

# INTRODUCTION AUX OUTILS POUR LE WEB 2.0

## CHAPITRE 3 - JAVASCRIPT

COPYRIGHT LAURENT HENOCQUE  
POLYTECH MARSEILLE  
DÉPARTEMENT INFORMATIQUE  
MIS À JOUR EN NOVEMBRE 2013

[LAURENT.HENOCQUE.COM](http://LAURENT.HENOCQUE.COM)



# LICENCE CREATIVE COMMONS

---



Cette création est mise à disposition selon le Contrat  
Paternité-Partage des Conditions Initiales à l'Identique  
2.0 France disponible en ligne

<http://creativecommons.org/licenses/by-sa/2.0/fr/>

ou par courrier postal à Creative Commons, 559 Nathan  
Abbott Way, Stanford, California 94305, USA.



# JAVASCRIPT PLAN

---

- présentation
- syntaxe
- événements
- objets du navigateur
- le modèle objet du document (DOM)



# JAVASCRIPT : LE LANGAGE DE PROGRAMMATION DES CLIENTS WEB

---

- Javascript est devenu le langage incontournable pour le développement de code côté client web
- Accès au stockage local (bases de données)
- Accès à des sites et serveurs distants (Ajax)
- Réalisation d'interfaces sophistiquées (bibliothèques comme JQuery, scriptaculous, etc.)
- Interaction favorable avec CSS: plus besoin de la solution propriétaire Adobe Flash



# JAVASCRIPT : UN LANGAGE DE PROGRAMMATION SERVEUR

---

- Javascript peut être utilisé pour réaliser des serveurs (Nodejs)
- Javascript peut être utilisé pour réaliser des applications de bureau (Node-webkit)



# BONJOUR LE MONDE

---

```
<!DOCTYPE html>
```

```
<html>
```

```
<body>
```

```
<script>
```

```
document.write("<p>Bonjour !</p>");
```

```
</script>
```

```
</body>
```

```
</html>
```

Bonjour !

Javascript écrit à l'endroit où  
le code est présent dans le  
source de la page



# INSERTION DANS UN ÉLÉMENT

---

```
<body>
```

```
<h1 id="demo">Ma Page</h1>
```

```
<script>
```

```
document.getElementById("demo").
```

```
    innerHTML="Bonjour !";
```

```
</script>
```

**Bonjour !**



# FONCTIONS

---

```
<head>
```

```
<script>
```

```
    function f() {  
        alert("Bonjour !");  
    }
```

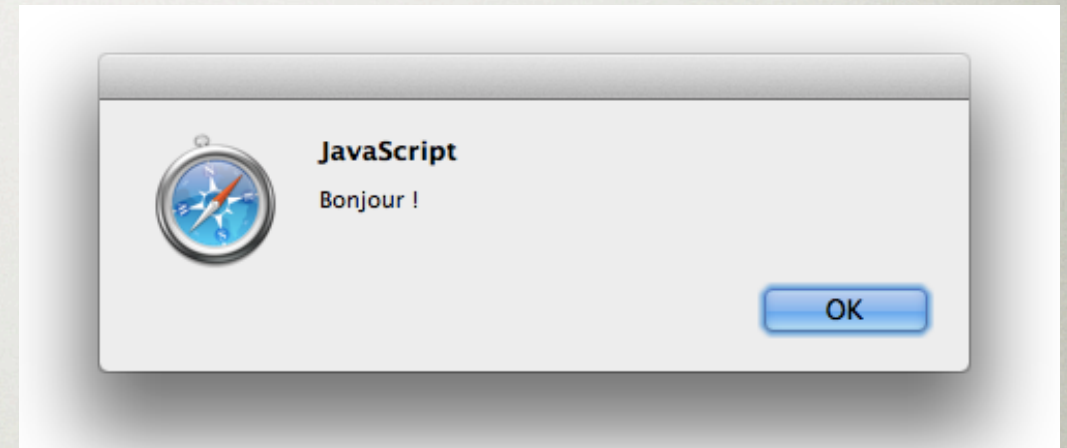
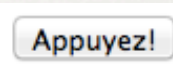
```
</script>
```

```
</head>
```

```
<body>
```

```
<button onclick="f()">Appuyez!</button>
```

```
</body>
```





# JAVASCRIPT DANS UN FICHER EXTERNE

---

```
<h1 id="demo">Ma Page</h1>
```

```
<button type="button" onclick="f()">
```

Appuyez !

```
</button>
```

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



monScript.js doit implanter la fonction "f()"



# JAVASCRIPT

SYNTAXE



# CARACTÈRES

---

- Javascript est sensible à la casse (majuscules)
- Javascript peut insérer automatiquement un point virgule (semicolon) en fin de ligne, avec des effets perturbants.
- Il est recommandé de toujours terminer les expressions avec des points virgules
- Commentaires comme en C: `//` et `/* ... */`



# VARIABLES

---

- Une variable est globale si elle est déclarée en dehors des fonctions ou si elle n'est pas précédée de 'var'
- Une variable peut être déclarée après sa première utilisation
- Une variable globale n'a pas à être déclarée
- Un mode 'strict' permet d'empêcher l'utilisation de variables non déclarées



# VARIABLES: UN PANORAMA

---

```
var x = 0; // globale

function foo() {
  var z = 'Z', r = 'R'; // 2 variables locales
  m = 'M'; // globale non déclarée ailleurs
  function bar() {
    var r = 'S'; // variable locale (pas de conflit avec 'foo'
    z = 'Y'; // la variable de foo (mécanisme appelé 'closure').
  }
  v = 20; // locale car déclarée juste après
  var v;
  bar(); // appel de bar
  return x; // 'x' visible car globale
}
foo(); // appel de foo
alert(z); // provoque une ReferenceError exception (z inaccessible)
```



# TYPES PRIMITIFS : LES NOMBRES

---

- Le seul type numérique est un nombre **flottant double IEEE-754**, à 16 chiffres significatifs.
- Egaleme<sup>nt</sup> '**Infinity**' et '**NaN**' (Not a Number)
- Beaucoup d'opérations arithmétiques donnent des résultats étranges, à cause de l'encodage.
  - Par exemple,  $5 * 1.015 \neq 5.075$  et  $0.06 + 0.01 \neq 0.07$ .
- Les opérations sur les entiers ne donnent les résultats espérés que si tous les calculs intermédiaires sont entiers



# TYPES PRIMITIFS: LES CHÂÎNES

---

- Une chaîne de caractères est entourée de guillemets simples ou doubles: `"test"` `'une autre chaîne'`
- `'+'` binaire est l'opérateur de concaténation de chaînes `'a' + 'b'` vaut `'ab'`
- le caractère à la position `'i'` d'une chaîne est obtenu par:  

```
var slt = "Salut !";  
var l = slt.charAt(2);  
var a = slt[1];
```
- la comparaison de chaînes primitives (`==`) donne les résultats attendus. Elle dépend de la casse.



# L'OBJET STRING

---

- On peut aussi créer des chaînes sous forme d'instances de 'String'.
- `var slt = new String("Salut!");`
- Ici l'opérateur de comparaison '==' compare les objets et non les chaînes

```
var s1 = new String("Salut !");
```

```
var s2 = new String("Salut !");
```

```
s1 == s2; // false, 2 objets distincts.
```

```
s1.valueOf() == s2.valueOf(); // true.
```



# LE TYPE BOOLEAN

---

- Valeurs '**true**' et '**false**'
- Dans un contexte logique, **0**, **-0**, **null**, **NaN**, **undefined**, et la chaîne vide ('') valent '**false**'
- Javascript réalise automatiquement de nombreuses conversions.
- Il est conseillé au débutant de ne pas programmer en s'appuyant sur ces conversions implicites



# TABLEAUX

---

- Les tableaux (Array) sont des objets indexés par des entiers, à partir de la position zéro
- Sont 'creux' et supportent des éléments indéfinis
- Ont une propriété ('length'), et des fonctions ('join', 'push', 'slice')
- On peut déclarer un 'Array' littéral, ou avec un constructeur



# EXAMPLES ARRAY

---

```
myArray = [0,1,,,4,5];           //2 undefined
myArray = new Array(0,1,2,3,4,5); //length 6
myArray = new Array(365);        //length 365

alert(myArray[1]);
alert(myArray["1"]);

myArray.push(6);
```



# OBJETS = TABLEAUX ASSOCIATIFS

---

- Contrairement aux tableaux indexés par des entiers, les objets sont indexés par des labels

```
ferrari = {couleur: "rouge",  
puissance: "430"};
```

```
ferrari["couleur"]; // "rouge"
```

```
ferrari.couleur;    // "rouge"
```



# TABLEAUX DE TABLEAUX ET D'OBJETS

---

```
voitures = [{col: "rouge", cv: 430},  
             {col: "blanc", cv: 90}];
```

```
voitures[0]["cv"];           // 430
```

```
motos = {bfg:{col:"noir", cv:130},  
          voxan:{col:"or", cv:170}};
```

```
motos["bfg"]["cv"]; // 130
```

```
motos.voxan.col;      // "or"
```



# DATE

---

- Une date compte les millisecondes depuis 1970-01-01 00:00:00 UT dans un intervalle de  $\pm 10$  puissance 8 jours. Les mois commencent à zéro

```
new Date()           // la milliseconde courante
```

```
new Date(2013,11,25) // le 25 novembre 2013
```

```
var d = new Date(2013,11,25,14,25,30)
```

```
//25 novembre 2013 à 14:25:30
```

```
new Date("2013-11-25 14:25:30")
```

```
// à partir d'une chaîne.
```



# FONCTIONS D'ACCÈS DE DATE

---

```
// affiche '2013-11-25':  
alert(d.getFullYear() + '-' +  
(d.getMonth()+1) + '-' +  
d.getDate() );  
  
// toString produit 'Mon Nov 25 2013  
14:25:30 GMT-0500 (EST)':  
alert(d);
```



# MATH

---

- La classe Math définit les constantes mathématiques usuelles:

**Math.E**, **Math.PI**, **Math.LN2** ...

- Et les fonctions:

**Math.abs(n)**, **cos(rad)**, **acos(n)**,  
**ceil(n)**, **floor(n)**, **round(n)**,  
**max(a,b)**, **pow(x,n)**, **sqrt(n)**,  
**random()** ...



# ERROR

---

- Pour signaler une erreur:

```
throw new Error("On galère ici...");
```

- L'erreur sera récupérée dans la console d'un débbugger (par exemple 'Firebug' avec Firefox, ou Chrome en mode debug):



# OPÉRATEURS DE COMPARAISON

---

- == Egalité
- != Inégalité
- > Supérieur strict
- >= Supérieur ou égal
- < Inférieur strict
- <= Inférieur ou égal
- === Identique (== et du même type)
- !== Non identique



# COMPARAISON ET CONVERSIONS

---

- `==` et `!=` réalisent des conversions implicites (dites en anglais 'type coercion')
- `===` et `!==` ne font pas de conversion, et comparent en premier lieu le type des objets
- Utiliser de préférence `===` et `!==`



# UNDEFINED ET NULL

---

- **'undefined'**: type que possède une variable encore non initialisée, ou une propriété inexistante sur un objet.
- **'undefined'** est un type primitif, converti en **'false'** en contexte logique. Il n'y a pas de littéral **'undefined'**. Pour tester on écrira:
  - `typeof x === 'undefined'`
- **'null'**: affecté pour indiquer qu'une variable est vide.
- **'null'** est de type **'object'**, égal à **'false'** en contexte logique



# OPÉRATEURS

---

- arithmétique binaire:  $+$ ,  $-$ ,  $*$ ,  $/$  (flottante),  $\%$  (modulo entier)
- arithmétique unaire:  $+$  (conversion de chaîne en nombre),  $-$  (opposé),  $++$  et  $--$  (incrément et décrément postfixes)
- affectation  $=$ ,  $+=$ ,  $-=$ ,  $*=$ ,  $/=$ ,  $\%=$



# EXEMPLE D'AFFECTATION DE TYPES PRIMITIFS

---

```
var x = 1;
```

```
x *= 3;
```

```
alert(x); // affiche: 3
```

```
x /= 3;
```

```
alert(x); // affiche: 1
```

```
x -= 1;
```

```
alert(x); // affiche: 0
```



# AFFECTATIONS D'OBJETS

---

```
var o1 = {a: 1};  
var o2 = {a: 0};  
var o3 = o2; // o3 est une référence à o2
```

```
o2.a = 2;  
alert(o1.a + " " + o2.a + " " + o3.a); // affiche 1 2 2
```

```
o2 = o1; // o2 est maintenant une référence à o1  
        // o3 reste la seule référence à ce qu'était o2  
alert(o1.a + " " + o2.a + " " + o3.a); // affiche 1 1 2
```

```
o2.a = 7; // modifie o1  
alert(obj_1.a + " " + obj_2.a + " " + obj_3.a); // affiche 7 7 2
```



# AFFECTATIONS DÉSTRUCTURÉES

---

```
var a, b, c, d, e;
```

```
[a, b] = [3, 4];
```

```
alert(a + ',' + b); // affiche: 3,4
```

```
e = {foo: 5, bar: 6, baz: ['Baz', 'Boz']};
```

```
var arr = [];
```

```
({baz: [arr[0], arr[3]], foo: a, bar: b}) = e;
```

```
alert(a + ',' + b + ',' + arr); // 5,6,Baz,,,Boz
```

```
[a, b] = [b, a]; // échange a et b
```

```
alert(a + ',' + b); // affiche: 6,5
```



# OPÉRATEURS LOGIQUES

---

- NOT (NON) : **!a**
- OR (OU) : **a || b**
- AND (ET) : **a && b**
- CONDITIONNEL : **(c ? t : f)**

Dans un contexte logique, toute expression évalue à '**true**' sauf: **false** bien sûr, les chaînes "", ' ', les nombres **0**, **-0**, **NaN**, les valeurs spéciales **null**, et **undefined**

La fonction **Boolean(exp)** peut être utilisée pour convertir en booléen



# IF .. ELSE

---

```
if (expr) {  
    //instructions;  
} else if (expr2) {  
    //instructions;  
} else {  
    //instructions;  
}
```



# SWITCH

---

```
switch (expr) {  
    case UNEVALEUR:  
        //instructions;  
        break; //(optionnel)  
    case UNEAUTREVALEUR:  
        //instructions;  
        break; //(optionnel)  
    ...  
    default: //(optionnel)  
        //instructions;  
        break;  
}
```



# FOR

---

```
for (init; condition; incrément) {
```

```
    /*
```

```
        instructions exécutées à chaque  
tour de boucle, tant que 'condition' est  
satisfaite, avec évaluation de  
l'incrément à la fin
```

```
    */
```

```
}
```



# WHILE

---

```
while (condition) {
```

```
    /*
```

```
        instructions exécutées à chaque  
tour de boucle, tant que 'condition'  
est satisfaite
```

```
    */
```

```
}
```



# FONCTIONS

---

- Une fonction permet de grouper les instructions d'un ensemble de calculs
- Une fonction possède un nom, des noms de paramètres, et une valeur de retour.

```
function foo(text) {alert (text);}
```

- On appelle une fonction en la nommant et en passant des valeurs de paramètres entre parenthèses

```
foo("message");
```

- On accède aux paramètres comme à des variables usuelles



# RETURN

---

- La valeur de retour d'une fonction est donnée par l'instruction "return"

```
function deuxfois(i) {return 2*i;}
```

```
var a= deuxfois(10);
```

- Si cette instruction est manquante, la valeur retournée est 'undefined'



# PARAMÈTRES D'APPEL DES FONCTIONS

---

- Les paramètres possédant des types primitifs sont passés par valeur: la fonction ne peut pas modifier cette valeur à l'extérieur
- Les paramètres sont désignés par leur nom, et peuvent être utilisés comme des variables locales
- Un paramètre manquant à l'appel est 'undefined'
- La liste '**arguments**' peut être utilisée comme un tableau (ce n'en est cependant pas un) pour accéder à la totalité des paramètres
- **arguments[0], arguments[1], ... arguments[n]**
- **arguments.length // le nombre d'arguments**



# FONCTION = OBJET

---

- En Javascript une fonction est un objet, pouvant être affectée à une variable.

```
function foo(text) {...}
```

```
alert (foo("du texte"))
```

```
var bar=foo;
```

```
alert (bar("du texte"))
```



# CLOSURE

---

- Une fonction peut être déclarée dans une autre fonction.
- Elle accède alors à toutes les variables locales de la fonction englobante
- Ces variables et leurs valeurs sont préservées même après que la fonction englobante ait fini de s'exécuter (mécanisme de 'closure')



# EXEMPLE DE 'CLOSURE'

---

```
var v = "v dehors", bar, baz;
function foo() {
    var v = "v dedans";
    bar = function() { alert(v) };
    baz = function(x) { v = x; };
}
foo();
baz("v dedans après baz");
bar(); // affiche "v dedans après baz"
alert(v); // affiche "v dehors"
```



# OBJETS

---

- Les objets ont une identité et ne peuvent être égaux qu'à eux mêmes
- Ils ont des propriétés, qui peuvent être des fonctions (alors appelées méthodes)
- Javascript a quelques objets prédéfinis: Array, Boolean, Date, Function, Math, Number, Object, RegExp et String
- Les autres objets sont définis à l'exécution



# STATIQUES, AJOUT, SUPPRESSION DE PROPRIÉTÉS

---

```
function MonObj(a, b) {  
    this.a = a;  
    this.b = b;  
}  
MonObj.staticC = "bleu"; // sur la fonction MonObj!  
    alert(MonObj.staticC); // bleu  
obj = new MonObj('rouge', 430);  
    alert(obj.staticC); // undefined  
obj.c = new Date(); // ajout d'une propriété  
delete obj.b; // suppression d'une propriété  
    alert(obj.b); // undefined  
delete obj; // (rarement utilisé car récupération de mémoire)  
    alert(obj.a); // exception
```



# CONSTRUIRE UN OBJET

---

- Nous avons vu que l'on peut créer un objet 'littéral'.

```
var obj= {a:1,b:function (){alert (this.a);}};
```

- On peut construire un objet avec une fonction. Par convention son nom débute par une majuscule, 'this' dénote l'objet et il faut utiliser 'new'.

```
function Foo(){this.a=1; this.b=function()...;}
```

```
var obj= new Foo();
```

- On accède aux propriétés d'un objet avec le 'point'

```
obj.b();
```

```
alert (obj.a);
```



# MÉTHODES

---

- On appelle méthode une fonction utilisée comme propriété d'un objet.
- Quand une méthode est appelée sur un objet '**o**' (par exemple **o.f()**), '**this**' est initialisé à '**o**', et donc toutes les propriétés de '**o**' sont disponibles (via **this**).
- Une méthode peut être ajoutée ou retirée dynamiquement, pas seulement dans le constructeur



# HÉRITAGE

---

- Un objet est créé sur la base d'un objet prototype ({} par défaut).
- Le prototype est une propriété du constructeur.
- Le prototype possède une propriété 'constructor'
- Un changement de l'objet prototype n'impacte que les instances créées dans le futur
- Un changement de propriété du prototype impacte toutes les instances créées et futures



# EXEMPLE POUR L'HÉRITAGE

---

```
function A() {
    this.foo = function() {alert("A::foo()");}
    this.bar = function() {alert("A::bar()");};
}
a=new A();
function B() { // hérite de A: voir plus bas
    this.foo = function() {alert("B::foo()");};
}
B.prototype = a; // avant toute création de 'B'
B.prototype.constructor = B; // pas toujours nécessaire
b = new B(); // Copie B.prototype vers la propriété cachée 'prototype' de b
a.baz = function() {alert("A::baz()");}
b.bar(); // A::bar() : OK, normal
b.foo(); // B::foo() : OK, normal
b.baz(); // A::baz() : Trouvé dynamiquement car a a été modifié
alert(b.baz == B.prototype.baz); // true
```



# APPEL DU CONSTRUCTEUR DE LA SUPER CLASSE

---

```
function MaClasseMere(p1) { this.attribut1 = p1;}
MaClasseMere.prototype = { methode: function() { ...; } }
function MaClasse(p1, p2) {
    MaClasseMere.call(this, p1);
    this.attribut2 = p2;
    uneMethode = function() {...;} // façon normale
}
MaClasse.prototype = new MaClasseMere();
var obj = new MaClasse("valeur 1", "valeur 2");
obj.methode();
obj.uneMethode();
```



# AFFICHER LE CONTENU D'UN OBJET

---

**JSON.stringify(Votre objet, null, 4)**

- le paramètre 'null' permet de fournir une fonction de filtrage, pour ne pas tout afficher
- le paramètre '4' indique le nombre d'espaces à utiliser pour la tabulation dans le cas d'objets contenant d'autres objets



# EXCEPTIONS

---

```
try {  
    // code pouvant faire 'throw'  
} catch(errorValue) { // optionnel  
    // si une exception a été lancée avec 'throw (...)'  
} finally { // optionnel  
    // code exécuté de toutes façons  
}
```

- Dans un navigateur, on utilisera l'événement 'onerror' pour traiter les exceptions

```
onerror = function (errorValue, url, lineNr) {  
    ...  
    return true;  
};
```



# JAVASCRIPT

DANS LE NAVIGATEUR



# PROPRIÉTÉS RENDUES DISPONIBLES PAR LE NAVIGATEUR

---

Quand Javascript est invoqué dans un navigateur, il a accès à plusieurs propriétés, qui permettent de consulter et d'agir sur le document

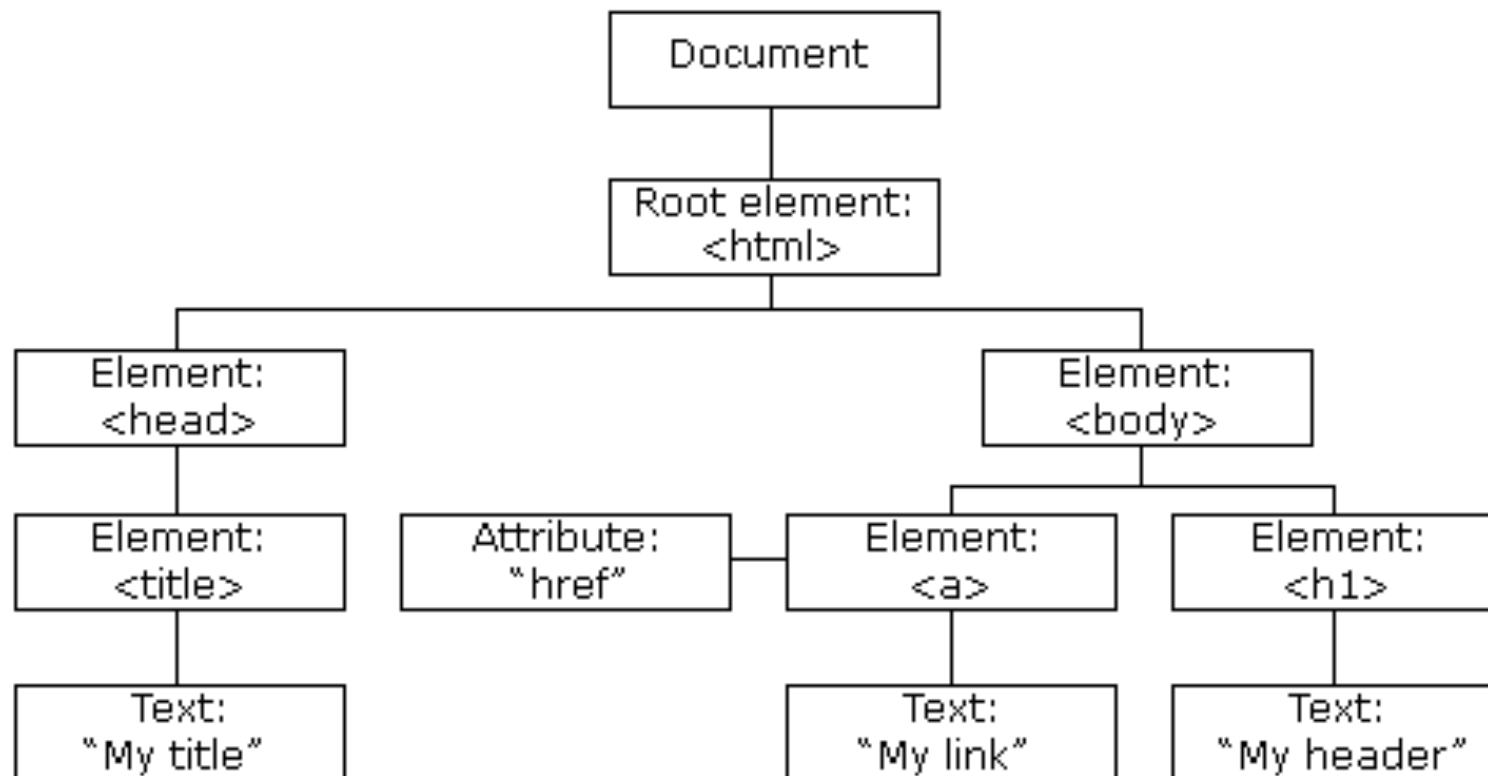
- **document**: les données html de la page
- **window**: les propriétés de la fenêtre
- **screen**: les propriétés physiques de l'écran
- **navigator** : les spécificités du navigateur
- **history** : l'historique de navigation
- **location** : l'adresse de la page concernée



# LE MODÈLE DU DOCUMENT: LE DOM

---

- Javascript accède à et manipule la structure du document html





# DOCUMENT: PROPRIÉTÉS

---

Le 'document object' est le contexte d'un arbre de document. Aucun élément ne peut exister en dehors de ce contexte

- **doctype** : la déclaration doctype, **inputEncoding**, **xmlVersion**
- **documentURI** l'uri du document
- **documentElement** : l'élément racine : **<HTML>**
- **childNodes** : noeuds du document , **firstChild**, **lastChild**
- **nodeName**, **nodeType**, **nodeValue**
- et *async*, *strictErrorChecking*, *xmlStandalone*, *domConfig*, *implementation*, *xmlEncoding*



# DOCUMENT: MÉTHODES DE MANIPULATION HTML

---

- `createElement()`, `createAttribute(name)`
- `createTextNode()`
- `getElementById(id)`, `getElementsByTagName()`,  
`getElementsByClassName()`
- `renameNode()`

*et adoptNode(sourcenode), importNode(nodetoimport,deep),  
getElementsByTagNameNS(), createElementNS(),  
createAttributeNS(uri,name), createCDATASection(), createComment(),  
createDocumentFragment(), createEntityReference(name),  
createProcessingInstruction(target,data), normalizeDocument()*



# TROUVER LES ÉLÉMENTS

---

```
var i=document.getElementById("..");  
var t=d.getElementsByTagName("..");  
var c=d.getElementsByTagName("..");
```



# NODELIST

---

- Les fonctions d'accès groupé aux éléments d'une page (**getElement<sup>s</sup>**) renvoient des **NodeList**
- Propriété:
  - **length** : le nombre d'éléments (mis à jour en direct)
- Méthode:
  - **item ( idx )**: élément à la position i
- équivalent à
  - **list[idx]**



# MODIFIER LE DOCUMENT

---

- le flux html du document

```
document.write(Date());
```

- le contenu d'un élément html (il suffit de modifier la valeur de la propriété)

```
var el= document.getElementById(id);
```

```
el.innerHTML="...";
```

- la valeur d'un attribut d'un élément html

```
el.src="...";
```



# CHANGER LE STYLE

---

- `element.style.property=...`

```
<!DOCTYPE html><html><body>
```

```
<h1 id="t">Titre</h1>
```

```
<button type="button"
```

```
onclick="document.getElementById('t').style  
color='red' ">
```

```
Appuyez!</button>
```

```
</body></html>
```



# PROPRIÉTÉS DE NODE

---

Element hérite de Node. Chaque node peut être exploré (attributs, texte) et donne accès à ses voisins et parents

- **attributes, textContent**
- **childNodes, parentNode**
- **firstChild, lastChild**
- **nextSibling, previousSibling**
- **nodeName,.nodeType , nodeValue**
- **ownerDocument** : l'objet document qui contient l'élément
- et *prefix, baseURI, localName , namespaceURI*



# MÉTHODES DE NODE

---

Chaque Node peut être modifié

- **appendChild(), removeChild(), replaceChild(),**
- **insertBefore()**
- **cloneNode()**

Et questionné:

- **hasAttributes(), hasChildNodes()**
- **isEqualNode(), isSameNode()**
- *et getUserData(key), setUserData(key,data,handler), isSupported(),  
lookupNamespaceURI(), lookupPrefix(), normalize(), compareDocumentPosition(),  
isDefaultNamespace(), getFeature(feature,version)*



# EXEMPLE - AJOUT D'UN PARAGRAPHE

---

```
<div id="d1">
```

```
  <p id="p1">Premier pa
```

```
  <p id="p2">Second par
```

```
</div>
```

```
<script>
```

```
  var para=document.createElement("p");
```

```
  var node=document.createTextNode("Troisième...");
```

```
  para.appendChild(node);
```

```
  var document.getElementById("d1").appendChild(para);
```

```
</script>
```

Premier paragraphe.

Second paragraphe.

Troisième paragraphe, ajouté au vol.



# API DE ELEMENT

---

Element ajoute à Node la prise en compte des attributs HTML

- **getElementsByTagName(nom)**
- **hasAttribute(nom)**
- **getAttribute(nom)** // la valeur de l'attribut
- **setAttribute(nom,v)** // change la valeur
- **removeAttribute (nom)**
- **getAttributeNode(nom)** // retourne le noeud 'attribute'
- **setAttributeNode(node)** // crée ou remplace le noeud
- **removeAttributeNode(node)** // supprime et retourne le noeud



# PROPRIÉTÉS DE DOCUMENT

---

Document donne accès au contenu de la page HTML

- **body, title**
- **anchors, applets, cookie, forms, images, links**
  - listes de ces éléments
- **readyState** : état de chargement ('loading', 'interactive', 'complete')
- **URL, domain** : informations sur l'adresse
- **referrer**: le site précédant la page courante (d'ou l'on vient)



# MÉTHODES DE DOCUMENT

---

- `getElementsByName()`
- `write()` écrit dans le document
- `writeln()` écrit avec un retour à la ligne
- `open()` ouvre un flux pour l'écriture (pour rediriger `write`, `writeln`)
- `close()` ferme le flux ouvert avec `document.open()`



# PROPRIÉTÉS DE WINDOW

---

(les propriétés soulignées sont modifiables)

- name
- document, navigator, location, history, screen : l'objet associé
- opener, parent, top, self : la window correspondante
- closed, status, defaultStatus
- innerHeight, innerWidth, outerHeight, outerWidth
- frames, length : les, le nombre de frames/iframes dans une fenêtre
- pageXOffset, pageYOffset
- screenLeft, screenTop, screenX, screenY : coordonnées relatives à l'écran



# MÉTHODES DE WINDOW

---

- `alert()` , `confirm()`, `prompt()`, `createPopup()`
- `open()` , `close()`
- `focus()`, `blur()`
- `print()` // `imprime`
- `resizeBy()` , `resizeTo()` // pixels, taille cible
- `moveBy()`, `moveTo()` // pixels, taille cible
- `scrollBy()`, `scroll()` // pixels, taille cible
- et *`clearInterval()`*, *`clearTimeout()`*



# HISTORY: PROPRIÉTÉS ET MÉTHODES

---

Cet objet conserve l'historique de la navigation

- **length** : le nombre d'URL mémorisées
- **back()** : revient en arrière
- **forward()** : avance
- **go()** : charge un élément de l'historique



# SCREEN

---

- `availHeight`, `availWidth` : espace disponible
- `colorDepth` : précision de la palette de couleurs
- `height`, `width` : dimensions totales de l'écran
- `pixelDepth` : précision des couleurs de l'écran en bits par pixel



# JAVASCRIPT

ÉVÉNEMENTS



# ÉVÉNEMENTS SOURIS

---

- **onclick, ondblclick** : (double) click sur un élément
- **onmousedown, onmouseup**: bouton enfoncé, relaché sur un élément
- **onmousemove** : souris bougée sur un élément
- **onmouseover , onmouseout** : souris entrant, sortant sur un élément



# ÉVÉNEMENTS CLAVIER ET FENÊTRE

---

- Événements clavier
  - **onkeydown** , **onkeyup**, **onkeypress**
- Événements fenêtre
  - **onload** (document, frameset, <object> chargé),
  - **onresize**, **onscroll**
  - **onunload**
  - **onabort** (<img> arrêtée - <object>),
  - **onerror** (erreur <img> - <object>, <body>),



# EXAMPLE: BODY ONLOAD

---

```
<html><head><script>
function load(){
    alert("La page est chargée");
}
</script></head>
<body onload="load()">
<h1>Bonjour!</h1>
</body></html>
```



# AGIR AVANT ONLOAD

---

- Utile lorsqu'une page attend de nombreuses ressources (images) après être chargée

```
// alternative à DOMContentLoaded
```

```
document.onreadystatechange = function () {  
    if (document.readyState == "interactive") {foo(); }  
}
```

```
// alternative à onload
```

```
document.onreadystatechange = function () {  
    if (document.readyState == "complete") {bar();}  
}
```



# INSTALLATION DES ÉVÉNEMENTS

---

- Via l'attribut correspondant, et this est initialisé à l'objet qui reçoit l'événement

```
<div onmouseover="mOver(3,this)"  
onmouseout="mOut(this)">Venez avec la souris</div>
```

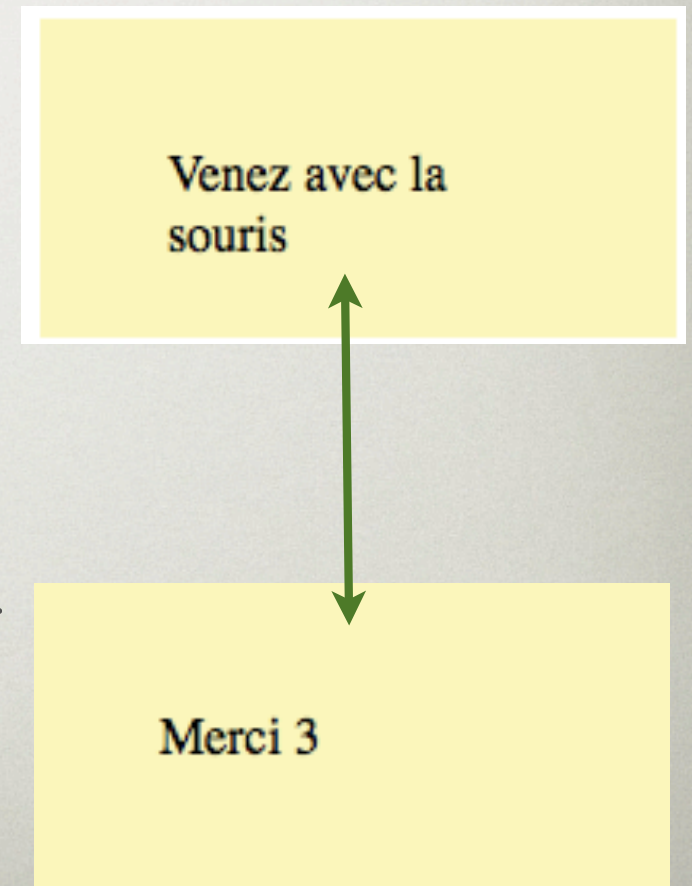
```
<script>
```

```
var old;
```

```
function mOver(d,obj) {  
    old=obj.innerHTML;  
    obj.innerHTML="Merci "+ d;  
}
```

```
function mOut(obj){obj.innerHTML=old;}
```

```
</script>
```





# INSTALLATION DES ÉVÉNEMENTS PAR PROGRAMME

---

```
<div onmouseover="mOver(3,this)">Venez avec la  
souris</div>
```

```
<script>
```

```
var old;
```

```
function mOver(d,obj) {
```

```
    old=obj.innerHTML;
```

```
    obj.innerHTML="Merci "+ d;
```

```
    obj.onmouseout=function() {mOut(obj); }
```

```
}
```

```
function mOut(obj){obj.innerHTML=old; }
```



# EVÉNEMENTS TEMPORELS

---

- Javascript permet d'appeler une fonction de manière répétée selon un intervalle de temps donné,

```
var inter; //intervalle
```

```
inter=window.setInterval(fn,milliseconds);
```

- de stopper les appels

```
window.clearInterval(inter);
```

- ou une fois après un certain délai

```
window.setTimeout(fn,milliseconds);
```

```
window.clearTimeout(timeout);
```



# JAVASCRIPT

LES FORMULAIRES, ET UN RETOUR VERS HTML



# FORMULAIRES

---

- Les formulaires permettent à l'utilisateur de fournir des données au site
- Ils sont donc un lieu essentiel d'interaction et d'événements
- HTML prévoit un jeu de balises spécifiques à cet usage, organisées autour de <FORM>



# FORMULAIRES

---

- **onsubmit** (du formulaire)
- **onselect** (l'utilisateur entre du texte: pour `<input>`, `<textarea>`)
- **onchange** (l'état d'un objet est modifié: pour `<input>`, `<select>`, et `<textarea>`)
- **onreset** (du formulaire)
- **onfocus** (d'un élément quelconque)
- **onblur** (perte de focus d'un élément quelconque)



# LA BALISE <FORM>

---

- **<form>** est un élément permettant la saisie de données par l'utilisateur
- Habituellement, ces données sont soumises (submit) au site pour faire des calculs

**<form>**

... elements de type **<input>**

**</form>**



# LA BALISE <INPUT> ET SES ATTRIBUTS

---

- **name**: le nom de l'input (sera utilisé par le serveur)
- **type**: présentés plus loin
- **value**: la valeur d'un input qui sera envoyée au serveur
- **checked**: bouton radio ou checkbox pré-sélectionné
- **disabled**: input désactivé
- *et: autofocus (au chargement), autocomplete, ...*



# L'ATTRIBUT 'TYPE' DE <INPUT>

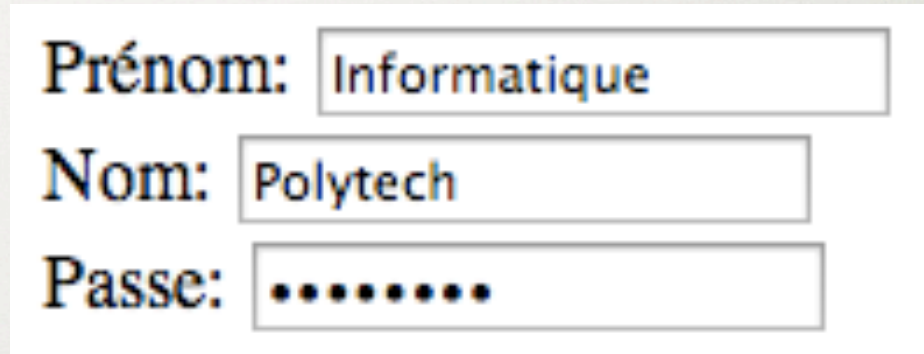
---

- **button, submit, reset** : envoi, reset du formulaire
- **text, password, number** : texte monoligne
- **checkbox, radio** : cases à cocher
- **hidden** : pour créer des éléments invisibles (utiles au serveur)
- **file** : pour envoyer les fichiers
- et *color, date, datetime, datetime-local, email, image, month, range, search, tel, time, url, week*



**<INPUT TYPE="TEXT">,  
"PASSWORD"**

---



Prénom: Informatique  
Nom: Polytech  
Passe: .....

**<form>**

**Prénom: <input type="text" name="pn"><br>**

**Nom: <input type="text" name="nm"><br>**

**Passe: <input type="password" name="pwd">**

**</form>**



# RADIO, CHECKBOX, SUBMIT

---

```
<form>
```

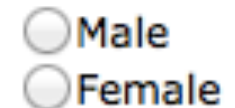
```
<input type="radio" name="sex" value="male">Male
```

```
<input type="radio" name="sex" value="female">Female
```

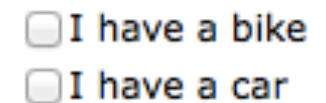
```
<input type="checkbox" name="vehicle" value="Bike">I have a bike
```

```
<input type="checkbox" name="vehicle" value="Car">I have a car
```

```
</form>
```



A form containing two radio buttons. The first radio button is selected and is followed by the text "Male". The second radio button is unselected and is followed by the text "Female".



A form containing two checkboxes. The first checkbox is unselected and is followed by the text "I have a bike". The second checkbox is unselected and is followed by the text "I have a car".

```
<form name="input" action="html_form_action.asp" method="get">
```

```
Username: <input type="text" name="user">
```

```
<input type="submit" value="Submit">
```

```
</form>
```



A form containing a text input field labeled "Username:" and a submit button labeled "Submit".



# LISTES DÉROULANTES AVEC <SELECT>

---

```
<!DOCTYPE html><html><body>
```

```
<form action="">
```

```
<select name="autos">
```

```
<option value="volvo">Volvo</option>
```

```
<option value="saab">Saab</option>
```

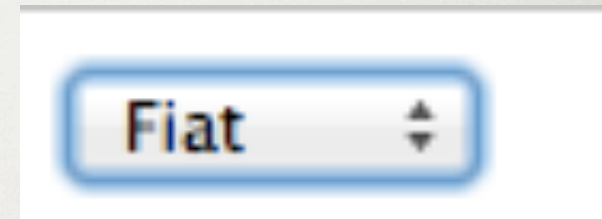
```
<option value="fiat" selected>Fiat</option>
```

```
<option value="audi">Audi</option>
```

```
</select>
```

```
</form>
```

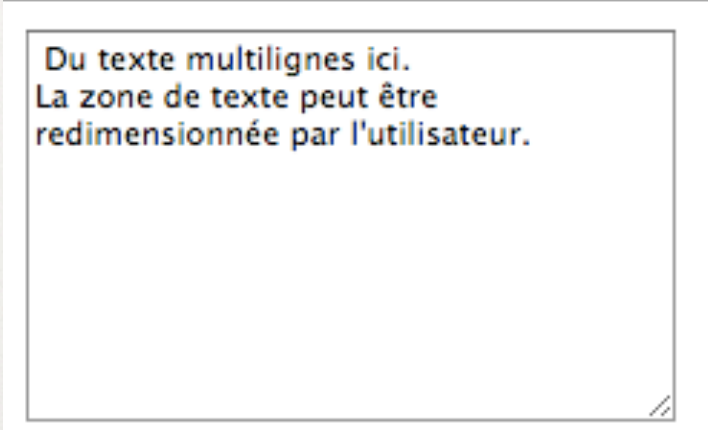
```
</body></html>
```





# <TEXTAREA>

---



Du texte multilignes ici.  
La zone de texte peut être  
redimensionnée par l'utilisateur.

```
<form><textarea rows="10" cols="30"> Du  
texte multilignes ici.
```

```
La zone de texte peut être  
redimensionnée par l'utilisateur.</  
textarea>
```

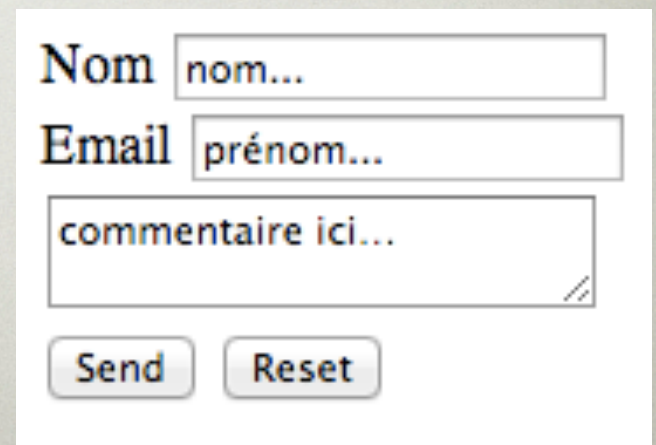
```
</form>
```



# EX: ENVOYER UN EMAIL

---

```
<form action="MAILTO:jim@aol.com" method="post"
enctype="text/plain">
<label for="nm">Nom</label>
<input id="nm" type="text" name="nom" value="nom..."><br>
<label for="em">Email</label>
<input id="em" type="text" name="mail" value="mail..."><br>
<textarea name="commentaire" >commentaire ici...</
textarea><br>
<input type="submit" value="Send">
<input type="reset" value="Reset">
</form>
```



Nom

Email



# AJAX

---

- Javascript permet de modifier du contenu dans une page sans la recharger, en faisant une requête distante
- On utilise pour cela l'objet **XMLHttpRequest**
- Permet que les requêtes soient asynchrones



# AJAX EN UN TRANSPARENT

---

```
var xmlhttp;

if (window.XMLHttpRequest) { // tout sauf IE
    xmlhttp=new XMLHttpRequest();
} else { // IE6, IE5
    xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
}

xmlhttp.onreadystatechange=function(){
    if (xmlhttp.readyState==4 &&xmlhttp.status==200){
        document.getElementById("monDiv").innerHTML=xmlhttp.responseText;
    }
}

xmlhttp.open("GET","monurl.fr",true);
xmlhttp.send();
```



# CONCLUSION SUR JAVASCRIPT

---

- Le langage de la dynamique des pages web côté client
- Un langage aujourd'hui doté de très riches fonctionnalités et de plus en plus performant
- Complète CSS pour définir des transitions riches et fluides