# JavaScript, langage et utilisation

## Introduction

- Langage côté **CLIENT**
- Javascript n'est **pas** java !
- intégré aux navigateurs: pas de problème de déploiement comme avec les applets java ou flash
- permet de rendre les applications plus intéractives

Les inconvénients :

- problèmes de standardisation (tendent à se réduire)
- pas forcément *activé* (c.f. extension NoScript pour firefox) ;
- un robot (ex. google) ne sais pas naviguer en utilisant JavaScript.

Les règles :

- un site doit pouvoir fonctionner **sans javascript**.
- le code doit tourner sur les principaux navigateurs et les principales plateformes (attention aux macs !)
- pour un intranet avec une politique logicielle stricte, les choses sont plus faciles : le navigateur est connu.
- privilégiez le code simple.
- Utilisez des bibliothèques!

## Liens utiles

- http://www.ecma-international.org/publications/standards/Ecma-262.htm Spécifications d'ECMAScript, version "standardisée" de javascript
- Webdeveloper : http://chrispederick.com/work/webdeveloper/ ; extension de firefox très pratique.
- www.quirksmode.org : site sur la compatibilité entre navigateurs. (en particulier http://www.quirksmode.org/dom/compatibility.html).
- http://www.w3schools.com/ documentations diverses.

## Un petit exemple pour commencer

```
<script type="text/javascript">
function calculerLaSomme() {
  var n1= parseInt(document.getElementById('nombre1Ex1').value);
  var n2= parseInt(document.getElementById('nombre2Ex1').value);
  document.getElementById('resultatEx1').value= n1+n2;
}
</script>
<p>nombre 1 <input type="text" name="nombre1Ex1" id="nombre1Ex1"></p>
<p>nombre 2 <input type="text" name="nombre2Ex1" id="nombre2Ex1"></p>
<p>Résultat <input type="text" name="resultatEx1" id="resultatEx1"></p>
<button type="button" onclick="calculerLaSomme()">calculer la somme</button>
```

nombre 1

nombre 2

Résultat

calculer la somme

# Le même avec une table...

```
<script type="text/javascript">
function calculerLaSomme_1() {
  var n1= parseInt(document.getElementById('nombre1Ex1_bis').value);
  var n2= parseInt(document.getElementById('nombre2Ex1_bis').value);
  document.getElementById('resultatEx1_bis').value= n1+n2;
}
</script>
<table>
<tbody><tr><td>nombre 1</td><td> <input type="text" name="nombre1Ex1" id="nom
<tr><td>nombre 2</td><td> <input type="text" name="nombre2Ex1" id="nombre2Ex1
<tr><td>Résultat</td><td> <input type="text" name="resultatEx1" id="resultatE
</tbody></table>
<button type="button" onclick="calculerLaSomme_1()">calculer la somme</button
```

nombre 1

nombre 2

Résultat

calculer la somme

# Javascript dans une page web

Comment inclure du code dans une page ?

Plusieurs solutions :

- La balise `script`
- appels liés à des événements
- Les pseudo-liens javascript

## La balise script

- Le code javascript inclus dans la balise script est exécuté quand il est lu.
- Généralement, on l'utilise pour définir des fonctions.
- Deux possibilités :
  1. code javascript écrit dans la page html:

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

```
... code javascript
</script>
```

(à réserver à du code court et non réutilisable)

2. inclusion de code, pour tout ce qui est réutilisable :

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

### Appels javascript lié à des événements

- On peut demander qu'un bout de code javascript soit exécuté lors de certains événements:
  - on a pressé un bouton donné (onclick)
  - le pointeur de la souris est passé sur un élément (onmouseover)
  - on a pressé une touche (onkeyup,...)
  - la page a terminé de se charger (onload)
- Le code est alors écrit comme la valeur d'un attribut de l'élément concerné: clickez moi

```
<button onclick="alert('click !')">clickez moi</button>
```

- On écrira du code très court. idéalement un appel de fonction.

## Constructions de base

Le code JavaScript est une suite *d'instructions*, dont certaines peuvent être des *déclarations de fonctions* ou des *affectations*.

Les instructions sont séparées par des passage à la ligne ou par le caractère ";"

On suggère d'utiliser systématiquement ";", mais le passage à la ligne reste toujours un séparateur

On peut écrire :

```
euros= 6.56* francs; alert(euros);
```

ou

```
euros= 6.56* francs
alert(euros)
```

ou

```
euros= 6.56* francs;
alert(euros);
```

## Variables en javascript

- Les variables en javascript sont dynamiques et faiblement typées.
- Si une variable n'existe pas, lui donner une valeur la déclare automatiquement

- Une variable peut changer de type:
  montrer a

```
<script type="text/javascript">
        a= 'salut';
        a= 3;
        a= a + 2;
        a= a+7;
</script>
<button onclick="alert('valeur de a '+ a)">montrer a</button>
```

- Il est conseillé de déclarer explicitement les variables à l'aide du mot-clef `var` :

```
<script type="text/javascript">
        var a;
</script>
```

  Notez que ça ne fixe pas son type pour autant.

# Types de base

- nombres (un seul type pour les nombres réels et les entiers)
- booléens
- string
- Deux valeurs spéciales pour les objets : `null` et `undefined`.

noter l'absence d'un type caractère !

# Quelques opérations sur les types de bases

## Les nombres

on dispose entre autres des opérateurs +, -, *, /, % (modulo)

note : pour la division euclidienne de x par y, on fera :

```
q= Math.floor(x/y);
```

(Math.floor() calcule la partie entière).

Quand un calcul donne un résultat infini, on a une valeur spéciale : `Infinity`

```
a= -1/0;
alert (a==-Infinity); // affiche "true"
```

Quand un résultat n'est pas calculable (racine carrée de -1 ou 0/0), on obtient NaN ("Not a number").

NaN est le seul nombre... pas égal à lui même. Pour le tester, il faut utiliser isNaN() :

```
a= 0/0;
alert (a==NaN); // affiche false
alert(isNaN(a)); // affiche true
```

## Les booléens

Opérateurs booléens usuels && (et), ! (not), || (ou).

## Les strings

La déclaration de string peut se faire en utilisant des guillements simples '...' ou doubles "....".

```
s1= "un texte";
s2='un autre texte';
```

L'opérateur "+" permet de concaténer deux string, voire une string et autre chose (un nombre par exemple).

```
nom="Turing";
salut="bonjour monsieur ";
complet= salut + nom;
// complet vaut "bonjour monsieur Turing"
x= 3*4;
reponse="la valeur calculée est "+ x + "!!!";
// reponse vaut "la valeur calculée est 12!!!
```

# Comparaisons

## Égalité

Deux comparateurs disponibles : "==" qui sait faire des conversions, et "===" (trois signe égal) qui n'en fait pas.

**`a === b`**

si a et b ont le même type, et correspondent à la même valeur (pour les objets, s'ils ont la même adresse).

**`a == b`**

si, après conversion éventuelle, a et b ont la même valeur. Les règles de conversions sont complexes.

si a ou b est un nombre, et que l'autre est une string, on converti la string en nombre, et on compare les

valeurs

ainsi, les comparaisons suivantes sont vraies :

```
"01" == "01" (deux chaînes identiques)
1 == "01" (la chaîne "01" est convertie en nombre, et vaut 1)
```

En revanche :

```
"01" != "1" (deux chaînes différentes)
```

# Comparaisons

## Ordre

On a les opérateurs usuels < , <=, >, >=

ils fonctionnent sur les nombres

sur les strings: ordre lexicographique

# Conversions automatiques

Règles complexes et peu intuitives :

- 1 + '0' -> '10'
- 1 * '0' -> 0
- 1+ true -> 2

Suggestion: se contenter de la concaténation; pour les autres : utiliser les conversions explicites

```
s="15";
a= 2 * parseInt(s);
alert(a); // affiche 30
```

## les chaînes

- sont des objets
- constantes déclarées entre "..." *ou* entre '...'.
- ont une propriété, `length` (leur longueur)
- se comparent avec "==" !!!
- de nombreuses méthodes:
  - `charAt(pos)` retourne une String
  - `charCodeAt(pos)` retourne le code **unicode** du caractère (si inférieur à 65535)

- `split(separateur)` : renvoie le tableau obtenu en découpant la chaîne.

```
s= "un,deux,trois";
t= s.split(",");
```

- indexOf(sousChaine,index), indexOf(sousChaine): premier indice de la sous-chaîne, à partir de l'index donné, ou -1.

```
s= "un,deux,trois";
i= s.indexOf("tr"); // i vaut 8
```

- substr(debut,longueur) : renvoie la sous-chaîne démarrant à "debut", de longueur "longueur".
- toLowerCase() : renvoie la chaîne en minuscules ;
- toUpperCase() : renvoie la chaîne en majuscules.
- String.fromCharCode(c1,c2....) : permet de passer d'une suite de codes ascii à la chaîne correspondante.

# Chaînes et nombres

Deux fonctions de javaScript sont particulièrement intéressantes:

- parseInt(CHAINE) : analyse chaîne comme un entier. Renvoie la valeur NaN (not a number) en cas d'échec.
- parseFloat(CHAINE) : analyse chaîne comme un entier. Renvoie la valeur NaN en cas d'échec.

Notez que ces deux fonctions sont très permissives. Elles acceptent des chaînes qui *commencent* par un nombre, et, s'il y a du texte après, elle l'ignorent. Pour tester si elles ont réussi, on utilisera la fonction isNaN().

```
s="1234";
i= parseInt(s);
if (isNaN(i)) {
        alert(s + " n'est pas un nombre");
}
```

# les tableaux (Array)

- doivent être créés, mais on n'a pas besoin de fixer leur taille:

```
tab= new Array(); // Création du tableau
tab[0]= 2;
tab[1]= 3;
s= 0;
for (i= 0; i < tab.length; i++) {
        s= s + tab[i];
}
```

- peuvent être initialisés à la création :

```
tab= new Array("un", "deux", "trois");
```

- longueur donnée par l'attribut `length`

- accès aux cases par la notation habituelle.
- passage d'un tableau à une chaîne de caractères par "join": l'argument de join est inséré entre les éléments.

```
s= tab.join(""); // s vaut undeuxtrois
s= tab.join(":"); // s vaut un:deux:trois
```

- On peut leur ajouter une case avec la commande push :

```
tab= new Array("un", "deux", "trois");
tab.push("quatre");
// tab[3] vaut maintenant "quatre"
```

- inversement, pop() supprime le dernier élément du tableau, et le retourne:

```
tab= new Array("un", "deux", "trois");
a= tab.pop();
// tab vaut ["un", "deux"] et a vaut "trois"
```

# Tableaux associatifs

On a introduit ensuite la notion de tableau associatif. Ils associent une clef à une valeur.

```
var t= {};
t["java"]= "langage de programmation orienté objet";
t["lisp"]= "langage de programmation fonctionnel";
```

Autre méthode de création :

```
var t= {"java": "langage de programmation orienté objet",
        "lisp": "langage de programmation fonctionnel"};
```

Le mécanisme de javascript pour les tableaux associatifs est celui qui sert à implémenter les objets. Nous reviendrons sur les détails quand nous aborderons le concept.

# Supprimer un élément dans un tableau

Pour supprimer un élément dans un tableau, on peut utiliser la méthode `splice`.

```
t.splice(indice,nombre);
```

Supprimer *nombre* éléments dans t à partir de l'indice *indice*.

En option :

```
t.splice(indice,nombre, e1, e2,...);
```

remplace *nombre* éléments dans t à partir de l'indice *indice* par e1, e2...

# Structures de contrôle

# if

```
if (CONDITION) {
  ACTIONS1;
} else {
  ACTIONS2;
}
```

# for

Deux syntaxes possibles

```
for (INIT; TEST; INCREMENT) {
  ACTIONS;
}
```

exemple

```
s= 0;
for (var i= 0; i < tab.length; i++) {
  s= s+ tab[i];
}
```

# for (suite)

Pour parcourir *les indices* d'un tableau, on peut utiliser `for (var ... in ...)`

```
function forTab() {
 var t= new Array("un", "deux", "trois");
 for (var x in t) {
 alert( x + " : " + t[x]);
 }
}
```

démo

affichera successivement "0 : un", "1: deux " et "2 : trois".

# for (fin)

Fonctionne aussi pour les tableaux associatifs :

```
var t= {"java": "langage de programmation orienté objet",
                "lisp": "langage de programmation fonctionnel"};

for (var nom in t) {
  alert("information sur le langage " + nom + " " + t[nom]);
}
```

# while

```
while (CONDITION) {
  ACTIONS;
}
```

### exemple

```
var a=10;
while(a>1){
  alert(a);
  a--; //decremente a de 1
       //attention si vous l'oubliez boucle infinie...
}
```

# do...while

```
do {
  ACTIONS;
} while (CONDITION);
```

### exemple

```
var a=10;

do{
  alert(a);
  a--; //decremente a de 1
       //attention si vous l'oubliez boucle infinie...
}while(a>1);
```

# switch

```
switch (VALEUR_A_TESTER) {
  case VAL1:
        ACTIONS;
  break;
  case VAL2:
        ACTIONS;
  break;
  default:
        ACTIONS;
}
```

Contrairement au C et au Java, la valeur à tester peut être une chaîne de caractères:

```
switch (reponse) {
        case "oui":
                alert("merci");
        break;
        case "non":
                alert("au revoir");
        break;
}
```

# les fonctions

- Bout de code réutilisable, avec un nom.
- Déclarées à l'aide du mot-clef `function`.
- Une fonction *peut* retourner une valeur, à l'aide du mot-clef `return`

```
<script type="text/javascript">
        function somme(a,b) {
                        return a+b;
        }
</script>
<button onclick="alert(somme(3,2))"> montrer somme (entiers) </button>
<button onclick="alert(somme('bonjour ','monde'))"> montrer somme (chaînes)</button>
```

montrer somme (entiers) montrer somme (chaînes)

à noter:

- on n'explicite pas le type des arguments
- les fonctions sont uniquement identifiées par leur nom. On ne peut pas avoir deux fonctions qui s'appellent `somme`

# Variables globale

- Une variables déclarée en dehors d'une fonction est *globale*.

montrer exempleVariable1

```
        <script type="text/javascript">
        // marche aussi sans var...
        var exempleVariable1= 0;

        function augmenter1() {
                        exempleVariable1= exempleVariable1 + 1;
        }
        </script>
        <button onclick="augmenter1(); alert(' la variable globale vaut '+ exempleVar
                montrer exempleVariable1 </button>
```

Dans cet exemple, la variable exempleVariable1 modifiée par la fonction est la même que la variable globale.

# Variables locale

- Une variable est locale si elle est déclarée dans une **fonction**.
- On utilise `var`.

montrer exempleVariable2

```
<script type="text/javascript">
var exempleVariable2= 0;

function augmenter2() {
        var exempleVariable2=55; // variable locale.
        exempleVariable2= exempleVariable2 + 1;
        alert("dans la fonction : "+exempleVariable2);
}
</script>

<button onclick="augmenter2(); alert('la var. globale vaut '+exempleVariable2);">
        montrer exempleVariable2
</button>
```

Notez que `var` est optionnelle (quoique *très fortement conseillée*) pour les déclarations de variables globales, mais obligatoire pour les variables locales. Une variable utilisée dans une fonction, mais pas déclarée, est considérée comme globale.

Les paramètres des fonctions sont automatiquement considérés comme des variables locales

Attention, la notion de localité est différente de celle de java : localité par rapport aux fonctions et non par rapport aux blocs

# Manipulation de la page Web

But: pouvoir lire et modifier des éléments de la page web par programme.

- Pour changer leur aspect
- Pour lire et écrire des éléments de formulaire
- On passe par le DOM « Document objets model ».

# Première approche du DOM

- On dispose d'un objet nommé **`document`**, qui représente la page web.
- Pour manipuler un élément, on lui donne un identifiant.
- On peut alors récupérer l'élément à l'aide de la méthode **`getElementById`**

  Un texte

  peindre

```
<p id="dom1">Un texte</p>
<button onclick="document.getElementById('dom1').style.color='red';">peindre</bu
```

# Propriétés des éléments

Un élément retourné par getElementById a plusieurs propriétés, que l'on peut lire et/ou écrire.

- style : permet l'accès à toutes les caractéristiques graphiques CSS de l'élément.

- value : valeur d'un champ de formulaire. En lecture et en écriture.
- innerHTML : propriété qui permet de modifier le *contenu* de l'élément. Ne fonctionne pas pour tous les éléments.
- id : l'id de l'élément
- className : classe de l'élément (pour les CSS)

Nous allons les détailler.

# DOM et CSS (Cascading Style Sheets)

- Les attributs de style CSS d'un élément du DOM permettent de changer son aspect.
- Ils sont accessibles par la propriété "style" des éléments.

Les caractéristiques de cet élément sont contrôlées par les CSS.

color black Changer Css

```
<div id="controleParCSS" style="border-style:solid">
        Les caractéristiques de cet élément sont contrôlées par les CSS.
</div>


<p>
        <select name="cssName" id="cssName">
                <option value="color">color</option>
                <option value="background">background</option>
                <option value="fontSize">fontSize</option>
                <option value="display">display</option>
                <option value="position">position</option>
                <option value="left">left</option>
                <option value="right">right</option>
                <option value="top">top</option>
                <option value="bottom">bottom</option>
                <option value="width">width</option>
                <option value="float">float</option>
                <option value="clear">clear</option>
        </select>
        <input id="cssValue" name="cssValue" value="black">
        <button type="button" onclick="setCSS()">Changer Css</button>
</p>


<script type="text/javascript">
function setCSS() {
        var elt= document.getElementById('controleParCSS');

        var cssValue= document.getElementById('cssValue').value;

        var cssName= document.getElementById('cssName').value;

        switch (cssName) {
                case 'background':
                        elt.style.background= cssValue;
                break;
                case 'color':
```

```
                        elt.style.color= cssValue;
                break;
                case 'fontSize':
                        elt.style.fontSize= cssValue;
                break;
                        case 'display':
                        // none : caché ; inline : dans le cours du texte;
                        // block : comme un bloc.
                        elt.style.display= cssValue;
                break;
                case 'position':
                        // static, absolute, relative ou fixed
                        // static : le défaut.
                        // ne prend pas en compte left, right, top ou bottom.
                        // absolute : position donnée par rapport au conteneur
                        // relative : relatif à l'élément précédent
                        // fixed : relatif à la fenêtre du navigateur. (pas sur IE).
                        elt.style.position= cssValue;
                break;
                case 'left':
                        elt.style.left= cssValue;
                break;
                case 'right':
                        elt.style.right= cssValue;
                break;
                case 'top':
                        elt.style.top= cssValue;
                break;
                case 'bottom':
                        elt.style.bottom= cssValue;
                break;
                case 'width':
                        elt.style.width= cssValue;
                break;
                case 'float':
                        // élément flottant : left, right, none
                        elt.style.cssFloat= cssValue;
                break;
                case 'clear':
                        // refuse des flottants sur les côtés
                        // left, right, both, none
                        elt.style.clear= cssValue;
                break;
        }
}
</script>
```

# DOM et CSS (suite)

On utilise en particulier la propriété "display", et sa valeur "none", pour faire apparaître et disparaître des éléments.

cacher

Vous le voyez ?

```
<button onclick="document.getElementById('coucou').style.display='none';">cacher</but
<div style="border: 2px solid red" id="coucou">
```

```
Vous le voyez ?
</div>
```

Exercice: écrire une fonction qui bascule l'élément entre l'état "caché" et l'état visible, et vice-versa.

# Manipulation des images

On peut changer ce qu'une image affiche en modifiant sa propriété "src":

```
  <img id="img1" src="coursJavascriptdata/ledlightblue.png"
       onmouseover="document.getElementById('img1').src='coursJavascriptdata/ledgreen
       onmouseout="document.getElementById('img1').src='coursJavascriptdata/ledlightb
       alt="une bille"
/>
```