

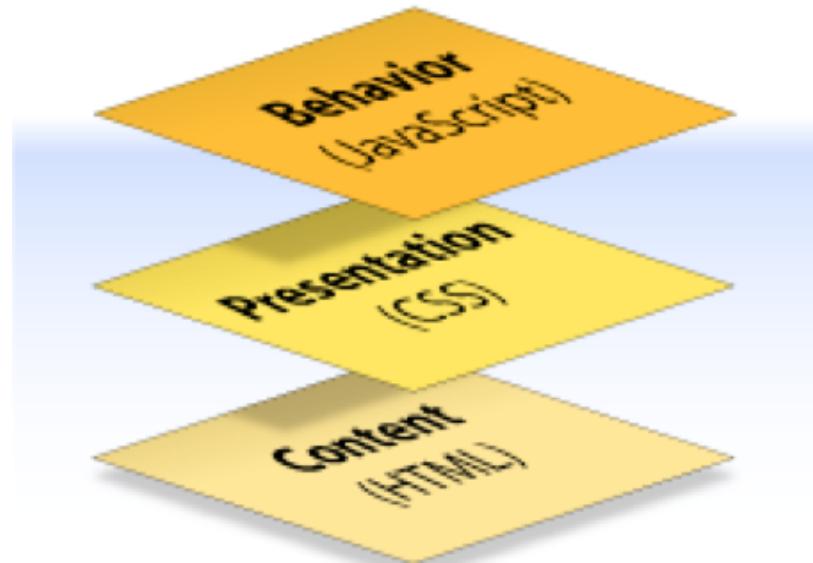
SHAKE THE FUTURE.



# Javascript

Myriam Servières

- Document web



SHAKE THE FUTURE.

# Plan

- Généralités
- Les bases
- Les objets
- Les évènements
- Les boites de dialogue

SHAKE THE FUTURE.



# Plan

- Généralités
  - Les bases
  - Les objets
  - Les évènements
  - Les boîtes de dialogue

SHAKE THE FUTURE.



# Intérêt

- Rendre dynamique les pages HTML

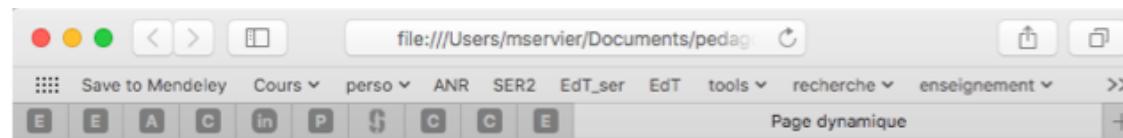
Exemple de page statique :

```
<!DOCTYPE html>
<head>
<title>
    Page statique
</title>
<meta charset="UTF-8"/>
</head>
<body>
    <p>Nous sommes le 23/01/2018</p>
</body>
</html>
```

SHAKE THE FUTURE.



# Intérêt



Nous sommes le 23/1/2018

SHAKE THE FUTURE.



# Intérêt [1]

- Page dynamique – obtention de la date courante à l'ouverture de la page :

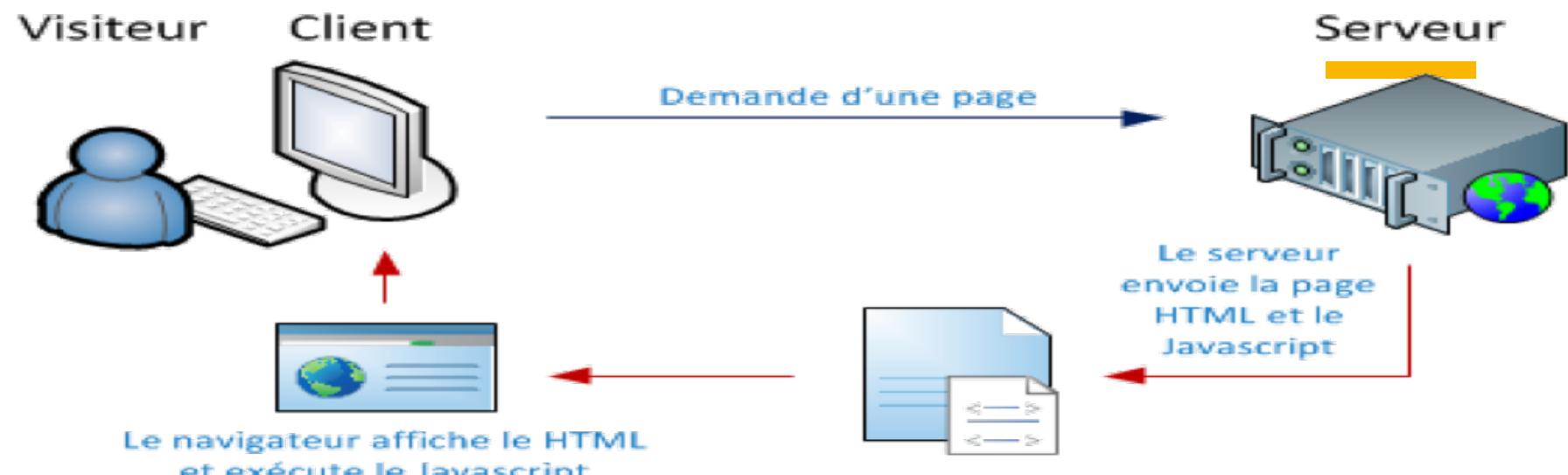
```
<!DOCTYPE html>
<head>
<title>
Page dynamique
</title>
<meta charset="UTF-8"/>
</head>
<body>
<p>
<script type="text/javascript">
<!--
//masquage du script pour les anciens navigateurs
    date = new Date();
    document.writeln("Nous sommes le ", date.getDate() + "/" + (date.getMonth() + 1)
    + "/" + date.getFullYear());
-->
</script>
</p>
</body>
</html>
```

SHAKE THE FUTURE.

# Script [1]

- Portion de code qui vient s'insérer dans une page HTML
- Le code du script n'est toutefois pas affiché dans la fenêtre du navigateur :
  - il est compris entre des balises spécifiques
  - qui signalent au navigateur qu'il s'agit d'un script écrit en langage JavaScript
- Interprété du côté client
- Utilisé coté serveur (ex: Node.js)

# Script



SHAKE THE FUTURE.

Source Openclassrooms



# Introduction à Javascript

- Le Javascript est un langage de programmation de scripts orienté objet.
- C'est un langage de script créé par Netscape
- Il sert à créer des pages web interactives
- Il est interprété (il n'y a pas de compilation) par le navigateur
- ! Javascript ≠ Java
  - Javascript ne peut pas lire/écrire dans des fichiers
  - Javascript ne peut pas exécuter d'autres programmes

# Rappel : le DOM (Document Object Model) [extrait de 5]

- Le navigateur affiche une page à partir de ces informations :

```
<!DOCTYPE html>
<html lang="fr-fr">
  <head>
    <title>test HTML</title>
    <meta charset="UTF-8" />
  </head>
  <body>
    <h1>Principe de l'exercice</h1>
    <p>La mise en oeuvre comprend :</p>
    <ul>
      <li>La création d'une page html</li>
      <li>Les balises de titre, de paragraphes, de listes, ...</li>
    </ul>
  </body>
</html>
```



# Rappel : le DOM (Document Object Model) [extrait de 5]

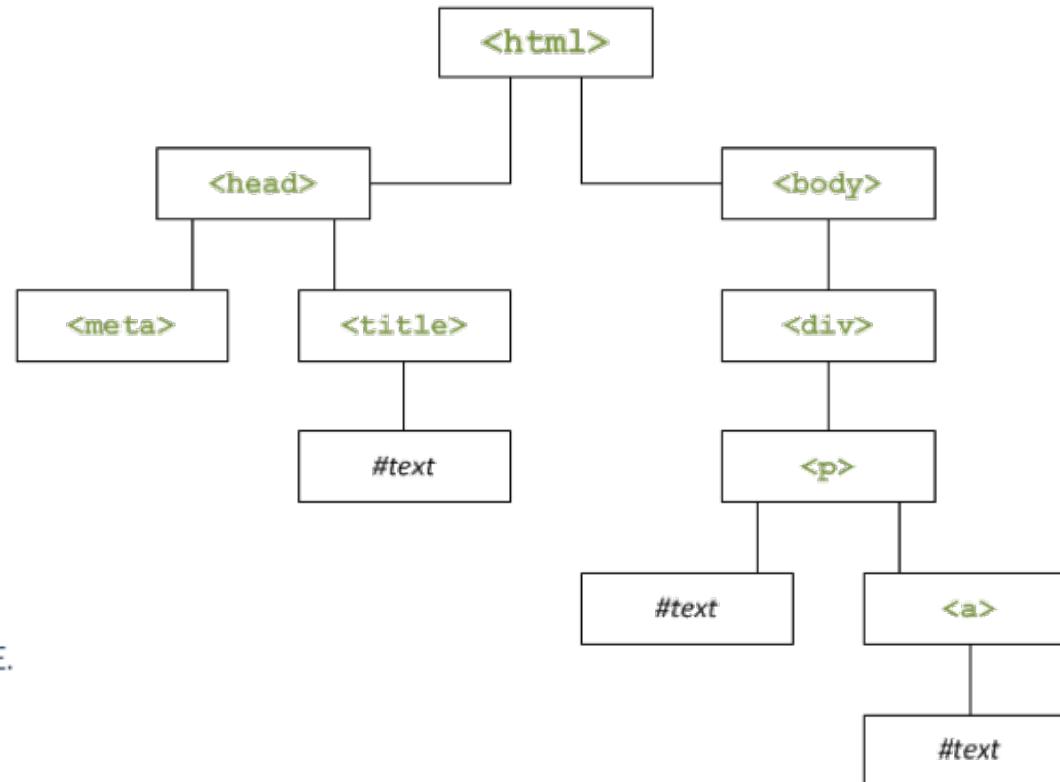
- Comment le navigateur manipule t'il ces informations ?
- Pour chaque élément présent, le navigateur possède un objet qui le décrit :
  - Sa position
  - Quel type (texte, bouton, ...)
  - Son style, sa classe (au sens CSS)
  - Dans quel objet est-il situé ?
  - Quels autres objets contient-il ?
  - ...
  - Plus tout un tas de méthodes

# Rappel : le DOM (Document Object Model) [extrait de 5]

- Le DOM = Document Object Model
- Interface de Programmation (API – Application Programming Interface) permettant :
  - d'accéder ou
  - de mettre à jour
  - le contenu, la structure ou le style de documents HTML et XML
- DOM = manipulation de :
  - L'ensemble des objets
  - Liens entre ces objets
- C'est-à-dire l'ensemble de tout ce qui est utilisé pour décrire la structure d'une page

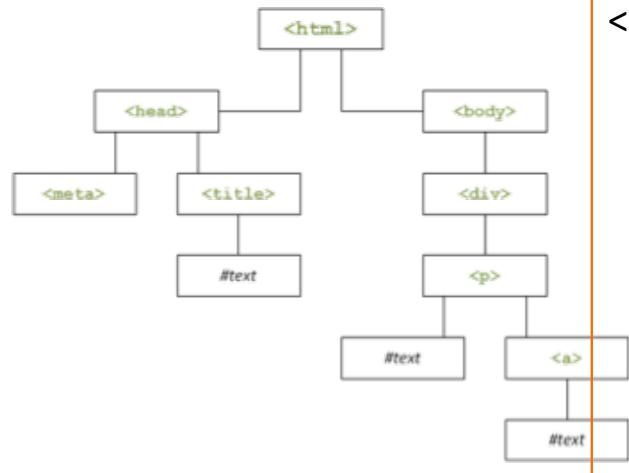
# Page Web et Dom [extrait de 5]

- Représentation schématique d'une page Web via le DOM :



# Page Web et Dom [extrait de 5]

- Code HTML correspondant :



```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
  </head>
  <body>
    <div>
      <p>Un peu de texte <a>et un lien</a></p>
    </div>
  </body>
</html>
```

# JavaScript et DOM [extrait de 5]

- Il est important de comprendre ce qu'est le DOM
- Car JavaScript va interagir avec nos pages HTML via le **DOM** !
- Pour ce faire, un utilisera un objet particulier du DOM :  
**document**
- L'objet document possède trois méthodes principales :
  - **getElementById(...)**
  - **getElementsByName(...)**
  - **getElementsByTagName(...)**

# JavaScript et DOM [extrait de 5]

- Le DOM nous permet de requêter le **document** afin d'obtenir les éléments de notre page HTML
- Les méthodes citées précédemment nous permettent de récupérer les éléments HTML par :
  - Leur identifiant :
    - `getElementById("monBouton")` permet de récupérer une balise dont on recherchera le id
    - `<input type= "button" id= "monBouton"> ...`
  - Leur « type » :
    - `getElementsByTagName("p")` permet de récupérer **toutes** les balises `<p>`
  - Leur nom :
    - `getElementsByName("monLabel")` permet de récupérer les éléments dont le nom est "monLabel »
    - `<label name="monLabel >Salut !</label>`
- Ceci nous oblige à définir une valeur spéciale qui représente l'absence d'élément !
- C'est fait via le mot clé **null** (qui est utilisé dans de nombreux langages de programmation)
- Par exemple si je demande un élément dont l'id est "salut" (avec `getElementById("salut")`) alors **qu'il n'y a aucun élément dans le document avec cet id** → le DOM va me retourner **null** !

# DOM : Objets `window` et `document` [extrait de 5]



Rq : `window` est l'objet global dans les navigateurs

Dans d'autres applications hôtes comme [Node.js](#), l'objet global est *global*

SHAKE THE FUTURE.



Source : <http://eligeske.com/jquery/what-is-the-difference-between-document-and-window-objects-2/>

# DOM : Objet Document [extrait de 5]

- **document**

- Représente le document effectivement affiché
- Nous permet de :
  - naviguer dans le document
  - récupérer les différents éléments de l'HTML
- Exemple :

```
document.getElementById("monId").innerHTML = "<p>Mon contenu</p>";
```

- On peut accéder aux éléments du HTML via :
  - Leurs identifiants (id)
  - Le types des balises (tags)
  - Leurs noms

# Les objets natifs du navigateur [extrait de 5]

- Trouver un élément HTML
  - `document.getElementById()`
  - `document.getElementsByTagName()`
  - `document.getElementsByName()`
- Changer un élément HTML
  - `element.innerHTML` = Change le contenu d'un élément HTML
  - `element.textContent` = Obtient le contenu texte d'un élément
  - `element.getAttribute(attribute)` = Change l'attribut d'un élément HTML
  - `element.setAttribute(attribute,value)` = Change l'attribut d'un élément HTML
  - `element.style.property` = Change le style d'un élément HTML

[http://www.w3schools.com/js/js\\_htmldom\\_document.asp](http://www.w3schools.com/js/js_htmldom_document.asp)

SHAKE THE FUTURE.



# Javascript dans les documents HTML

- On peut ajouter du code Javascript dans un document XHTML

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

- On peut aussi associer un fichier Javascript à un document XHTML

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

FR WEB

- Ou à un document HTML5

```
<script src="file.js"></script>
```

- Ou exécuter directement dans la console du navigateur

# Javascript dans les documents HTML

## [1]

- Les éléments situés dans l'en-tête se comportent comme des déclarations, ils ne s'exécutent pas directement
- Les éléments situés dans le corps s'exécutent au fur et à mesure du chargement de la page

PRWEB

SHAKE THE FUTURE.



# Javascript dans les documents HTML

- Utilisation dans une URL, en précisant que le protocole utilisé est du JavaScript ex :

```
<a href="javascript:instructionJavaScript;">Texte</a>
```

- Utilisation des attributs de balise pour la gestion événementielle :

```
<balise onEvenement="instructionJavaScript">...</balise>
```

PRWEB

SHAKE THE FUTURE.



# Javascript dans les documents HTML

## [1]

```
<!DOCTYPE html>
<head>
    <title>Exemple de page HTML contenant du JavaScript</title>
    <meta charset="UTF-8"/>
    <script type="text/javascript">
        function texte() { document.write("Texte généré."); }
    </script>
</head>
<body>
    <script type="text/javascript">
        document.write("Vous pouvez mettre du code javascript dans le corps du document.");
    </script>
    <p>
        Ou bien dans une fonction appelée en cliquant
        <a href="Javascript:texte()">ici</a>,
    </p>
    <p>
        ou en passant la souris au-dessus de
        <a href="" onMouseOver="texte()">cela</a>...
    </p>
</body>
</html>
```

# Javascript dans les documents HTML

## [1]

```
<!DOCTYPE html>
<head>
  <title>Exemple de page XHTML contenant du Javascript</title>
  <meta charset="UTF-8"/>
  <script type="text/javascript">
    function fenetre() { alert('Message d\'alerte dans une fonction.');?>
  </script>
</head>
  <body onload="alert('Message d\'alerte généré à la fin du chargement.')">
  <script type="text/javascript">
    alert('Message d\'alerte dans le corps du document.');
  </script>
  <p>
    Ceci est le corps du document.
    <a href="javascript:fenetre()">Message d\'alerte</a>.
  </p>
</body>
</html>
```

# Utilité de Javascript

- Vérification des saisies dans un formulaire (e-mail, code postal, date...)
- Calculs suite à la saisie dans un formulaire
- Gestion des dates et des heures
- Gestion des cookies
- Menus dynamiques
- Animation graphiques (survol d'image ou de texte, bannières animées)

SHAKE THE FUTURE.



# Utilité de Javascript [2]

- Dans un navigateur, permet :
  - de spécifier des changements sur le document :
    - après son chargement,
    - au cours de sa vie dans la fenêtre du navigateur,
    - sur le contenu, la structure, le style
  - en les planifiant à l'avance
  - en interceptant des évènements (souris, clavier, doigts...)

# Utilité de Javascript [2]

- Mais également (API HTML5) :
  - d'échanger avec un serveur (AJAX)
  - de dessiner (canvas - bitmap - ou svg – vectoriel)
  - de se géolocaliser
  - d'enregistrer localement du contenu (cache ou bdd) de
  - jouer des fichiers audio ou vidéo
  - etc...

# Plan

- Généralités
- Les bases
- Les objets
- Les évènements
- Les boites de dialogue

SHAKE THE FUTURE.



# Syntaxe de JavaScript [extrait de 5]

- Instructions :
  - Comme beaucoup de langages de programmation, les instruction de JavaScript sont délimitées par des « ; »

```
instruction_1;  
instruction_2;  
instruction_3;
```
  - JavaScript n'est pas sensible aux espaces :

```
instruction_1;  
    instruction_1_1;  
    instruction_1_2;  
instruction_2;  instruction_3;
```

# Syntaxe de JavaScript [extrait de 5]

- Commentaires :
  - il peut être utile de **laisser des annotations** faites par le développeur pour expliquer le fonctionnement :
    - d'un script
    - d'une instruction
    - ou même d'un groupe d'instructions
  - Un commentaire peut permettre également **de ne pas faire exécuter** une (ou plusieurs) instruction(s)
  - Les commentaires ne gênent pas l'exécution d'un script
  - Il existe 2 types de commentaires :
    - Les commentaires de fin de ligne : servent à commenter une instruction
    - Les commentaires multilignes : permet de mettre des commentaires sur plusieurs lignes

# Syntaxe de JavaScript [extrait de 5]

- Commentaires :
  - le texte placé en commentaire est ignoré lors de l'exécution du script
  - Exemples :

```
instruction_1; // Ceci est ma première instruction  
instruction_2;  
// La troisième instruction ci-dessous :  
instruction_3;
```

```
instruction_1; // Ceci est ma première instruction  
instruction_2;  
// La troisième instruction ci-dessous pose problème, je l'annule temporairement  
// instruction_3;
```

# Syntaxe de JavaScript [extrait de 5]

- Commentaires :
  - le texte placé en commentaire est ignoré lors de l'exécution du script
  - Exemples :

```
/* Ce script comporte 3 instructions :  
 - Instruction 1 qui fait telle chose  
 - Instruction 2 qui fait autre chose  
 - Instruction 3 qui termine le script  
 */  
instruction_1;  
instruction_2;  
instruction_3; // Fin du script
```

```
/* Un commentaire multilignes aussi */  
/*instruction_1;  
instruction_2;*/  
/* Les deux instructions ci-dessus  
ne seront pas exécutées */  
instruction_3;
```

# Les bases [2]

- 3 instructions pour démarrer
  - Afficher une boîte sommaire avec un message

```
alert("Bonjour!");
```
  - Écrire du texte dans la console (pour débuguer)

```
console.log("Texte d'essai");
```
  - Écrire quelque chose dans le document :
    - dans une balise HTML
    - qui a l'id "monId"
    - `document.getElementById("monId").innerHTML = "<p>Mon contenu</p>";`
    - Ici on a modifié le contenu de l'élément HTML dont l'id est "monId"

# Les Bases [2]

```
<!DOCTYPE html>
<head>
  <title>Exemples de base Javascript</title>
  <meta charset="UTF-8"/>
  <script src="js/exemple01.js"></script>
</head>
<body>

  <p>Hello <b id='boldStuff'>world</b> </p>
  <input type='button' onclick='changeText()' value='Change Text'/>

</body>
</html>
```

SHAKE THE FUTURE.



# Les Bases [2]

## Fichier exemple01.js

```
alert("Bonjour !");  
console.log("Texte d'essai");  
  
/* Un commentaire */  
function changeText(){  
    document.getElementById('boldStuff').innerHTML = "à tous !";  
}
```

SHAKE THE FUTURE.



# Les Bases

Pour écrire dans la console ou pourra utiliser :

```
window.console.log("Hello world");
// ou
global.console.log("Hello world");
```

Ou directement :

```
console.log("Hello world");
```

Qui sera compatible dans tous les environnements.

SHAKE THE FUTURE.



# Notion de fonctions [2]

- Un exemple de fonction JavaScript :
  - Une fonction est définie soit :
    - Dans l'entête (balise `<head>`) du document HTML
    - Dans un fichier JavaScript (extension `.js`)
  - Cette fonction aura 2 paramètres d'entrées : **id** et **message**
- Puis 2 exemples d'utilisation de cette fonction :
  - Une utilisation : dans un fichier HTML

```
// déclaration de la fonction
function afficher(id, message) {
    console.log("Message: " + message);
    document.getElementById(id).innerHTML = message;
}
// deux exemples d'appel
SHAKETH afficher("special1", "<p>Du contenu !</p>");
afficher("special2", "Un autre contenu.");
```

# Notion de fonctions [2]

```
// déclaration de la fonction
function afficher(id, message) {
    console.log("Message: " + message);
    document.getElementById(id).innerHTML = message;
}
// deux exemples d'appel
afficher("special1", "<p>Du contenu !</p>");
afficher("special2", "Un autre contenu.");
```

- Un ensemble d'instructions prêt à être utilisé après sa déclaration.
  - Permet la ré-utilisabilité du code
  - Deux temps : la déclaration, puis l'appel
  - Peut avoir des paramètres (ici **id** et **message**)

# Notion de fonctions [2]

```
<!DOCTYPE html>
<head>
    <title>Exemples de base Javascript</title>
    <meta charset="UTF-8"/>
    <script>
        function afficher(id, message) {
            console.log("Message: " + message);
            document.getElementById(id).innerHTML = message;
        }
    </script>
</head>
<body>
<div>
<a id="special1" onclick='afficher("special1", "<p>Du contenu !</p>");'> test1</a>
<a id="special2" onclick='afficher("special2", "Un autre contenu.");'> test2</a>
</div>
</body>
</html>
```

# Notion de fonctions [2]

- Peut retourner une valeur avec *return*

// déclaration d'une autre fonction

```
function perimetre(longueur, largeur) {  
    return 2 * (longueur + largeur);  
}
```

// appel imbriqué des deux fonctions précédentes

```
afficher("special3", "Périmètre du rectangle : " + perimetre(100, 40));
```

Remarque : passage des paramètres par valeur

# Notion de fonctions [2]

- Peut être déclarée et appelée du même coup.

Dans ce cas, le nom est souvent omis.

Penser aux parenthèses.

PAS RECOMMENDÉ

*// déclaration de la fonction suivi de l'appel*

```
(function(id, message) {  
    console.log("Message: " + message);  
    document.getElementById(id).innerHTML = message; } ("special4", "Un dernier contenu."));
```

SHAKE THE FUTURE.



# Notion de fonctions [2]

```
<!DOCTYPE html>
<head>
<title>Exemples de base Javascript</title>
<meta charset="UTF-8"/>
</head>
<body>
<div>
<p id="special4" > test</p>
<script>
    (function(id, message) {
        console.log("Message: " + message);
        document.getElementById(id).innerHTML = message;
    }("special4", "Un dernier contenu."));
</script>
</div>
</body>
</html>
```

# Notion de fonctions [1]

- Les arguments

```
function somme() {  
    var argv = somme.arguments;  
    var argc = somme.arguments.length;  
    var result = 0;  
    for (var i = 0 ; i < argc ; i++) {  
        result += argv[i];  
    }  
    return result;  
}
```

somme(1,2,3) retourne 6 et somme(2) retourne 2

SHAKE THE FUTURE.



# Notion de fonctions [1]

```
<!DOCTYPE html>
<head>
    <title>Exemples de base Javascript</title>
    <meta charset="UTF-8"/>
    <script>
        function somme() {
            var argv = somme.arguments;
            var argc = somme.arguments.length;
            var result = 0;
            for (var i = 0 ; i < argc ; i++) {
                result += argv[i];
            }
            return result;
        }
    </script>
</head>
<body>
    <!-- Appel de la fonction créée -->
    <p onclick="document.getElementById('somme1').innerHTML=somme(1,2,3);"> somme(1,2,3) :
        <div id='somme1'></div></p>
    <p onclick="document.getElementById('somme2').innerHTML=somme(2);"> somme(2) :
        <div id='somme2'></div></p>
</body>
</html>
```

# Notion de fonctions - Exemple d'affichage de date et heure

```
function afficheDate(){
    // Tableau contenant le nom des jours, indexe; de 0 (dimanche) a 6 (samedi)
    var Jour = new Array("dimanche","lundi","mardi","mercredi","jeudi","vendredi","samedi");
    // Tableau contenant le nom des mois, indexe de 0 (janvier) a 11 (decembre)
    var Mois = new
        Array("janvier","f&eacute;vrier","mars","avril","mai","juin","juillet","ao&ucirc;t","septembre","octobre" );
    var d = new Date(); // Cette fonction affecte d de tous les champs relatifs & l'instant present
    // les fonctions getDay() et getMonth() renvoient un entier permettant d'indexer les tableaux
    Jour et Mois
    document.write(Jour[d.getDay()]+" "+d.getDate()+" "+Mois[d.getMonth()]+" ");
    // Prise en compte de l'interpretation differente de getYear() entre Netscape et Internet
    Explorer
    document.write ((navigator.appName == "Netscape") ? 1900+d.getYear() : d.getYear());
    // L'ecriture de l'heure comporte forcement 2 chiffres, le 1er eventuellement 0
    var H = d.getHours();
    document.write(" &grave; " + ((H < 10) ? "0" : "") + H + " heure" + ((H > 1) ? "s" : ""));
    var M = d.getMinutes();
    document.write(((M < 10) ? "0" :"") + M + " minute" + ((M > 1) ? "s" : ""));
    var S = d.getSeconds();
    document.write("et " + ((S < 10) ? "0" :"") + S + " seconde" + ((S > 1) ? "s" : " "));
}
```

# Notion de fonctions - Exemple d'affichage de date et heure

```
<!DOCTYPE html>
<head>
<title>
Affichage de la date complète
</title>
<meta charset="UTF-8"/>
<script src="js/date.js"></script>
</head>
<body>
<p>
<script> afficheDate();</script>
</p>
</body>
</html>
```

SHAKE THE FUTURE.



# Notion de fonctions - Exemple d'affichage de date et heure

- Instruction With

Permet d'utiliser plusieurs fois de suite un objet complexe  
(ex : document)

# Notion de fonctions - Exemple d'affichage de date et heure

```
function afficheDate(){
    // Tableau contenant le nom des jours, indexe; de 0 (dimanche) a 6 (samedi)
    var Jour = new Array("dimanche","lundi","mardi","mercredi","jeudi","vendredi","samedi");
    // Tableau contenant le nom des mois, indexe de 0 (janvier) a 11 (decembre)
    var Mois = new
        Array("janvier","f&eacute;vrier","mars","avril","mai","juin","juillet","ao&ucirc;t","septembre","octobre" );
    var d = new Date(); // Cette fonction affecte d de tous les champs relatifs & l'instant present
    // les fonctions getDay() et getMonth() renvoient un entier permettant d'indexer les tableaux
    Jour et Mois
    document.write(Jour[d.getDay()]+" "+d.getDate()+" "+Mois[d.getMonth()]+" ");
    // Prise en compte de l'interpretation differente de getYear() entre Netscape et Internet
    Explorer
    document.write ((navigator.appName == "Netscape") ? 1900+d.getYear() : d.getYear());
    // L'ecriture de l'heure comporte forcement 2 chiffres, le 1er eventuellement 0
    var H = d.getHours();
    document.write(" &grave; " + ((H < 10) ? "0" : "") + H + " heure" + ((H > 1) ? "s" : ""));
    var M = d.getMinutes();
    document.write((M < 10) ? "0" : "") + M + " minute" + ((M > 1) ? "s" : " ");
    var S = d.getSeconds();

    document.write("et " + ((S < 10) ? "0" : "") + S + " seconde" + ((S > 1) ? "s" : ""));
}
```

# Notion de fonctions - Exemple d'affichage de date et heure

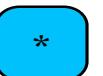
```
function afficheDate(){
    // Tableau contenant le nom des jours, indexe; de 0 (dimanche) a 6 (samedi)
    var Jour = new Array("dimanche","lundi","mardi","mercredi","jeudi","vendredi","samedi");
    // Tableau contenant le nom des mois, indexe de 0 (janvier) a 11 (decembre)
    var Mois = new
        Array("janvier","f&eacute;vrier","mars","avril","mai","juin","juillet","ao&ucirc;t","septembre","octobre" );
    var d = new Date(); // Cette fonction affecte d de tous les champs relatifs à l'instant present
    with(document){
        // les fonctions getDay() et getMonth() renvoient un entier permettant d'indexer les tableaux
        Jour et Mois
        write(Jour[d.getDay()]+ " "+d.getDate()+" "+Mois[d.getMonth()]);
        // Prise en compte de l'interpretation differente de getYear() entre Netscape et Internet
        Explorer
        write ((navigator.appName == "Netscape") ? 1900+d.getYear() : d.getYear());
        // L'ecriture de l'heure comporte forcement 2 chiffres, le 1er eventuellement 0
        var H = d.getHours();
        write(" &grave; " + ((H < 10) ? "0" : "") + H + " heure" + ((H > 1) ? "s " : " "));
        var M = d.getMinutes();
        write(((M < 10) ? "0 ":"") + M + " minute" + ((M > 1) ? "s " : " "));
        var S = d.getSeconds();
        write("et " + ((S < 10) ? "0 ":"") + S + " seconde" + ((S > 1) ? "s " : " "));
    }
}
```

SHAKE THE FUTURE.

# Opérateurs [2]



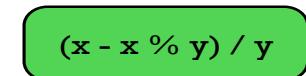
**Addition de nombres et concaténation de chaînes**



Division  
flottante



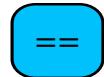
Reste de la  
division entière



Division  
entière



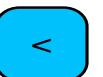
**Affectation de valeur à une variable**



**Comparaison large (conversion à la volée)**



**Comparaison stricte (sans conversion de type)**



**Si  $x == \text{true}$ , vaut  $y$  sinon vaut  $z$**

# Expressions [1]

- arithmétique

$(3+4) * (56.7 / 89)$

- chaîne de caractères

"L' étoile" + " " + "filante"

- logique

`temp == 37`

`h2o = (temp<100) ? "eau" : "vapeur";`

`h2o = (temp>0) ? (`

`(temp<100) ? "eau" : "vapeur") :`  
 `"glace");`

# Structures de contrôle[2]

`if(expr) { ... }`

`if(expr) { ... } else { ... }`

`if(expr) { ... } else if { ... } else if { ... } ... else { ... }`

`while(expr) { ... }`

`do { ... } while(expr)`

`switch(expr) { case value1 : ... case value2 : ... default : ... }`

`break`

`for(ini ; cond ; iter) { ... }`

SHAKE THE FUTURE.



# Variables [2]

```
// déclaration d'une variable  
var monMessage = "<p>Du contenu !</p>";
```

```
// utilisation avec la variable précédente  
afficher("bloc1", monMessage);  
afficher("bloc2", monMessage);
```

- Ici : monMessage contient le texte « <p>Du contenu !</p> »
- On la déclare avec **var** et on peut la réutiliser partout après.

# Types [extrait de 5]

- Attention !
- JavaScript est ce qu'on appelle un langage faiblement typé
- C'est à dire qu'une variable n'est pas déclarée avec son type
- On peut donc écrire :

```
var maVar = 2;  
maVar = "<font color=red>Hello</font>";  
maVar = 3.14159;  
maVar = function(id, message)  
{  
    document.getElementById(id).innerHTML = message;  
}
```

# Types [2]

```
// déclaration de trois variables
var monMessage = "<p>Du contenu!</p>";
var monNombre = 17.2;
var maFonction = function(id, message)
{
    console.log("Message: " + message);    document.getElementById(id).innerHTML =
    message;
}
// utilisation
maFonction("bloc1", monMessage);
```

SHAKE THE FUTURE.



# Types [2]

- Dans une variable, on peut stocker

- Un booléen (true ou false)
- Une chaîne de caractères : "chaine" ou 'chaine'
- Un nombre : entier ou réel
- Une fonction

# Types [2]

- Types simples :
  - Booléen
  - Chaine de caractères
  - Nombre
- Conversion en type simple
  - parseInt, parseFloat
  - String

```
var x = 17;  
var y = 18.2;  
var c = "bonjour";  
var d = "15.25";  
  
alert(x + parseInt(d));  
alert(x + parseFloat(d));  
alert(string(x) + c);
```

```
// Résultats :  
// 32  
// 32.25  
// 17bonjour
```

# Types

- Déclaration de types

- `var length =16;` // Number
- `var lastName = "Johnson";` // String
- `var cars = ["Saab", "Volvo", "BMW"];` // Array
- `var x = {firstName:"John", lastName:"Doe"};` // Object

- Les types sont dynamiques

- `var x;` // x est indéfini
- `var x = 5;` // x est un nombre entier
- `var x = "John";` // x est une chaîne de caractères
- `var x = 123e-5;` // x est un réel 0.00123

- Obtention du type : typeOf

- `typeof "John"` // Returns string
- `typeof 3.14` // Returns number
- `typeof false` // Returns boolean
- `typeof [1,2,3,4]` // Returns object
- `typeof {name:'John', age:34}` // Returns object

# Types

```
var maVariable1 = 0;           // lie "maVariable1" à une donnée de valeur 0
var maVariable2 = maVariable1; // lie "maVariable2" à la donnée liée à "maVariable1"
maVariable1++;                // équivalent à "maVariable1 = maVariable1 + 1;", relie "maVariable1" à une nouvelle donnée de
                             // valeur maVariable1 + 1 (affectation)
alert(maVariable1);          // affiche 1
alert(maVariable2);          // affiche 0

var maVariable3 = [1, 2, 3];   // lie "maVariable3" à une donnée de valeur [1, 2, 3]
var maVariable4 = maVariable3; // lie "maVariable4" à la donnée liée à "maVariable3"
maVariable3 = [4, 5, 6];      // relie "maVariable3" à une nouvelle donnée de valeur [4, 5, 6] (affectation)
alert(maVariable3);          // affiche [4, 5, 6]
alert(maVariable4);          // affiche [1, 2, 3]

var maVariable5 = [1, 2, 3];   // lie "maVariable5" à une donnée de valeur [1, 2, 3]
var maVariable6 = maVariable5; // lie "maVariable6" à la donnée liée à "maVariable5"
maVariable5.push(4);         // modifie la donnée liée à "maVariable5" et "maVariable6"
alert(maVariable5);          // affiche [1, 2, 3, 4]
alert(maVariable6);          // affiche [1, 2, 3, 4]
```

# Types

```
<!DOCTYPE html>
<head>
  <title>Exemples Object</title>
  <meta charset="UTF-8"/>
</head>
<body>
<p id="demo"></p>
<script>
var person = {
  firstName : "John",
  lastName : "Doe",
  age : 50,
  eyeColor : "blue"
};
document.getElementById("demo").innerHTML =
person.firstName + " is " + person.age + " years old.";
</script>
</body>
</html>
```

SHAKE THE FUTURE.



# Types [2]

- Puisque « fonction » est un type de données en JavaScript, on peut :
  - le passer en paramètre d'une fonction
  - le renvoyer comme valeur d'une fonction

```
function afficher(message) {
    console.log("Message: " + message);
    document.getElementById("info").innerHTML = message;
}
function calculerEtAfficher(cote, affichage) {
    var resultat = 4 * cote;
    affichage("Périmètre du carré : " + resultat);
}
calculerEtAfficher(10, afficher);
```

PRWEB

Ici, la fonction « afficher » est un *callback* :  
elle est passée en paramètre d'une autre fonction, au même titre que  
n'importe quel autre paramètre.

SHAKET



# Types [2]

```
<!DOCTYPE html>  
<head>  
  <title>Exemples de base Javascript</title>  
  <meta charset="UTF-8"/>  
  <script src="js/exemple03.js"></script>  
</head>  
<body>  
  <p onclick='calculerEtAfficher(10, afficher);'>  
    Calcul du périmètre : <b id='info'></b>  
  </p>  
</body>  
</html>
```

SHAKE THE FUTURE.



# Types [2]

```
function creeAffichage(id) {
    return function(message) {
        console.log("Message: " + message);
        document.getElementById(id).innerHTML = message;
    }
}
creeAffichage("info1")("Du contenu tout frais !");
calculerEtAfficher(10, creeAffichage("info2"));
```

Ici, « `creeAffichage` » est une « usine à fonctions » :  
elle crée une fonction à la demande, que l'on peut stocker, réutiliser,  
passer en paramètre, etc.

SHAKI



# Types [2]

exemple04.js :

```
function calculerEtAfficher(cote, affichage)
{
    var resultat = 4 * cote;
    affichage("Périmètre du carré : " + resultat);
}

function creeAffichage(id) {
    return function(message) {
        console.log("Message: " + message);
        document.getElementById(id).innerHTML = message;
    }
}
```

SHAKE THE FUTURE.



# Types [2]

```
<!DOCTYPE html>
<head>
<title>Exemples de base Javascript</title>
<meta charset="UTF-8"/>
<script src="js/exemple04.js"></script>
</head>
<body>
<script>
    // Création de la fonction
    var monAffichage = creeAffichage("special1");
</script>
<!-- Appel de la fonction créée -->
<p id="special1" onclick='monAffichage("Du contenu tout frais!");'> Du contenu </p>
<!-- Création et appel en une seule ligne -->
<p id="special2" onclick='creeAffichage("special2")("Du contenu tout frais!");'> truc</p>
<!-- Création et passage en paramètre en une seule ligne -->
<p id="special3" onclick='calculerEtAfficher(10, creeAffichage("special3"));'> truc2</p>
</body>
</html>
```

SHAKE THE FUTURE.



# Visibilité et portée des variables [2]

- Une fois qu'une variable est déclarée, elle n'est pas nécessairement visible (utilisable) en tous les endroits du script.

```
var globale1 = 17;  
globale2 = 18;  
  
function maFonction() {  
    globale3 = 19;  
    var locale = 20;  
    alert(globale1 + globale2 + globale3 + locale); // ok  
}  
  
alert(globale3); // erreur  
maFonction();  
alert(globale3); // ok
```

## Règle 1

Une variable déclarée à l'extérieur d'une fonction ou sans le mot-clé **var** est dite **globale**, et visible partout après sa définition.

SHAK



# Visibilité et portée des variables [2]

```
// déclaration d'une fonction
function maFonction()
{
    var monMessage = "<p>Du contenu bien frais !</p>";
    document.getElementById(id).innerHTML = monMessage;
}

maFonction(); // ok : exécute la fonction

alert(monMessage); // undefined : la variable monMessage
                    // n'est pas accessible ici
```

## Règle 2

Une variable non globale est dite **locale**.

Elle n'est accessible que dans la fonction où elle est définie...

SHAKE THE



# Visibilité et portée des variables [2]

```
// déclaration d'une fonction
function afficheur(id)
{
    var monMessage = "<p>Du contenu bien frais !</p>";
    return function() {
        document.getElementById(id).innerHTML = monMessage;
    }
}

var disp = afficheur("bloc");
disp(); // ok
```

## Règle 2

...y compris dans les fonctions créées dans le contexte.

SHAKE THE FUTURE.



# Visibilité et portée des variables

permet de déclarer une variable dont la portée est celle du bloc

permet de créer une constante nommée accessible uniquement en lecture

```
// 1. Déclaration dans un bloc

if (true) {          // début du bloc
    var maVariable1; // déclaration de la variable
    let maVariable2; // déclaration de la variable
    const maVariable3; // déclaration de la variable
                      // fin du bloc mais pas de la portée de maVariable1

    alert(maVariable1); // ne soulève pas d'erreur
    alert(maVariable2); // erreur : la variable est hors de sa portée
    alert(maVariable3); // erreur : la variable est hors de sa portée

// 2. Déclaration dans une fonction

function maFunction() { // début de la fonction
    var maVariable4; // déclaration de la variable
    let maVariable5; // déclaration de la variable
    const maVariable6; // déclaration de la variable
}
                      // fin de la fonction et de la portée des variables

alert(maVariable4); // erreur : la variable est hors de sa portée
alert(maVariable5); // erreur : la variable est hors de sa portée
alert(maVariable6); // erreur : la variable est hors de sa portée
```

# Visibilité et portée des variables - masquage

```
var maVariable1 = 0; // définition de la variable parente

// 1. Affectation

function maFonction1() { // fonction enfant
    maVariable1 = 1;      // affectation de la variable parente
}

alert(maVariable1);      // affiche 0
maFonction1();           // affecte la variable parente
alert(maVariable1);      // affiche 1

// 2. Masquage

var maVariable2 = 0;      // définition de la variable parente

function maFonction2() { // fonction enfant
    var maVariable2;     // déclaration de la variable enfant masquant la variable parente
    maVariable2 = 1;      // affectation de la variable enfant
}

alert(maVariable2);      // affiche 0
maFonction2();
alert(maVariable2);      // affiche 0
```

# Visibilité et portée des variables - déclaration

- Cas de **var** pré-initialisée à *undefined*

```
alert(maVariable); // affiche undefined  
var maVariable = 0;  
alert(maVariable); // affiche 0
```

# Visibilité et portée des variables - déclaration

- Cas de **let** ou **const** non pré-initialisées donc inaccessibles avant leur déclaration

```
// 1. Avec initialiseur

alert(maVariable1); // erreur : accès impossible avant l'initialisation
alert(maVariable2); // erreur : accès impossible avant l'initialisation
let maVariable1 = 5;
const maVariable2 = 8;
alert(maVariable1); // affiche 5
alert(maVariable2); // affiche 8

// 2. Sans initialiseur

alert(maVariable3); // erreur : accès impossible avant l'initialisation
alert(maVariable4); // erreur : accès impossible avant l'initialisation
let maVariable3;
const maVariable4; // erreur : initialisation manquante
alert(maVariable3); // affiche undefined
alert(maVariable4); // erreur : initialisation manquante
```

- JavaScript autorise la redéclaration de la même variable dans sa portée lexicale, mais uniquement avec le mot-clé **var**

```
var maVariable = 2;
var maVariable = 9;
```

SHAKE THE FUTURE.



# Notions de visibilité : fermetures [2]

```
// déclaration d'une fonction
function afficheur(id)
{
    var monMessage = "<p>Du contenu bien frais !</p>";
    return function() {
        document.getElementById(id).innerHTML = monMessage;
    }
}
```

## Règle 3

On appelle **fermeture** d'une fonction (closure) l'ensemble des variables qui lui sont visibles au moment de sa déclaration, globales et locales.

La fermeture d'une fonction F est visible quel que soit le moment de l'appel de F, même si l'appel a lieu en dehors du contexte de déclaration de F.

# Notions de visibilité : fermetures [2]

```
// déclaration d'une fonction
function afficheur(id)
{
    var monMessage = "<p>Du contenu bien frais !</p>";
    return function() {
        document.getElementById(id).innerHTML = monMessage;
    }
}
```

Ici la fermeture de la fonction interne (anonyme) est { id, monMessage }.  
La fonction est créée à l'intérieur de **afficheur**, mais elle sera appelée à l'extérieur. Sa fermeture sera alors quand même visible.

# Notions de visibilité : fermetures [2]

- Exemple

```
function externe()
{
    var a = 7;
    var interne = function()
    {
        alert(a);
    }
    a = 8;
    return interne;
}

externe()();
```

# Notions de visibilité : fermetures [2]

Réponse : 8.

La fermeture, c'est l'environnement des variables qui existent au moment de la création de la fonction.

Leurs valeurs sont les dernières valeurs connues au moment de l'appel.

# Notions de visibilité : fermetures

- Exemple

```
var compteur = (function () {
    var i = 0; // propriété privée

    return { // méthodes publiques
        obtenir: function () {
            alert(i);
        },
        mettre: function (valeur) {
            i = valeur;
        },
        incrementer: function () {
            alert(++i);
        }
    };
})(); // module

compteur.obtenir(); // affiche 0
compteur.mettre(6);
compteur.incrementer(); // affiche 7
compteur.incrementer(); // affiche 8
compteur.incrementer(); // affiche 9
```

# Plan

- Généralités
- Les bases
- **Les objets**
- Les évènements
- Les boîtes de dialogue

SHAKE THE FUTURE.



# Objets [2]

- JavaScript supporte la notion d'objet :
  - objet = attributs + méthodes
  - voiture = { marque, modèle } + { accélérer, freiner }
- En première approche, il n'y a pas vraiment de distinction entre attributs et méthodes :
  - une méthode est un attribut de type fonction
  - on dispose de la variable **this**

# Objets – constructeur [2]

```
function Voiture(marque, modele) {  
    this.marque = marque;  
    this.modele = modele;  
    this.afficher = function() {  
        alert(this.marque + " " + this.modele);  
    }  
}  
  
var maVoiture = new Voiture("Ford", "Fiesta");  
maVoiture.afficher();  
console.log(maVoiture.marque);
```

Toute fonction appelée avec new est un constructeur d'objet et peut donc utiliser this.

SHA On peut utiliser this dans toutes les fonctions. En dehors d'un contexte d'objet, this représente l'espace global.

# Objets [2]

```
<!DOCTYPE html>
<head>
    <title>Exemples Object</title>
    <meta charset="UTF-8"/>
</head>
<body>
    <p id="demo"></p>
    <script>
        function Voiture(marque, modele) {
            this.marque = marque;
            this.modele = modele;
            this.afficher = function() {
                return (this.marque + " " + this.modele);
            }
        }
        var maVoiture = new Voiture("Ford", "Fiesta");
        document.getElementById("demo").innerHTML = maVoiture.afficher();
        console.log(maVoiture.marque);
    </script>
</body>
</html>
```

# Objets – accès aux membres [2]

```
// Accès aux attributs et méthodes  
console.log(maVoiture.marque);  
maVoiture.afficher();  
  
// Ajout de membre a posteriori  
maVoiture.km = 8000;  
maVoiture.rouler = function(distance) {  
    this.km += distance;  
}  
  
// Utilisation des crochets  
alert( maVoiture["km"] );
```

- Tous les membres d'un objet sont publics.
- On peut ajouter des membres a posteriori.
- Utilisation de l'opérateur . (point) et des [] (crochets).
- On peut mettre une variable entre les crochets (de type chaîne de caractères...)



# Objets – accès aux membres [2]

```
<!DOCTYPE html>
<head>
    <title>Exemples Object</title>
    <meta charset="UTF-8"/>
    <script src="js/voiture.js"></script>
</head>
<body>
<script>
var maVoiture = new Voiture('Ford', 'Fiesta');
</script>
<p id="demo" onclick="
document.getElementById('demo').innerHTML = maVoiture.afficher();
console.log(maVoiture.marque);">Ma voiture</p>
<script>
    // Ajout de membre a posteriori
    maVoiture.km = 8000;
    maVoiture.rouler = function(distance) {
        this.km += distance;
    }
</script>
<p><button onclick="alert(maVoiture['km']); >>kilom&acute;trage</button></p>
</body>
</html>
```

SHAKE THE FUTURE.



# Objets – Format JSON [2]

- **JSON** (*JavaScript Object Notation*) est un format de données textuelles dérivé de la notation des objets du langage JavaScript. Il permet de représenter de l'information structurée comme le permet XML par exemple.
- Un document JSON a pour fonction de représenter de l'information accompagnée d'étiquettes permettant d'en interpréter les divers éléments, sans aucune restriction sur le nombre de celles-ci.
- Un document JSON ne comprend que deux types d'éléments structurels :
  - des ensembles de paires nom / valeur ;
  - des listes ordonnées de valeurs.
- Ces mêmes éléments représentent trois types de données :
  - des objets ;
  - des tableaux ;
  - des valeurs génériques de type tableau, objet, booléen, nombre, chaîne ou null.

SHAKE THE FUTURE.



Source : wikipédia

# Objets – Format JSON [2]

- Exemple en utilisant la syntaxe JSON

```
// déclaration d'un objet selon la syntaxe JSON
var monObjet = {
    "nom" : "Fred",
    "age" : 28,
    "afficher" : function(id) {
        document.getElementById(id).innerHTML = this.nom;
    }
};
```

```
// utilisation de l'attribut nom
alert(monObjet.nom);
// utilisation de la méthode afficher
monObjet.afficher("bloc");
```

~couple clés/valeur

# Objets – Format JSON [2]

```
<!DOCTYPE html>
<head>
<title>Exemples Object</title>
<meta charset="UTF-8"/>
<script>
// déclaration d'un objet selon la syntaxe JSON
var monObjet = {
    "nom" : "Fred",
    "age" : 28,
    "afficher" : function(id) {
        document.getElementById(id).innerHTML = this.nom+", "
        +this.age+" ans.";
    }
};
</script>
</head>
<body>
<p id="demo" onclick="monObjet.afficher('demo');>Qui suis-je ?</p>
</body>
</html>
```

SHAKE THE FUTURE.



# Objets – Prototype [2]

```
function affichageObjet(id) {  
    document.getElementById(id).innerHTML = this.nom;  
}  
  
var monObjet1 = {  
    "nom" : "Fred",  
    "afficher" : affichageObjet  
};  
  
var monObjet2 = {  
    "nom" : "Paul",  
    "afficher" : affichageObjet  
};
```

Ici les méthodes sont dupliquées en mémoire.

Lorsque plusieurs objets ayant la même structure doivent être construits,  
on préfère utiliser la notion de **prototype**.

# Objets – Prototype [2]

```
function Voiture(marque, modele) {  
    this.marque = marque;  
    this.modele = modele;  
}  
  
Voiture.prototype.afficher = function() {  
    alert(this.marque + " " + this.modele);  
}  
  
var maVoiture1 = new Voiture("Ford", "Fiesta");  
var maVoiture2 = new Voiture("Renault", "Espace");  
maVoiture1.afficher();  
  
ma  
SH Un constructeur C peut être associé à un prototype P, qui est un  
SH objet «modèle». Tous les objets construits avec C possèdent alors les  
SH membres de P.
```

# Objets – Prototype [2]

```
<!DOCTYPE html>
<head>
  <title>Exemples Object</title>
  <meta charset="UTF-8"/>
  <script src="js/voiture2.js"></script>
</head>
<body>
<script>
var maVoiture1 = new Voiture("Ford", "Fiesta");
var maVoiture2 = new Voiture("Renault", "Espace");
</script>

<p><button onclick="maVoiture1.afficher();">Voiture 1</button></p>
<p><button onclick="maVoiture2.afficher();">Voiture 2</button></p>

</body>
</html>
```

# Objets – l'héritage [2]

- On peut simuler un héritage en JavaScript :
  - Le prototype est un objet « modèle »
  - Donc le prototype du fils doit être une instance du père, à laquelle on ajoute des membres spécifiques
  - Et le constructeur du fils doit appeler celui du père
- Opérateur instanceof :

```
console.log( maVoiture instanceof Voiture ); // true
```

# Objets – l'héritage [2]

```
// Voiture hérite de Produit
function Produit(nom, prix) {
    this.nom = nom;
    this.prix = prix;
}
Produit.prototype.afficher = function() {
    console.log(this.nom + " coûte " + this.prix + " euros");
}
function Voiture(marque, modele, prix) {
    // Appel du constructeur de produit
    Produit.call(this, marque + " " + modele, prix);

    this.marque = marque;
    this.modele = modele;
}
// Le prototype est un objet "modele", ici mon modèle est Produit
Voiture.prototype = new Produit();
// D'autres méthodes spécifiques à la voiture
Voiture.prototype.rouler = function() {
    console.log(this.nom + " roule...");
}
```

SHAKE THE FUTURE.



# Objets – l'héritage [2]

```
var monProduit = new Produit("Trombone", 0.3);
var maVoiture = new Voiture("Ford", "Fiesta", 12000);

console.log(monProduit instanceof Produit);
console.log(monProduit instanceof Voiture);
console.log(maVoiture instanceof Produit);
console.log(maVoiture instanceof Voiture);

maVoiture.afficher();
maVoiture.rouler();

monProduit.afficher();
console.log(monProduit.marque); // undefined
monProduit.rouler(); // erreur
```

SHAKE THE FUTURE.



# Objets [2]

- Structure de contrôle for...in

```
for( var prop in maVoiture ) {  
    console.log( prop + ": " + maVoiture[prop] );  
}
```

- Pour itérer sur chacun des membres d'un objet :
  - L'ordre de l'itération est **non fiable**
  - Ne pas ajouter de membres à l'objet pendant l'itération
  - Passe en revue les attributs et les méthodes
  - Avec *hasOwnProperty* il est possible de distinguer les membres «propres» de ceux qui viennent du prototype.

# Objets [2]

```
function Voiture(marque, modele) {  
    this.marque = marque;  
    this.modele = modele;  
}  
Voiture.prototype.afficher = function() {  
    alert(this.marque + " " + this.modele);  
}  
var maVoiture = new Voiture("Ford", "Fiesta");  
console.log("--- Test 1 ---");  
for( var prop in maVoiture ) {  
    console.log( prop + ":" + maVoiture[prop] );  
}  
console.log("--- Test 2 ---");  
for( var prop in maVoiture ) {  
    if( maVoiture.hasOwnProperty(prop) ) {  
        console.log( prop + ":" + maVoiture[prop] );  
    }  
}
```

SHAKE THE FUTURE.



# Objets - Tableaux [2]

```
var monTableau = new Array();  
  
monTableau[0] = "Toto";  
monTableau[1] = 17;  
monTableau[2] = true;  
monTableau[4] = "hello";  
  
console.log(monTableau.length + " éléments :");  
for(var i = 0; i < monTableau.length; i++) {  
    console.log(monTableau[i]);  
}  
  
// Ici monTableau[3] == undefined
```

## Java Script propose un constructeur de tableaux

SHAKE THE I

- Utilisation des crochets pour l'accès
- Propriété **length** pour avoir le nombre d'éléments

# Objets - Tableaux [2]

- Tableau au format JSON
  - Syntaxe raccourcie pour la déclaration d'un tableau
  - Utilisation des crochets pour la déclaration (au lieu des accolades pour les objets)

```
// déclaration d'un tableau selon la syntaxe JSON
var monTableau = [
    "Fred",
    "Gina",
    "Mehdi"
];

// les éléments sont numérotés à partir de zéro
alert(monTableau[0]); // Fred

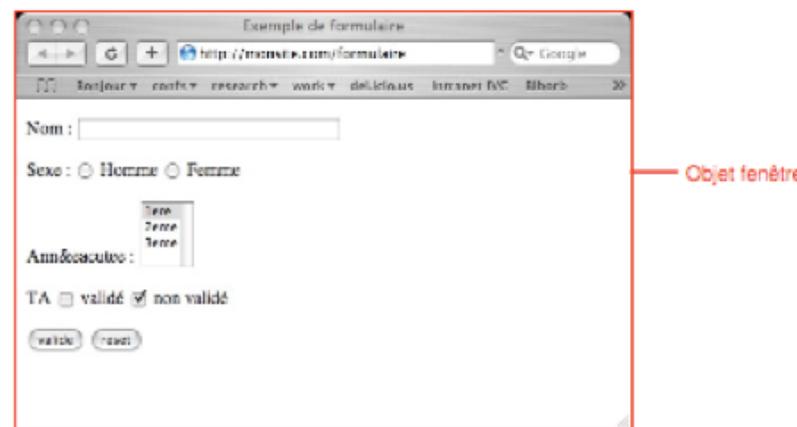
// nombre d'éléments dans le tableau
alert(monTableau.length); // 3
```

# Objets - Tableaux [1]

- Les méthodes de Array

```
var tableau3=tableau1.concat(tableau2);
var chaine=tableau.join(séparateur);
tableau.pop();
tableau.push(liste d' éléments);
tableau.reverse();
tableau.shift();
tableau.unshift(liste d'éléments);
tableau2=tableau1.slice(début, fin);
tableau.sort();
tableau.splice(début, longueur);
tableau.splice(début, longueur, liste d'éléments);
http://www.w3schools.com/js/js\_arrays.asp
http://www.w3schools.com/js/js\_array\_methods.asp
```

# Les objets natifs du navigateur

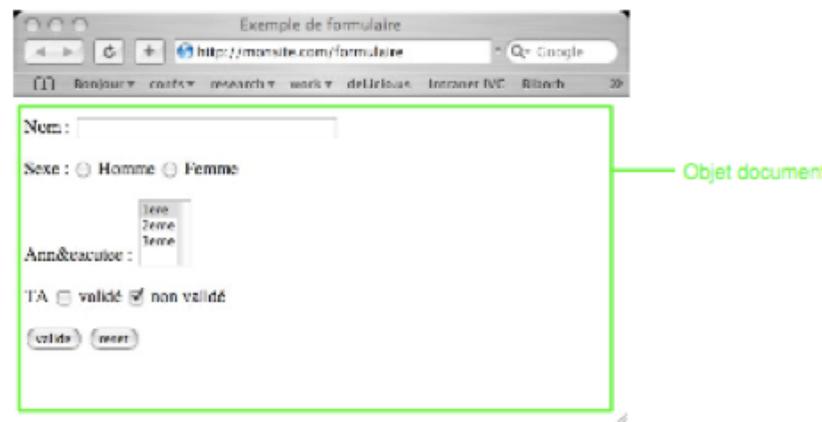


SHAKE THE FUTURE.

# Les objets natifs du navigateur [2]

- Window
  - l'objet racine
  - toute variable globale est un attribut de **window**
  - var x = 0; <=> window.x = 0; en contexte global

# Les objets natifs du navigateur



SHAKE THE FUTURE.

# Les objets natifs du navigateur [2]

- **document**

- représente le document effectivement affiché
- possède un certain nombre de méthodes et d'objets sous-jacents documentés par le W3C :  
le DOM (Document Object Model)
- exemple :  
`document.getElementById("monId").innerHTML = "<p>Mon contenu</p>";`

# Les objets natifs du navigateur

- Trouver un élément HTML
  - `document.getElementById()`
  - `document.getElementsByTagName()`
  - `document.getElementsByClassName()`
- Changer un élément HTML
  - `element.innerHTML` : Change le inner HTML d'un élément HTML
  - `element.textContent` : Obtient le contenu texte d'un élément
  - `element.setAttribute(attribute,value)` : Change l'attribut d'un élément HTML
  - `element.setAttribute(attribute,value)` : Change l'attribut d'un élément HTML
  - `element.style.property` : Change le style d'un élément HTML

# Les objets natifs du navigateur

- Exemple :

```
var x = document.getElementById("main");
var y = x.getElementsByTagName("p");
```

Trouve les élément avec l'id « main » puis donne les éléments <p> dans le « main »

- Eléments accessibles :

- [document.anchors](#)
- [document.body](#)
- [document.documentElement](#)
- [document.embeds](#)
- [document.forms](#)
- [document.head](#)
- [document.images](#)
- [document.links](#)
- [document.scripts](#)
- [document.title](#)

SHAKE THE FUTURE.



# Les objets natifs du navigateur

- Navigation dans le DOM

```
<html>
```

```
<body>
```

```
<h1 id="intro">My First Page</h1>
```

```
<p id="demo">Hello!</p>
```

```
<script>
```

```
var myText =
```

```
document.getElementById("intro").childNodes[0].nodeValue;
```

```
document.getElementById("demo").innerHTML = myText;
```

```
</script>
```

```
</body>
```

```
</html>
```

SHAKI

ENTRALE  
ANTES

# Les objets natifs du navigateur [2]

- **console**

- la console de debuggage
- exemple :

```
console.log("Texte d'essai");
```

# Les objets natifs du navigateur [2,1]

- **Math**

- un objet agrégeant des fonctions mathématiques
- Propriétés : Math.PI et Math.E
- méthodes :  
atan(), acos(), asin(), tan(), cos(), sin(), abs(), exp(), log(), max(), min(), pow(),  
round(), sqrt(), floor(), random()

- exemple :

```
var x = Math.round(18.22);
```

[http://www.w3schools.com/js/js\\_math.asp](http://www.w3schools.com/js/js_math.asp)

# Les objets natifs du navigateur [2]

- Timeouts :

[http://www.w3schools.com/js/js\\_timing.asp](http://www.w3schools.com/js/js_timing.asp)

```
function action() {
    alert("Tadam !");
}

// Toutes les 1000 millisecondes, Tadam !
var timeout = setInterval(action, 1000);

// Au bout de 3500 millisecondes, on arrete ça.
setTimeout(
    function() {
        clearTimeout(timeout);
    },
    3500
);
```



Un timeout est une fonction qui s'exécutera périodiquement ou au bout d'un temps défini.

- se déclare avec **setInterval** ou **setTimeout**
- S'annule avec **clearInterval** ou **clearTimeout**

# Les objets natifs du navigateur [1]

- Objet Date

```
var maDate = new Date()
```

- getYear() : 2 chiffres
- getFullYear() : 4 chiffres
- getMonth() : 0 – 11
- getDate() : 1 – 31
- getDay() : 0 – 6 (dimanche – samedi)
- getHours() : 0 – 23
- getMinutes : 0 – 59
- getSeconds() : 0 – 59

[http://www.w3schools.com/js/js\\_dates.asp](http://www.w3schools.com/js/js_dates.asp)

[http://www.w3schools.com/js/js\\_date\\_methods.asp](http://www.w3schools.com/js/js_date_methods.asp)

# Les objets natifs du navigateur [1]

- L'objet String

Lorsqu'on définit une constante ou une variable chaîne de caractères, JavaScript crée d'une façon transparente une instance String

- 1 propriété : length
- les balises HTML/XHTML ont leur équivalent en méthode
- Liste (non exhaustive) des méthodes:

bold(), italics(), fontcolor(), fontsize(), small(), big(), toUpperCase(), toLowerCase(),  
sub(), sup(), substring(), eval(), split(), replace()

[http://www.w3schools.com/js/js\\_strings.asp](http://www.w3schools.com/js/js_strings.asp)

[http://www.w3schools.com/js/js\\_string\\_methods.asp](http://www.w3schools.com/js/js_string_methods.asp)

# Plan

- Généralités
- Les bases
- Les objets
- **Les évènements**
- Les boîtes de dialogue

SHAKE THE FUTURE.



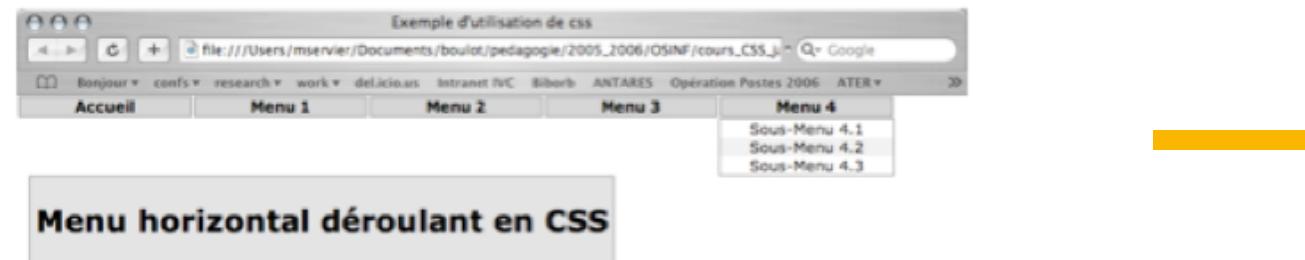
# Les événements

Ces événements permettent d'interagir avec l'utilisateur :

- onClick : clic sur le bouton de la souris
- onDblClick : double-clic sur le bouton de la souris
- onKeyDown : touche du clavier enfoncée
- onKeyPress : la touche du clavier enfoncée est relâchée
- onKeyUp : touche du clavier relâchée
- onMouseDown : bouton de la souris enfoncé sur l'élément input
- onMouseUp : bouton de la souris relâchée sur l'élément input
- onMouseOver : curseur survolant l'objet
- onMouseOut : curseur quittant le survol de l'objet

[http://www.w3schools.com/jsref/dom\\_obj\\_event.asp](http://www.w3schools.com/jsref/dom_obj_event.asp)

# Utilisation pour les menus



SHAKE THE FUTURE.



# Utilisation pour les menus

```
<body>
<div id="menu">
  <dl>
    <dt onmouseover="javascript:montre();"><a href="" title="Retour à l'accueil">Accueil</a></dt>
  </dl>
  <dl>
    <dt onmouseover="javascript:montre('smenu1');">Menu 1</dt>
    <dd id="smenu1">
      <ul>
        <li><a href="#">Sous-Menu 1.1</a></li>
        <li><a href="#">Sous-Menu 1.2</a></li>
        <li><a href="#">Sous-Menu 1.3</a></li>
        <li><a href="#">Sous-Menu 1.4</a></li>
        <li><a href="#">Sous-Menu 1.5</a></li>
        <li><a href="#">Sous-Menu 1.6</a></li>
      </ul>
    </dd>
  </dl>
```

SHAKE THE FUTURE.



# Utilisation pour les menus

```
<dl>
<dt onmouseover="javascript:montre();"><a href="">Menu 2</a></dt>
</dl>
<dl>
<dt onmouseover="javascript:montre('smenu3');">Menu 3</dt>
<dd id="smenu3">
<ul>
<li><a href="">Sous-Menu 3.1</a></li>
<li><a href="">Sous-Menu 3.2</a></li>
<li><a href="">Sous-Menu 3.3</a></li>
<li><a href="">Sous-Menu 3.4</a></li>
<li><a href="">Sous-Menu 3.5</a></li>
</ul>
</dd>
</dl>
<dl>
<dt onmouseover="javascript:montre('smenu4');">Menu 4</dt>
<dd id="smenu4">
<ul>
<li><a href="">Sous-Menu 4.1</a></li>
<li><a href="">Sous-Menu 4.2</a></li>
<li><a href="">Sous-Menu 4.3</a></li>
</ul>
</dd>
</dl>
</div>
<div id="site">
<h1>Menu horizontal déroulant en CSS</h1>
</div>
</body>
```

SHAKE THE FUTURE



# Utilisation pour les menus

```
body {  
margin: 0;  
padding: 0;  
background: white;  
font: 80% verdana, arial, sans-serif;  
color: black;  
}  
dl, dt, dd, ul, li {  
margin: 0;  
padding: 0;  
list-style-type: none;  
}  
#menu {  
position: absolute;  
top: 0;  
left: 0;  
z-index:100;  
width: 100%;  
}  
  
#menu dl {  
float: left;  
width: 12em;  
}  
#menu dt {  
cursor: pointer;  
text-align: center;  
font-weight: bold;  
background-color: #BFBFBF;  
border: 1px solid gray;  
margin: 1px;  
color: black;  
}
```

# Utilisation pour les menus

```
#menu dd {  
display: none;  
border: 1px solid gray;  
color: black;  
background-color: white;  
}  
#menu li {  
text-align: center;  
background: #BFBFBF;  
color: black;  
}  
#menu li a, #menu dt a {  
color: black;  
background: #FFFFFF;  
text-decoration: none;  
display: block;  
height: 100%;  
border: 0 none;  
}  
  
#menu li a:hover, #menu dt a:hover {  
background-color: #BFBFBF;  
color: black;  
}  
#site {  
position: absolute;  
z-index: 1;  
top : 70px;  
left : 10px;  
color: black;  
background-color: #DDDDDD;  
padding: 5px;  
border: 1px solid gray;  
}
```

# Utilisation pour les menus

```
function montre(id) {  
    var d = document.getElementById(id);  
    for (var i = 1; i<=10; i++) {  
        if (document.getElementById('smenu'+i))  
            {document.getElementById('smenu'+i).style.display='none';}  
    }  
    if (d) {d.style.display='block';}  
}
```

# Utilisation pour les menus



SHAKE THE FUTURE.



# Plan

- Généralités
- Les bases
- Les objets
- Les évènements
- Les boites de dialogue

SHAKE THE FUTURE.



# Boîtes de dialogue

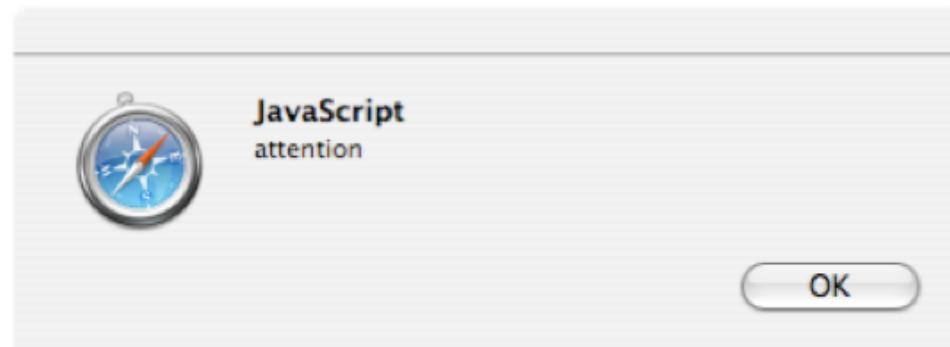
Il existe 3 types de boites de dialogue :

- alert : permet d'alerter l'utilisateur
- confirm : permet à l'utilisateur de confirmer ou d'informer et renvoie sa réponse
- prompt : permet de faire saisir une valeur à l'utilisateur

# Boîtes de dialogue

```
<head><title>Exemple d'utilisation de  
javascript</title></head>  
<body>  
<script type="text/javascript">  
alert("attention");</script>  
</body>
```

# Boîtes de dialogue



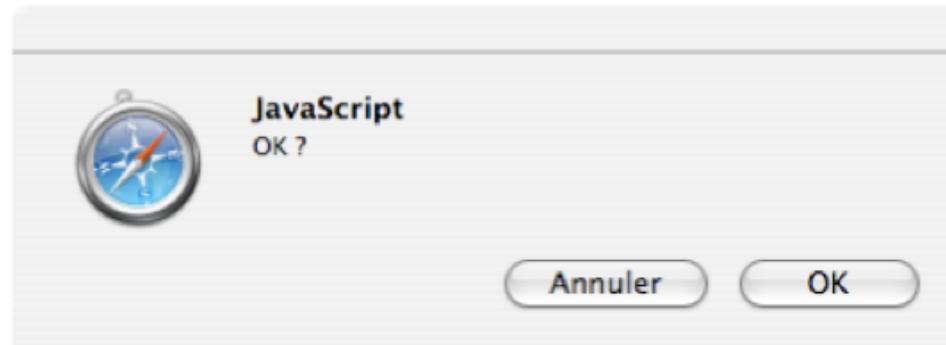
SHAKE THE FUTURE.



# Boîtes de dialogue

```
<head><title>Exemple d'utilisation de  
javascript</title></head>  
<body>  
<script type="text/javascript">confirm("OK  
?")</script>  
</body>
```

# Boîtes de dialogue



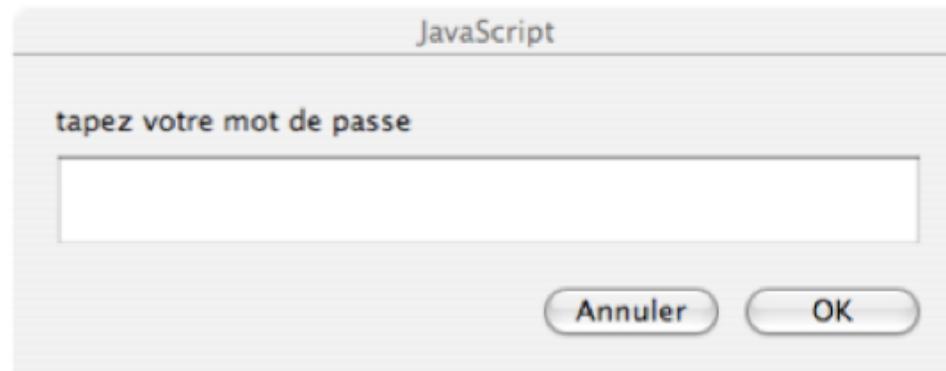
SHAKE THE FUTURE.



# Boîtes de dialogue

```
<head><title>Exemple d'utilisation de  
javascript</title></head>  
<body>  
<script type="text/javascript">prompt("tapez votre  
mot de passe")</script>  
</body>
```

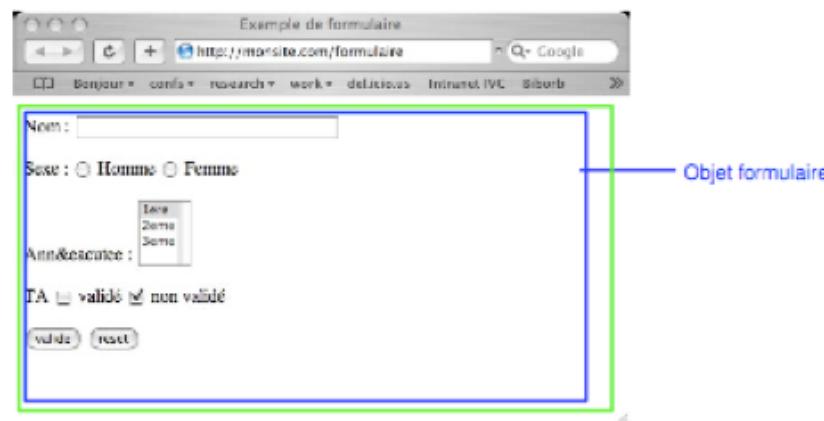
# Boîtes de dialogue



SHAKE THE FUTURE.



# Les objets natifs du navigateur



SHAKE THE FUTURE.

# Boîtes de dialogue

```
<head><title>Exemple d'utilisation de javascript</title></head>
<body>
<form name="mon_formulaire">
<input type="text" value="" name="mon_champ_texte" \ ><input
type="button" value="Saisir" onClick="PromptMessage()" \ >
</form>
<script type="text/javascript">
  function PromptMessage() {
    var saisie = prompt("Saisissez votre texte : ", "Texte par
défaut")
    if (saisie!=null) {
      document.forms ["mon_formulaire"] .
      elements ["mon_champ_texte"] .value=saisie;
    }
  }
</script>
</body>
```

SHAKI

# Exemples d'utilisation de javascript

<http://jquery.com/>

[http://en.wikipedia.org/wiki/Comparison\\_of\\_JavaScript\\_frameworks](http://en.wikipedia.org/wiki/Comparison_of_JavaScript_frameworks)

<http://www.javascripting.com/?sort=rating>

Bootstrap : <http://getbootstrap.com/javascript/>

# Bibliographie

1. Cours de Javascript, T. Lecroq, Université de Rouen
2. JavaScript : concepts de base, Kristen Le Liboux Juillet 2013
3. Javascript, François Pellerin, 2005
4. François Saradin, Les technologies du Web - Cours OSINF, 2005
5. Cours Javascript, Jean-Marie Normand, Science Com