Javascript et les spécificités du DOM

- Principes
- Le DOM
- Les évènements
- Compléments

Et JavaScript dans tout ça !!!

- HTML pour le contenu
- CSS pour la présentation
- JavaScript : langage de script pour le comportement de la page
  - Langage à prototype typé dynamiquement
  - Interprété coté client (par le navigateur)

  - pour l'aspect dynamique des pages
    Traitement local (client) d'évènements déclenchés par l'utilisateur
    - Applications coté client(calculettes, convertisseurs, …)
    - Aspects graphiques (modifications d'images liées à la position de la souris, …)

  - pour limiter les échanges entre client et serveur : permettre des vérifications localement (date, nombre,…)

Où et comment?

- Où placer les scripts?
  - Dans la page : NON

```
〈script type="text/javascript"〉

    /* code JavaScript */

〈/script〉
```

  - Dans un fichier externe (.js) : CE QU'IL FAUT FAIRE

En plaçant dans la partie head
〈script type="text/javascript" src="MonFichier.js"〉〈/script〉

- Comment sont-ils pris en compte?
  - Ils sont interprétés et exécutés par le ≪ navigateur ≫ à l'interprétation de la tête du fichier html (lorsque la balise script est traitée)

Ne pas confondre

- Java : langage compilé, typage statique, syntaxe verbeuse, langage ayant d'autres finalités (application stand alone …)
- EcmaScript : standardisation proposée par l'ECMA correspondant aux différentes versions de javaScript (v3 norme ECMA-262 : javascript 1.8.2)
- Jscript : implémentation de javascript par microsoft (95% de javascript)
- ActiveScript : moteur d'exécution des scripts javascript sous Windows
- AJAX : Asynchronous JavaScript and XML

Typage dynamique

- Variables : mot clé var

- Différents types de littéral :
  - Types prédéfinis
  - Number (*var compteur=0, var prix=4.43*)
    entier ou réel (la partie décimale noté par .)
  - String (*var nom='hernandez')*
  - Boolean (*var continue=true, var arret=false*)

  - Objet prédéfini
  - Null (variable définie mais non valorisée)
  - new Array() : tableau à dimensions quelconques
  - new Date() : date et heure
  - Function : fonction
  - RegExp : expression régulière
  - Object : objet quelconque (dont ceux issus du DOM)

- Fonction prédéfinie typeof(nomvar) retourne une chaîne de caractère correspondant au type interprété pour une variable nomvar

Les objets « utilitaires »

- Classes du langage
  - Date: var maDate = new Date (annee, mois, jour, h, min, s);
    - getDate() : retourne le jour du mois (entre 1 et 31)
    - getDay(), getHours(), getMinutes(), getMonth(),
    - getSeconds(),getFullYear()
  - String
    - length : attribut contenant la taille
    - toUpperCase()
    - ToLowerCase()
    - indexOf(chaine)
    - split(/séparateur/);

Les expressions régulières

- Les expressions régulières pour rechercher des motifs
- Création de l'objet
  - re=/exp/drapeau ou re=new RegExp("exp","drapeau")
    - exp : le motif (^ et $,.,[a-z],\d,\w,*,+,?,{n},{n, m},...)
    - Drapeau : le comportement (g:toutes les occurrences,i:non sensible à la case,gi)
- Utilisation de l'objet :
  - ses méthodes:
    - re.exec(chaine) : renvoie un tableau avec les occurrences du motif trouvées dans chaine
    - re.test(chaine) : renvoie true/false si le motif st trouvé dans chaine,
  - Les méthodes de String
    - String.match(re), String.split(re), ..

Ex: var reg = /^[a-z0-9._-]+@[a-z0-9]{2,}[.][a-z]{2,3}$/

If (reg.exec(email)!=null) alert(≪ l'adresse est valide ≫)

Opérateurs

- Opérateurs de comparaison
  - Égalité == *compteur ==0*
  - Différence != *compteur !=0*
  - Autres 〈 〈= 〉 〉=
- Opérateurs logiques
  - Et && *(compteur ==0) && (compteur2〈=44)*
  - Ou || *(compteur ==4) || (compteur2〈=3)*
  - Non ! *Var continue=true; … ! continue …*

Opérateurs exotiques

- Comparaison de la valeur et du type
  - Égalité stricte ===
  - Différence stricte !==

  Ex : '3'==3 mais '3' !==3

Instructions

- Fonction

function *⟨ident⟩* ([*⟨par*$_1$*⟩* [,*⟨par*$_2$*⟩* [..., *⟨par*$_p$*⟩*]...]){

⟨corps de function⟩

return(⟨valeur⟩);

}

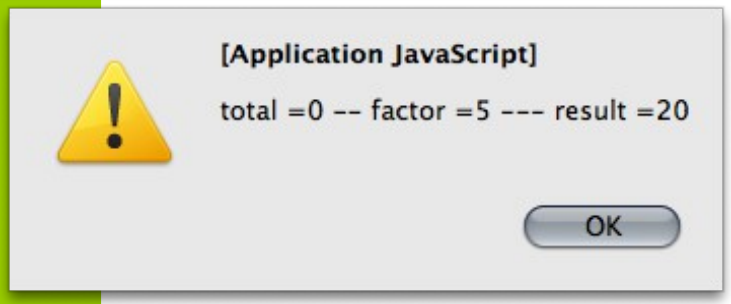
- Appel

⟨ident⟩ ([⟨par$_1$⟩ [,⟨par$_2$⟩ [..., ⟨par$_p$⟩]...])

Retour sur le typage dynamique

```
var total =0;
var factor = 5;
var result = 42;

function compute (base, factor) {
    result = base* factor;
    factor *=2;
    var total = result + factor;
    return total;
} //compute
alert ('compute(5,4) = '+compute(5,4));
alert ('total =' + total +' -- factor ='+ factor +
--- result =' +result);
```



- Variable non déclarée revient à
  - utiliser une variable globale si elle existe
  - Définir une nouvelle variable globale sinon
- Variable non déclarée != paramètre d'une fonction

=〉 Déclarer vos variables + const devant les constantes

Instructions

- Sélection

```
if (⟨expression booléenne⟩)
                    ⟨corps du if⟩
    [else if (⟨expression bouléenne⟩)
                    ⟨corps du if⟩ ]
    [else ⟨corps du else⟩ ]

switch (variable) {
      case ⟨valeur1⟩ : ⟨traitement de valeur1⟩
      case ⟨valeur2⟩ : ⟨traitement de valeur2⟩
  .....
  [default : ⟨traitement autres cas⟩ ;]
}
```

```
var text;
   switch (new Date().getDay()) {
      case 1:
      case 2:
      case 3:
      default:
         text = "Looking forward to the Weekend";
         break;
      case 4:
      case 5:
         text = "Soon it is Weekend";
         break;
      case 0:
      case 6:
         text = "It is Weekend";
   }
alert(text) ;
```

Instructions

- Boucles
- while (⟨*expression booléenne*⟩)
  ⟨*corps de boucle*⟩

- do
  ⟨*corps de boucle*⟩
  while (⟨*expression booléenne*⟩) ;

- for(var ⟨*ident*⟩=⟨*init*⟩; ⟨*test de fin*⟩; ⟨*opération*⟩)
  ⟨*corps de boucle*⟩

Les pop-up

- Message :

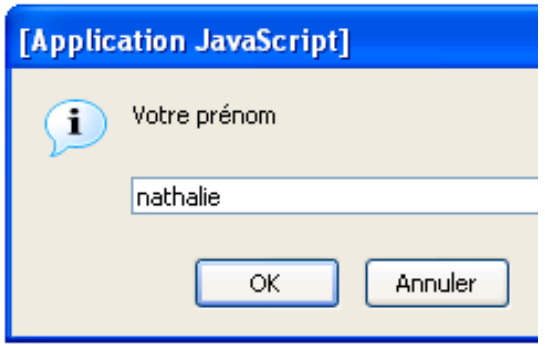
alert(≪ message ≫);

- Demande de confirmation :

booleen=confirm(≪ message ≫);

- Zone de saisie :

chaine=prompt(≪ message ≫);



=〉Penser à convertir la chaîne si vous voulez manipuler la réponse dans un autre type

var conv=parseInt(chaine,baseNum); //ou parseFloat

baseNum correspond en général à 10

Attention : parseInt("0123") sera automatiquement convertit en basse octal

parseInt("42dommage332",10) renvoie 42

Un petit .js

- Ecrire un script qui permet de saisir par l'intermédiaire d'une fenêtre de zone de saisie un numéro de sécu et qui vérifie si il est correct . Les critères à vérifier devront apparaître dans la page html.

Un numéro de sécu est constitué d'une séquence de chiffres :

S-AA-MM-DD-VVV-NNN-CC où :

S est le sexe (1 ou 2)

AA est l'année de naissance (2 chiffres)

MM est le mois de naissance (2 chiffres)

DD est le numéro du département de naissance (2 chiffres)

VVV est le numéro de la ville dans ce département (3 chiffres)

NNN est le numéro d'ordre sur le registre d'état civil (3 chiffres)

n-ième naissance ce mois-ci dans cette ville

CC est la clef vérifiant si le reste des infos est cohérent (2 chiffres correspondant au complément à 97 du reste de la division euclidienne du numéro de sécu par 97

- Mais où est le dynamisme ….

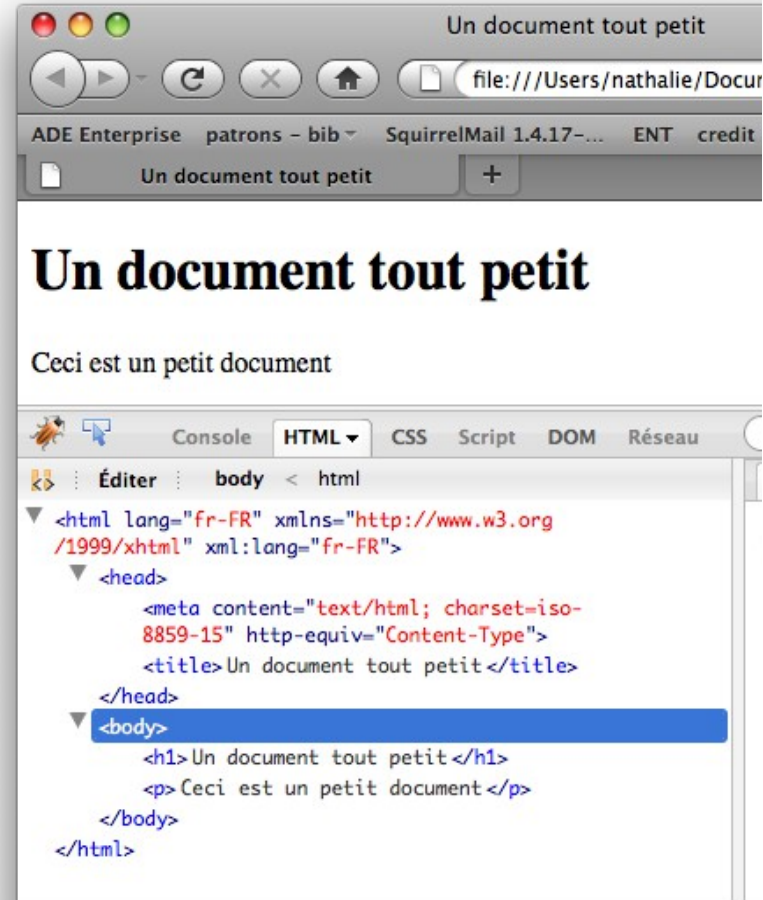
DOM

- Document Object Model :
  - Permet de représenter et de manipuler en mémoire les éléments des documents XML
  - Plusieurs modules :
    - Core
    - HTML
    - Events
    - Style …
  - Standardisé par le W3C
    - DOM niveau 2 (Nov 2000) suivi par les navigateurs
    - Niveau 3 (Février 2004) en cours…

DOM – ce qu'on devrait avoir

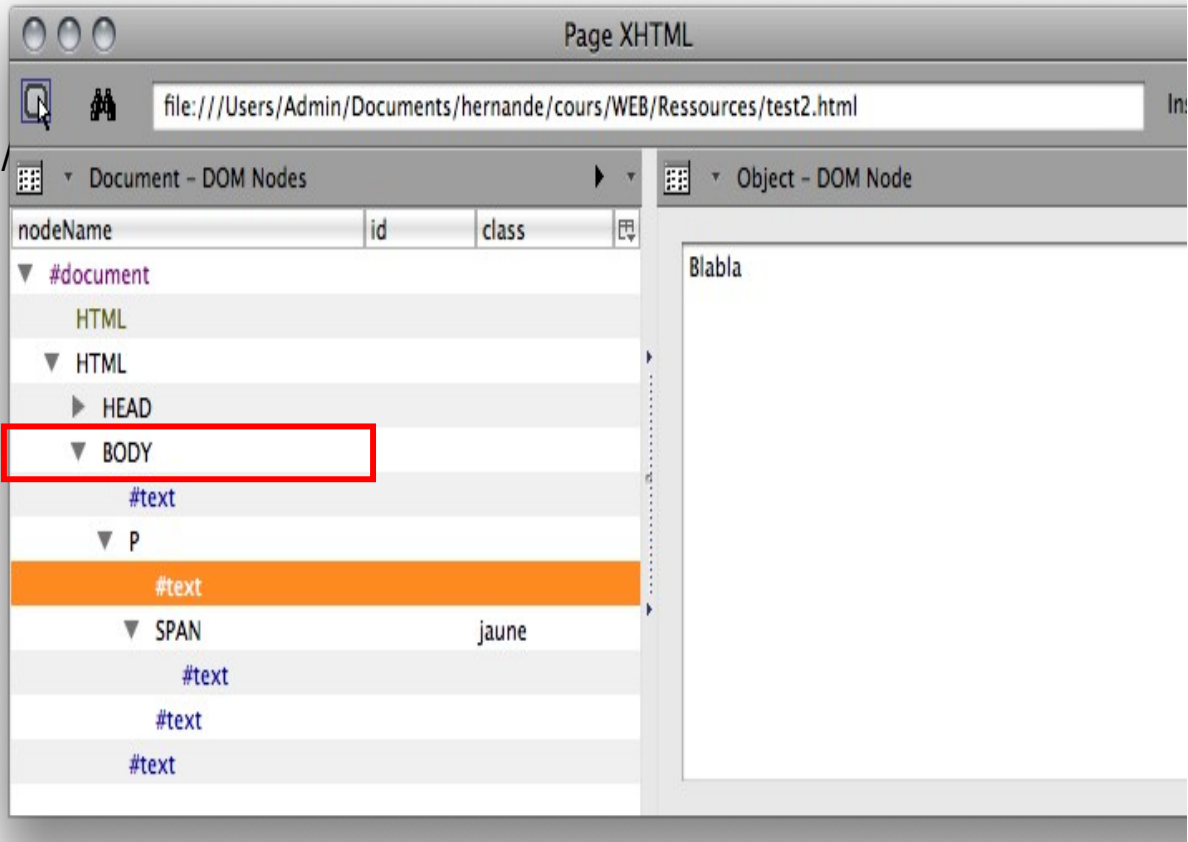
- Représentation homogène du document sous forme d'arbre

```
〈!DOCTYPE html〉
〈html〉
〈head〉
    〈meta charset="utf-8" /〉
    〈title〉Un document tout petit〈/title〉
〈/head〉
〈body〉
〈h1〉Un document tout petit〈/h1〉
〈p〉Ceci est un petit document〈/p〉
〈/body〉
〈/html〉
```



Attention DOM – ce qu'on a..

```
<!DOCTYPE html>
<html>
<head>
<title>Page XHTML</title>
<meta charset="utf-8" />
<link rel="stylesheet" type="text/css" href="style2.css" />
</head>
<body>
<p>Blabla
<span class="jaune">boîte en position relative</
 suite du blabla.</p>
</body>
</html>
```



+Balise
+Contenu textuel
+Attribut

NOEUD

- Chaque élément du document
  - est un nœud,
  - hérite de l'interface Node

- Propriétés de Node
  - **nodeName** (balise ou #text ou nom de l'attribut)
  - **nodeType** (Node.ELEMENT_NODE ou 1, Node.TEXTE_NODE ou 3, Node.ATTRIBUT_NODE ou 2)
  - **nodeValue**(texte, valeur de la propriété ou null)
  - Attributes (QUE pour les éléments)
  - parentNode
  - firstChild,lastChild
  - previousSibling,nextSibling

exemple

```
<h1 id="header">Personnes inscrites</h1>
<ul id="people">
    <li id="al">Alexis</li>
    <li id="nioute">Anne-Julie</li>
    <li id="elodie">Élodie</li>
    <li id="mimi">Marie-Hélène</li>
    <li id="xavier">Xavier</li>
</ul>
```

header.nextSibling.nodeType
header.firstChild.nodeType==
header.childNodes.length ==
xavier.lastChild.nodeValue ==
nioute.parentNode.nodeName

NOEUDS (suite)

- Méthodes définies sur les nœuds :
  - appendChild(noeud) ajoute un noeud fils à un nœud
  - replaceChild(new,old) remplace le fils old par new et renvoie old
  - removeChild(old) supprime le nœud old du DOM : seul moyen de supprimer un élément
  - hasChildNodes() : indique si le nœud à des fils
  - …

Interface Document

- L'interface Document :
  - Elle permet de traiter le document parsé
  - Permet d'accéder aux noeuds du document chargé dans le navigateur

- Les méthodes importantes :
  - document.createElement(nom) : Crée nœud élément
  - document.createTextNode(texte) : Crée un nœud texte
  - Recherche de noeuds :
    - document.getElementsByTagName(nomBalise) : recherche les éléments selon leur balise, retourne un tableau
    - document.getElementById(id) : recherche un élément selon son attribut id, retourne l'élément.

Interface Document

- Ex : Ajout du texte ≪ TRES IMPORTANT ≫ mise en évidence dans une balise STRONG dans le paragraphe d'id="monPara" :

```
//le nœud strong n'existe pas dans l'arbre DOM du document
nouveau_strong=document.createElement('strong');
//le nœud texte n'existe pas dans l'arbre DOM du document
nouveau_txt=document.createTextNode('TRES IMPORTANT');
//le nœud texte est intégré au nœud strong
nouveau_strong.appendChild(nouveau_txt);
//le nœud strong est intégré à l'arbre DOM du document au niveau de la balise paragraphe d'identifiant monPara
document.getElementById('monPara').appendChild(nouveau_strong);
```

Interface Element

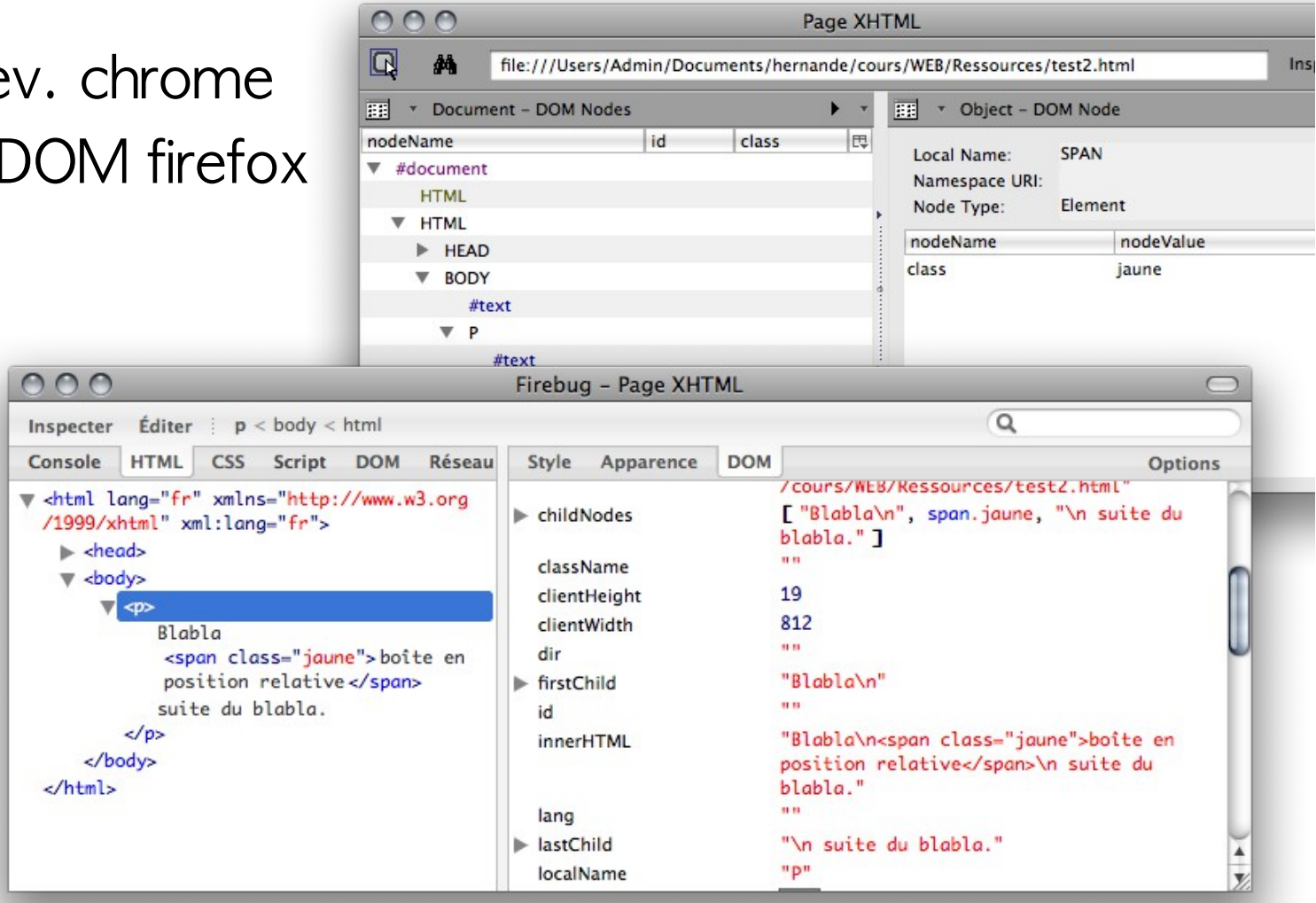
- Interface permettant de manipuler les noeuds qui correspondent à des balises
- Propriétés et méthodes
  - tagName : retourne la balise de l'élement
  - getAttribut(attName) : retourne la valeur de l'attribut attName
  - getElementsByTagName(tag) : recherche à partir de l'élément les descendants de balises tag
  - removeAttribute(name) : retirer la valeur d'un attribut (mais pas si la valeur par défaut)

HTMLDocument

- Module du DOM dédié à HTML
- Il définit une interface spécifique à chaque balise HTML
- Spécialise Element
- Pour tous les élements ajout de :
  - ClassName //classe css eventuelle
  - dir : direction du texte (langue orientale)
  - id
  - lang
  - title : texte alternatif

Outils DOM

- Utiliser des outils DOM pour connaître l'état actuel du DOM
  - Outils de dev. chrome
  - Inspecteur DOM firefox
  - Firebug



Retour au petit.js

- Essayez de modifier le .js pour que le résultat de la vérification du numéro de sécu apparaisse dans la page html

Gestion des évènements

- *Objectifs :*
  - *Ergonomie*: Aider l'utilisateur, ne pas le géner.
  - *Accessible*: Le site ne doit pas dépendre du script.
  - *Facile à implémenter*: discret pour le développeurs web.
  - Séparation du comportement et de la structure.
  - Association des fonctions aux événements par le script lui-même.
- Comment ?
  - Ajouter des comportements aux noeuds

Gestion des évènements

- Concrètement : ajouter une action pour gérer un événement arrivant sur un noeud ≪ node ≫
  - node.addEventListener(evenement,action, false) ; //pour tous sauf IE, false : permet de préciser si l'évènement bouillonne (ie se propage à ses descendants)
  - node.attachEvent (onevenement,action) //pour IE
  - action est une référence vers une fonction

Noms des evènements

- événements page et fenêtre
  - abort - s'il y a une interruption dans le chargement
  - error - en cas d'erreur pendant le chargement de la page
  - load - après la fin du chargement de la page
  - unload - se produit lors du déchargement de la page (par changement de page, en quittant)
  - onresize - quand la fenêtre est redimensionnée
- événements souris
  - click - sur un simple clic
  - dblclick - sur un double clic
  - mousedown - lorsque que le bouton de la souris est enfoncé, sans forcément le relâcher
  - mousemove - lorsque la souris est déplacée
  - mouseout - lorsque la souris sort de l'élément
  - mouseover - lorsque la souris est sur l'élément
  - mouseup - lorsque le bouton de la souris est relâché
  - scroll

Noms des evènements

- événements clavier
  - keydown - lorsqu'une touche est enfoncée
  - keypress - lorsqu'une touche est pressée et relâchée
  - keyup - lorsqu'une touche est relâchée
- événements formulaire
  - blur - à la perte du focus
  - focus - lorsque l'élément prend le focus (ou devient actif)
  - reset - lors de la remise à zéro du formulaire (via un bouton "reset" ou une fonction reset())
  - select - quand du texte est sélectionne
  - submit - quand le formulaire est validé

Gestion d'évènements

- Ajouter les évènements quand les éléments existent dans le DOM

```
function addListener(element,basename,handler) {
if (element.addEventListener) element.addEventListener(basename,handler,false);
else if (element.attachEvent)element.attachEvent('on'+basename,handler);// pour
   IE<11
}//addListener
function Monaction(event) {
//traitement à effectuer quand l'element d'id MONNOEUD recoit l'evenement event
}//addListener

function initEvenements() {
// initialiser les comportements des nœuds quand ils existent dans le
   document
    //var noeud=document.getElementById('MONNOEUD')

  //addListener(noeud,'MonEvent',Monaction);

}

addListener(window,'load',initEvenements);
```

Mon premier .js en mieux

- Reprendre le script de vérification du numéro de sécu et faire que le résultat de la validation s'affiche dans le document html

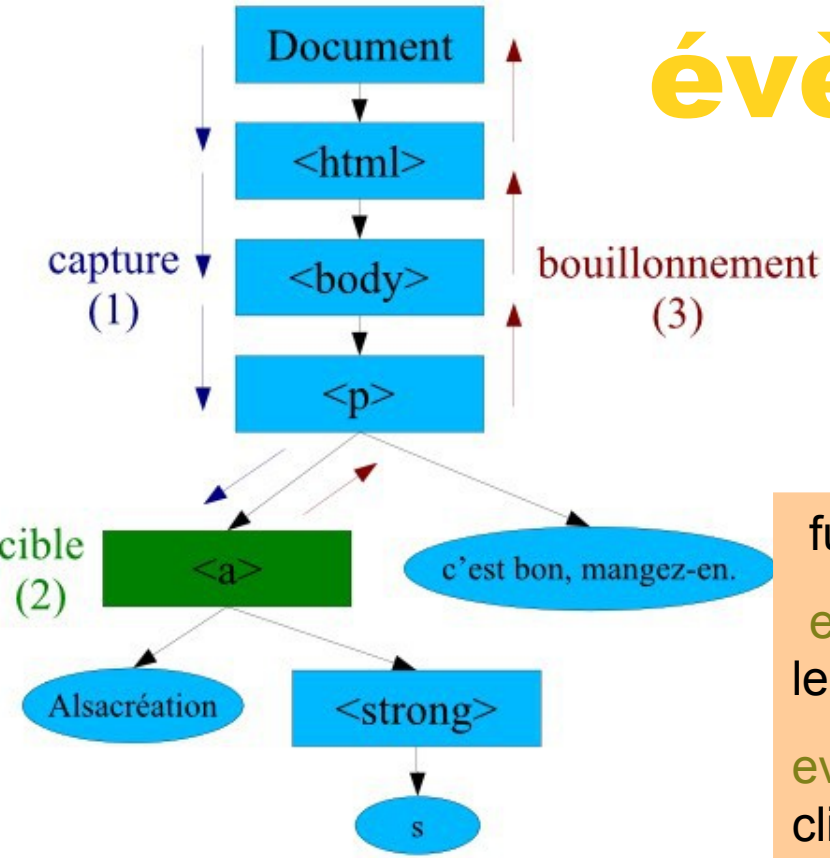
Mon premier .js en bcp mieux

- Reprendre le script de vérification du numéro de sécu, mais maintenant
  - Le numéro sera saisi dans un formulaire (et pas dans la fenêtre pop-up)
  - Réfléchir au type du bouton du formulaire

Mon premier .js en bcp mieux

- Reprendre le script de vérification du numéro de sécu, mais le bouton est de type « submit »

Propagation des évènements



Syntaxe non compatible avec IE<11

```
function monGestionnaire (event) {

 event.target ; //récupérer la cible d'un évènement (nœud sur lequel l'évènement s'est produit)

event.type ; //récupérer le type d'un évènement (load, click ,...)

 event.stopPropagation() ; //arrêt de la propagation : pas de bouillonnement :

 event.preventDefault(); //Eviter d'exécuter le traitement prédéfini d'un événement (si bien pas de gestionnaire pour l'évènement)

}

addListener(monNoeud, monEvenement, monGestionnaire);
```

Empêcher l'exécution de l'action associée à un formulaire

```
function stopEvent(e) {
    if (e.stopPropagation) {
        e.stopPropagation(); e.preventDefault();
    } else { //pour IE
        e.cancelBubble = true;
        e.returnValue = false;
    }
} // stopEvent

function checkForm(event) { //event correspond à l'évènement qui a déclenché la fonction
faulty=null;
//code permettant de mettre faulty a faux si mauvais format dans un champ et à vrai sinon
 if (!faulty) return;
    stopEvent(event); // sous entendu else ie dans le cas ou y a au moins une erreur on arrête
l'evenement
}
function addFormChecks() {
var form=document.getElementById('formu');
addListener(form, 'submit', checkForm)
}
addListener(window, 'load', addFormChecks);
```

Un langage à prototype

- Ne repose pas sur des classes et sous-classes pour l'héritage
- Tout objet est doté d'une propriété prototype
- Différences :
  - Pas de classe et instance / tout est instance
  - Définition des classes, puis création de l'instance avec le constructeur /Le constructeur sert à définir les objets et instances
  - L'objet père est un prototype de la fonction constructeur de la fille
  - Les fonctions constructeurs définissent un jeu initial de propriété qui peuvent être modifié pour un individu ou pour les objets reposant sur le même prototype

Extension du prototype

```
String.prototype.endsWith = function(suffix) {
return this.length - suffix.length == this.lastIndexOf(suffix);
};
String.prototype.isBlank = function() {
return null != this.match(/^\s*$/);
};
String.prototype.startsWith = function(prefix) {
   return 0 == this.indexOf(prefix);};
String.prototype.trim = function() {
 return this.replace(/^\s+|\s+$/g, '');};
```

Création de nouveaux objets

```
//fonction constructeur
function nvObjet(att1,att2,…,attn) {
//creation des attributs
  this.attr1=att1;
  this.att2=att2;
 …
  this.attn =attn
 //création des méthodes à partir de fonctions anonymes
this.method = function (param){
   //code
   }
//création des méthodes à partir de fonctions définies
this.method2=ma_fonction_définie; //=〉attention si des arguments sont passés à la
    fonction, la fonction ne sera plus affectée par référence mais ce sera le resultat de
    l'interprétration utiliser une fonction anonymes ou ma_fonction_définie.call(arg1,…)
 }

//création de l'objet
monObj=new nvObjet(…)
```

Capture des exceptions

```
Try {
 //code susceptible de lancer une exception
 }
Catch (e){
//traitement de l'exception stockée dans l'objet e
}
```

- Type d'erreur prédéfini :
  - ReferenceError : utilisation d'un identifiant inconnu
  - TypeError : argument ou variable ayant un type invalide
  - …

=〉rare

Gestion des exceptions

- Lancer ses propres erreurs :

```
//Créer un objet erreur compatible avec Error:
function CustomError(message,info) {
  this.name='CustomError'; //champ nécessaire pour la compatibilité
  this.message=message; //champ nécessaire pour la compatibilité
  this.customInfo =info
}
try{
// code qui s'execute
throw new CustomError('souci',42);
//code court-circuité
} catch(e) {
 if(e instanceof CustomError)
    alert (e.message +' / '+e.customInfo);
}
```