour commentaire
9	Nœud document
10	Nœud type de document
11	Nœud de fragment de document
12	Nœud pour notation

Javascript : Node

- Node = Noeud de l'arbre DOM
- Propriétés
 - nodeType : numéro du type du noeud
 - nodeValue : contenu du noeud
 - nodeName : nom du noeud
 - attributes : tableau de noeuds attribut associé
 - firstChild : premier noeud enfant
 - lastChild : dernier noeud enfant
 - nextSibling : noeud suivant dans l'arborescence
 - parentNode : noeud parent
 - previousSibling : noeud précédent dans l'arborescence

Javascript : Node

- Méthodes

- `appendData(texte)` : concatène du texte en fin de la valeur d'un noeud texte ou attribut
- `insertData(pos,texte)` : insère du texte à la position pos dans la valeur d'un noeud texte ou attribut
- `replaceData(pos,nb,texte)` : remplace du texte d'un noeud texte ou d'un noeud attribut (pos correspond à la position dans la chaîne et nb le nombre de caractères à supprimer)
- `deleteData(pos,nb)` : efface du texte d'un noeud texte ou d'un noeud attribut

- Méthodes

- `getAttribute(nom)` : retourne la valeur d'un noeud attribut
- `getAttributeNode(nom)` : retourne un noeud attribut
- `setAttribute(nom,valeur)` : fixe la valeur d'un noeud attribut
- `setAttributeNode(node)` : ajoute un noeud attribut
- `removeAttribute(nom)` : efface la valeur d'un noeud attribut
- `removeAttributeNode(node)` : supprime un noeud attribut

- Méthodes

- `appendChild(node)` : ajoute un noeud enfant (en tant que dernier enfant)
- `removeChild(node)` : retire un noeud passé en paramètre de la liste des fils du noeud
- `replaceChild(new,old)` : remplace un noeud enfant `old` par un noeud `new` dans la liste des fils du noeud
- `hasChildNodes()` : vérifie l'existence de noeuds enfants
- `insertBefore(new,avant)` : insère un noeud enfant avant un autre noeud enfant dans la liste de fils du noeud (`new` = nouveau noeud, `avant` = noeud enfant avant lequel insérer)
- `cloneNode(booleen)` : copie un noeud (si le booléen passé en paramètre est `true`, alors on copie aussi les fils du noeud)

Évènements

Gestionnaire d'évènement	Description
onClick	Lors d'un clic sur l'élément associé à l'évènement
onLoad	Lorsque le navigateur charge la page en cours
onUnload	Lorsque le navigateur quitte la page en cours
onMouseOver	Lorsque le curseur de la souris passe au-dessus de l'élément
onMouseOut	Lorsque le curseur de la souris quitte l'élément
onFocus	Lors de l'obtention du focus (élément sélectionné comme étant l'élément actif)
onBlur	Lors de la perte du focus (l'utilisateur clique hors de l'élément)
onChange	Lors de la modification du contenu d'un champ de données
onSelect	Lors de la sélection d'un texte (ou une partie d'un texte) dans un champ de type "text" ou "textarea"
onSubmit	Lors d'un clic sur un bouton de soumission d'un formulaire (le bouton qui permet d'envoyer le formulaire)

JQuery

L3 Info - 3I017

56/75

Javascript : JQuery

- La bibliothèque jQuery peut être appelée de trois manières différentes :

Via la fonction \$(expression)\$

```
$("div.test").add("p.quote").addClass("blue").slideDown("slow");
```

Via le préfixe jQuery(expression)

```
jQuery("#div a effacer").slideUp("fast");
```

Via le préfixe de fonction \$.

```
$$.each([1,2,3], function() {
    document.write(this + 1);
});
```


Javascript : JQuery

- Fonction `$()` permet différentes choses selon les paramètres
 - Une expression en CSS
 - ⇒ Permet de sélectionner facilement des éléments
 - ⇒ Forme un objet JQuery regroupant les noeuds correspondant à l'expressions CSS

Sélection d'objets avec sélecteurs CSS

```
// Sélection des éléments ayant pour classe "ique",
// dans l'élément ayant pour id "fixe"
var mon_objet_jquery = $("#fixe .ique");

// Sélection du premier élément <p> de l'élément <div>.
var mon_objet_jquery = $("div p: first-child");

// Sélection des éléments <p>, enfants directs d'un élément <div>
// et ayant pour classe "ique".
var mon_objet_jquery = $("div > p.ique");
```


Javascript : JQuery

- Fonction `$()` permet de regrouper des noeuds correspondant à un sélecteur passé en paramètre
 - ⇒ Objet JQuery contenant des noeuds DOM
 - ⇒ Possibilités d'y appliquer des méthodes JQuery
- Attention :
 - Fonctions Node pas applicables directement
 - ⇒ Accès aux éléments de l'objet par la méthode JQuery `get`
 - `get()` retourne un tableau contenant les noeuds sélectionnés
 - `get(index)` retourne le noeud à l'index `index`

Exemple utilisation JQuery

```
// Mauvaise utilisation JQuery (génère une erreur)
$("#comment > p").firstChild.appendData(" Post par Joe le 14/02/2012");

// Accès au premier élément de l'objet et application méthode au noeud
$("#comment > p").get(0).firstChild.appendData(" Post par Joe le 14/02/2012");

// équivalent :
$("#comment > p").get()[0].firstChild.appendData(" Post par Joe le 14/02/2012");
```

Javascript : JQuery

- Objet JQuery
 - ⇒ Propriété length : nombre d'éléments contenus
 - ⇒ Éléments rangés entre index 0 et length-1
 - ⇒ Méthode slice(debut,fin) : retourne l'objet jquery contenant uniquement les objets d'un jquery initial situés entre l'index debut et fin-1

Exemple utilisation JQuery

```
// Application m thode tous les lments s lectionn s
var jquery=$("#p");
for (i=0;i=jquery.length;i++){
    jquery.get(i).firstChild.appendData(" Post par Joe le 14/02/2012");
}
```


- before(x), after(x) : insère x avant ou après tous les éléments de l'objet (x peut être du code html, un élément DOM ou un objet JQuery)
- insertBefore(x), insertAfter(x) : insère tous les éléments de l'objet JQuery avant ou après l'objet x (x peut être un objet DOM ou une expression JQuery permettant de sélectionner des objets)
- append(x), prepend(x) : insère x à la fin ou au début de tous les éléments de l'objet
- appendTo(x), prependTo(x) : insère tous les éléments de l'objet JQuery à la fin ou au début de l'objet x
- replaceWith(x) : remplace tous les éléments de l'objet JQuery par x
- replaceAll(x) : remplace x par les éléments de l'objet

Javascript : JQuery

Exemple utilisation JQuery

```
<div class='a'>
  <div class='b'>b</div>
</div>
```

```
$('a').after($('c'));
<div class='a'>
  <div class='b'>b</div>
</div>
<div class='c'>c</div>
```

```
$('a').before($('c'));
<div class='c'>c</div>
<div class='a'>
  <div class='b'>b</div>
</div>
```

```
$('a').append($('c'));  
<div class='a'>  
  <div class='b'>b</div>  
  <div class='c'>c</div>  
</div>
```


Javascript : JQuery

- `animate(objectif, duree, [fonc])` : Produit une animation de transfert entre l'état courant et un état visé
 - Le paramètre `objectif` est un ensemble de `{propriété:valeur}`
 - ⇒ Uniquement sur les propriétés numériques
 - ⇒ Remplacer tirets par majuscules (exemple `margin-left` ⇒ `marginLeft`)
 - Le paramètre `duree` donne la durée de l'action
 - Le paramètre `fonc` est une fonction à lancer une fois l'action terminée

Animation JQuery

L3 Info 31017 Animation JQuery

Documentation Javascript :

<http://fr.selfhtml.org/javascript/index.htm> Documentation Javascript :

[https://developer.mozilla.org/en-](https://developer.mozilla.org/en-US/docs/Web/JavaScript/LanguageResources)

[US/docs/Web/JavaScript/LanguageResources](https://developer.mozilla.org/en-US/docs/Web/JavaScript/LanguageResources)

Documentation JQuery :

<http://api.jquery.com/>