# Cours Web

# JavaScript

Lionel Seinturier

Université Pierre & Marie Curie

Lionel.Seinturier@lip6.fr

11/7/02



# 12. JavaScript

## JavaScript

Programme Java s'exécutant **côté client Web** (ie dans le navigateur)

applet prog. "autonome" stocké dans un fichier `.class`
JavaScript prog. **source** embarqué dans une page `.html`

| | côté client | côté serveur |
|---|---|---|
| .class autonome | applet | servlet |
| embarqué dans .html | JavaScript | JSP |

## JavaScript

(quasiment) même syntaxe que Java mais
- langage interprété (par le navigateur)
- pas de classe, pas d'héritage, pas de typage, API moins riche



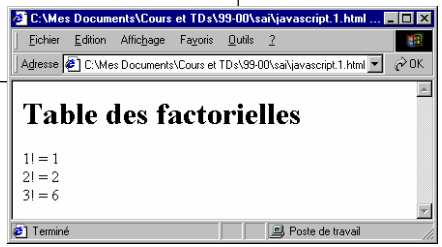
# 12. JavaScript

## JavaScript

- Java **embarqué** dans une page HTML entre les balises `<SCRIPT>` et `</SCRIPT>`
- le **chargement** de la page provoque l'**exécution** du code JavaScript
- le script JavaScript génère dynamiquement du **code HTML**

```
<HTML> <BODY>
<H1>Table des factorielles</H1>
<SCRIPT LANGUAGE="JavaScript">
    for ( i=1 , fact=1 ; i<4 ; i++ , fact *= i ) {
      document.write( i + "! = " + fact + "<BR>" );
    }
</SCRIPT>
</BODY> </HTML>
```

chargement de la page






# 12. JavaScript

## Principe de fonctionnement

```
<HTML> <BODY>
<H1>Table des factorielles</H1>

<SCRIPT LANGUAGE="JavaScript">
for ( i=1 , fact=1 ; i<4 ; i++ , fact *= i ) {
  document.write( i + "! = " + fact + "<BR>" );
}
</SCRIPT>

</BODY> </HTML>
```

du code JavaScript

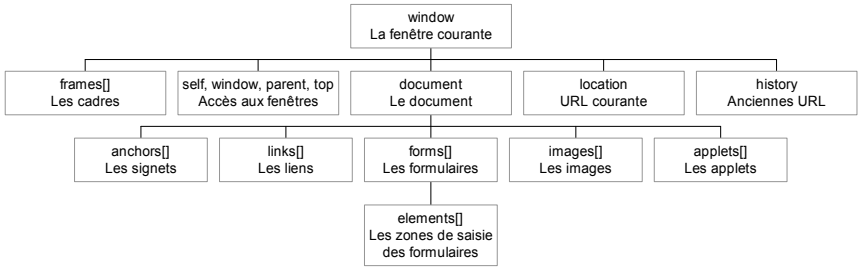
résultat = **HTML** généré via la fonction **write**

ce qu'affiche le navigateur

```
<HTML> <BODY>
<H1>Table des factorielles</H1>
1! = 1<BR>
2! = 2<BR>
3! = 6<BR>
</BODY> </HTML>
```

## Hiérarchie des éléments d'une page HTML

• hiérarchie de (pseudo)-objets JavaScript ⇨ représentation du contenu HTML
• consultables / modifiables ⇨ génération dynamique de contenu HTML








## Hiérarchie des éléments d'une page HTML

Exemple d'accès aux éléments d'une page HTML

```
<FORM ACTION="..." NAME="formulaire" METHOD=POST>

Nom <INPUT NAME="nom" SIZE=46> <P>
Prénom <INPUT NAME="prenom" SIZE=40> <P>
<INPUT TYPE=SUBMIT VALUE="Envoi">
<INPUT TYPE=RESET VALUE="Remise à zéro"> <P>

</FORM>
```

Lecture/écriture de la variable `document.forms[0].elements[1].value`
≡ consultation/modification automatique du contenu du champ `prenom`

Utilisation de façon alternative des attributs `NAME`
`document.formulaire.prenom.value`





## Hiérarchie des éléments d'une page HTML

Nombreuses variables et fonctions disponibles sur chaque élément

- *Form*`.submit()`
  force l'envoi des données du formulaire (≡ clic sur bouton *submit*)
- *Input*`.focus()`
  amène le curseur de saisie dans le champ
- `window.setInterval(` *fonction*`, nbMilliSecondes )`
  exécute un traitement périodiquement
- `window.alert( "message" )`
  affiche un message dans une boîte de dialogue
- `window.confirm( "message" )`
  affiche une boîte de dialogue oui/non
- `window.prompt( "message", "valeur par défaut" )`
  affiche une boîte de dialogue permettant de saisir une valeur





## Gestion d'événements

Interaction avec l'utilisateur via des **gestionnaires d'événements**

• événements gérables : clavier/souris
• ajout de gestionnaires sur des éléments du document HTML
  ⇨ ajout d'attributs (`onClick="..."`, `keyPressed="..."`, ...)
  sur des balises HTML (`<INPUT TYPE=SUBMIT onClick="...">`, ...)
• fonctions JavaScript (**définies par le programmeur**) associée aux gestionnaires
  ⇨ `<INPUT TYPE=SUBMIT onClick="`**`onAClique()`**`">`

⇨ appel de la fonction lorsque l'événement survient
⇨ exécution

# 12. JavaScript

## Type d'événements

Souris `onClick`, `onDblClick`, `onMouseDown`, `onMouseUp`
`onMouseOver`, `onMouseOut`
Clavier `onKeyPressed`, `onKeyDown`, `onKeyUp`
Mixte `onFocus`, `onBlur`

Pour les formulaires `onSubmit`, `onReset`
Pour les zones de saisie des form. `onChange`
Pour les documents et les images `onLoad`, `onUnload`, `onAbort`

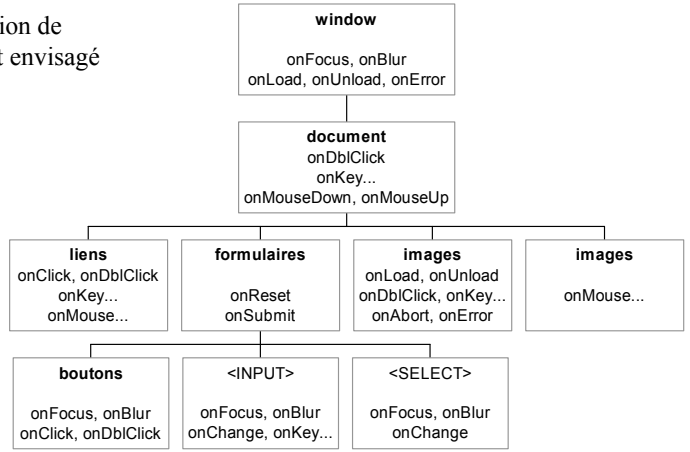
- `onMouseDown` : appui sur le bouton de la souris
- `onMouseUp` : relachement du bouton de la souris
- `onMouseOver` : arrivée sur une zone
- `onMouseOut` : départ de la zone
- `onFocus` : lorsque le curseur de saisie arrive dans une zone
- `onBlur` : lorsque le curseur de saisie repart d'une zone



# 12. JavaScript

## Type d'événements

En fonction de
l'élément envisagé






# 12. JavaScript

## Syntaxe

Types de base
nombre (pas de distinction entier, réel), chaîne de caractères, booléen

Langage faiblement typé
une variable peut à tout moment changer de type
le type est déterminé au moment de l'affectation

Syntaxe proche de Java
quasiment les mêmes mots-clés et opérateurs
**mais pas** de classe, ni d'héritage, ni d'exception, ni d'interface

Quelques pseudo-objets prédéfinis (manipulation de dates, tableaux, calculs)

Entrées/sorties rudimentaires (clavier, écran), pas de fichiers, ni de réseau

Langage sensible à la casse (var ≠ VAR)

Commentaires (idem Java, C/C++) /* */ ou //



# 12. JavaScript

## Déclaration de variables

• affectation d'une valeur à un identificateur
• utilisation du mot clé `var`

```
x = 24.5;  var y;
```

Nombres entiers signés, héxadécimaux (`0x..`), octal (`0..`), réels
Booléens `true` ou `false`
Chaînes `'...'` ou `"..."` ASCII 8 bits + `\" \' \n \t \\`

Opérations sur les chaînes de caractères (∃ nombreuses autres)

- `maChaine.length` : longueur de la chaîne
- `maChaine.charAt(`*i*`)` : *i*ème caractère de la chaîne
- `maChaine.indexOf(`*ch*`,`*start*`)` : indice de *ch* à partir de *start*
- `maChaine.substring(`*from*`,`*to*`)` : sous-chaîne de *from* (inclus) à *to* (exclus)
- `maChaine.split(`*delim*`)` : tableau de sous-chaînes séparées par *delim*

## Tableaux

- commencent à l'indice 0
- accès aux éléments avec `[]` `monTab[12]`
- type `Array`
- peuvent contenir des éléments de types hétérogènes
- leur taille peut être augmentée à la demande

Déclaration

```
  t = new Array( 'garage', '', 2 );
≡ t = [ 'garage', '' , 2 ];
≡ t = new Array();  t[0]='garage'; t[1]=''; t[2]=2;
≡ t = new Array(3); t[0]='garage'; t[1]=''; t[2]=2;
```

Tableaux multi-dimensionnels

```
matrice[i][j] = ...
```





## Tableaux

Opérations sur les tableaux

- `t.length` : taille du tableau
- `t.concat(t1,...)` : concaténation de *t*, *t1*, ...
- `t.join(sep)` : concaténation des éléments de *t* séparés par *sep*
- `t.reverse()` : renverse les éléments de *t*
- `t.slice(from,to)` : sous-tableau de *from* (inclus) à *to* (exclus)
- `t.sort()` : tri

- ces opérations retournent un résultat
- le tableau initial (*t*) est inchangé (ce n'est pas un objet au sens Java)

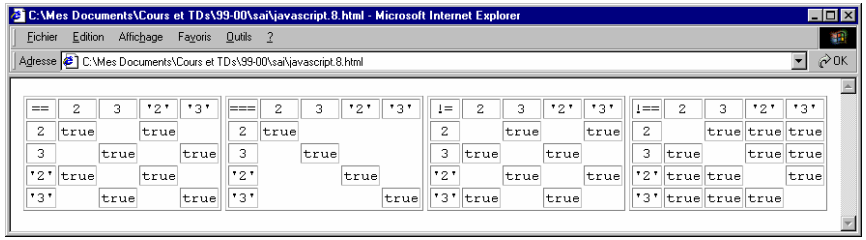




## Opérateurs

Comparaison `== != === !== < > <= >=`

| | | |
|---|---|---|
| `==` `!=` | test d'identité (de ¬ id.) | si nécessaire **conversion de type** |
| `===` `!==` | test d'**égalité** (de ≠) | pas de conversion |




Arithmétiques `+ - * / % ++ -- -`

division entre entiers donne un réel (≠ Java, C/C++)

JavaScript `1/2=0.5` Java, C/C++ `1/2=0`





## Opérateurs

Manipulation de bits `& | ^ ~ << >> >>>`

- `^` XOR
- `~` NOT
- `>>` décalage à droite en conservant le signe
- `>>>` décalage à droite en ajoutant des 0

Affectations `= += -= *= /= <<= >>= >>>= &= ~= |=` ternaire `?:`

Logiques `&& || !`

Sur les chaînes `+ +=`

Sur les variables `typeof variable`

retourne `'number'`, `'boolean'`, `'string'`, `'object'`, `'function'`

ou `'undefined'` selon le **type de la variable**

# 12. JavaScript

## Instructions

Identiques Java, C/C++

Conditions
```
if (condition) { ... } else { ... /* facultatif */ }
switch (expression) {
  case constante : ... break;
  ...
  default : ...
}
```

Boucles
```
for ( initialisation ; test ; increment ) { ... }
while (condition) { ... }
do { ... } while (condition);
```

Déroutements
`break` : interruption `for`, `case`, `while` ou `do/while`
`continue` : passage itération suivante `for`

Fonction prédéfinie `eval('expression JavaScript')`
évalue l'expression JavaScript passée en paramètre



# 12. JavaScript

## Fonctions

Définition identique aux fonctions C
**mais** pas de typage des arguments, ni de la valeur de retour

```
function factorielle(n) {
    if (n<2) return 1; else return n*factorielle(n-1);
}
```

Non déclaration exhaustive des arguments possible
⇨ accès par le tableau prédéfini `arguments`

```
function plus() {
    for ( i=0,s=0 ; i<arguments.length ; i++ ) {
        s += arguments[i];
    }
    return s;
}
```

Panachage déclaration arg. / non déclaration possible



# 12. JavaScript

## Fonctions

Appels imbriqués de fonctions possibles
!! Attention: les variables ont une portée globale !!

```
function f1(n) {
    x = 12;
    f2(14);
    // x vaut 13 maintenant
}

function f2(n) {
    // x vaut 12
    x = 13;
}
```



# 12. JavaScript

## Pseudo-objets JavaScript

But identique aux structures (`struct`) C
⇨ stocker des données dans un enregistrement

Déclaration

```
objet = { propriete1: valeur1, ... , proprieten: valeurn };
monRectangle = { longueur: 4.5, largeur: 2 };
```

Accès aux propriétés des objets `objet.propriete`

Itération sur les propriétés d'un objet

```
for ( variable in objet ) { ... }
for ( i in monRectangle ) { document.write(i); }
```
⇨ affichage : 4.5 2