Programmation Web avec JavaScript

Christophe Lecoutre
lecoutre@cril.fr

IUT de Lens - CRIL CNRS UMR 8188
Université d'Artois
France

Département SRC - 2010/2011

Outline

Outline

Bibliographie

Livres

- Nicholas Zakas
  Professional JavaScript for Web developpers
  2nd Edition, Wrox, 2009.

Sites

- Tout JavaScript sur `http://www.toutjavascript.com/main/index.php3`
- Javascriptfr `http://www.javascriptfr.com`
- w3school-Javascript sur `http://www.w3schools.com/js/default.asp`

Description Générale

JavaScript est un langage de programmation de scripts principalement utilisé dans les pages web interactives (comme par exemple, la validation de données d'un formulaire).
Il est formé de trois composants:

- ECMAScript, qui est défini dans l'édition ECMA-262, et qui fournit les fonctionnalités centrales
- DOM (Document Object Model) qui fournit les fonctionnalités pour interagir avec le contenu d'une page web
- BOM (Browser Object Model) qui forunit les fonctionnalités pour interagir avec le navigateur

Remarque

Il y a de grandes disparités de support des trois composants de Javascript par les navigateurs principaux (Firefox, IE, Chrome, Safari, Opera).

Javascript dans HTML

L'élément HTML `script` permet d'intégrer du code Javascript dans une page.
Les attributs de cet élément sont :

- `type` : indique le type de contenu (appelé aussi type MIME). La valeur est typiquement `"text/javascript"`.
- `charset` (optionnel) : indique le jeu de caractères utilisé.
- `defer` (optionnel) : indique si l'exécution du script doit être décalée.
- `src` (optionnel) : indique que le code se situe dans un fichier externe.

Warning

Même en utilisant `defer="defer"`, certains navigateurs ne retarderont pas l'exécution du code.

Inline Code

Il suffit d'utiliser uniquement l'attribut `type` et de placer le code au coeur de l'élément `script`.

Exemple

```
<script type="text/javascript">
  function sayHi() {
    alert("Hi!");
  }
</script>
```

Warning

Un erreur se produit si `</script>` apparaît dans le code.

```
<script type="text/javascript">
  function sayScript() {
    alert("</script>");
  }
</script>
```

External Files

Il suffit d'utiliser uniquement l'attribut `type` avec l'attribut `src`.

Exemple

```
<head>
  <title> Example </title>
  <script type="text/javascript" src="example1.js"> </script> />
  <script type="text/javascript" src="example2.js"> </script> />
  ...
```

Warning

Pour exécuter le code après le chargement de la page, on utilisera :

```
<body>
  <!-- content here -->
  <script type="text/javascript" src="example1.js"> </script> />
  <script type="text/javascript" src="example2.js"> </script> />
</body>
```

Inline code et symboles spéciaux

Le code suivant n'est pas correct en XHTML :

```
<script type="text/javascript">
  function test(a,b) {
    if (a < b) alert("premier parametre plus petit");
  }
</script>
```

Première solution: replacer < par <

Deuxième solution:

```
<script type="text/javascript">
// <![CDATA
  function test(a,b) {
    if (a < b) alert("premier parametre plus petit");
  }
// ]]>
</script>
```

Inline code ou External Files ?

Il est préférable d'utiliser des fichiers externes pour des raisons de :

- maintenabilité : le code javascript peut être rassemblé dans un unique répertoire
- caching : un fichier js partagé par deux pages ne sera téléchargé qu'une seule fois
- lisibilité : pas besoin d'astuces telles que // ... [CDATA qui polluent la page

Remarque

L'élément `noscript` permet d'afficher un contenu lorsque le navigateur ne supporte pas les scripts ou que le support a été désactivé. Par exemple :

```
<body>
  <noscript>
    <p> Cette page nécessite un support pour JavaScript </p>
  </noscript>
```

Outline

Identificateurs et commentaires

Les identificateurs

- sont sensibles à la casse: `test` n'est pas `Test`
- sont formés par convention en utilisant le style camel case comme dans `sommeNotes`

Exemple

Les commentaires

```
// single line comment

/*
 * multi-line
 * comment
 */
```

Identificateurs et commentaires

Les instructions

- se terminent par un point-virgule
- nécessitent des accolades lorsqu'elles forment un bloc

Exemple

```
var sum = a + b;
...
if (sum > 0) alert("positif");
...
if (a < b) {
  var tmp = a;
  a = b;
  b = tmp;
}
```

Variables

Déclarer une variable nécessite l'utilisation du mot-clé `var`.

Exemple

```
var sum;
...
var v = 100;
...
v = "coucou";  // autorisé mais déconseillé
...
var x = 10, y = 20, z = false;  // multiple déclaration
```

Portée locale

```
function test() {
  var message = "hi";
}
test();
alert(message);  // undefined
```

Portée globale

```
function test() {
  message = "hi";
}
test();
alert(message);  // "hi"
```

Types

Il y a principalement trois types classiques de données Boolean, Number and String et un type complexe Object (liste de couples nom-valeurs). Il est possible de déterminer le type (courant) d'une variable avec l'opérateur `typeof` qui retourne l'une des valeurs suivantes:

- undefined si la valeur est indéfinie (variable déclarée mais pas initialisée ou variable non déclarée)
- boolean
- number
- string
- object si la valeur est un objet ou null
- function si la valeur est une fonction

Remarque

Les fonctions sont considérées comme des objets qui ont des propriétés spéciales.

Valeurs undefined et null

La valeur undefined fait référence à une variable non déclarée ou déclarée mais pas initialisée, null fait référence à une variable censée référencer un objet mais non encore disponible.

Exemple

```
var message = "coucou";
alert (typeof message);  // "string"
var m1;
alert(m1);  // "undefined"
alert(m2);  // erreur
alert(typeof m1);  // "undefined"
alert(typeof m2);  // undefined
...
var car = null;
alert(typeof car);  // "object"
```

Boolean et Number

Le type Boolean a deux valeurs true et false. Le type Number sert à représenter des valeurs entières et des valeurs flottantes (réelles).

Exemple

```
var x = 55;
var y = 070;  //octal pour 56 (en base 10)
var z = 0xA,  //hexadecimal pour 10
var f1 = 10.2;
var f2 = 3.12e7;  // représente 31 200 000
var f3 = 3e-10;  // représente 0,0000000003
```

Remarque

Les valeurs particulières du type Number sont :

- Number.MIN_VALUE, Number.MAX_VALUE
- Number.NEGATIVE_INFINITY, Number.POSITIVE_INFINITY
- NaN (Not a Number)

Conversions numériques

Relatifs aux valeurs et conversions des nombres, on trouve 4 fonctions :

- `isNaN` détermine si un paramètre donné n'est pas un nombre
- `Number` effectue une conversion
- `parseInt` effectue une conversion en valeur entière
- `parseFloat` effectue une conversion en valeur réelle

Exemple

```
alert(isNaN(10));  // false
alert(isNaN("10")); // false - peut être convertie
alert(isNaN("blue")); // true - ne peut être convertie
var num1 = Number("hello world");  // NaN
var num2 = Number("00001"); // 1
var num3 = Number(true);  // 1
var num3 = parseInt("");  // NaN
var num4 = parseInt(22.5);  // 22
var num5 = parseInt("70",10);  // 70 - la base 10 est spécifiée
var num6 = parseFloat("22.5"); // 22.5
```

String

On utilise les quotes simples (apostrophes) ou doubles (guillemets) pour définir des valeurs chaînes de caractères. L'attribut `length` permet de déterminer la longueur d'une chaîne.

Exemple

```
var nom = "Wilde";
var prenom = 'Oscar';
alert(prenom.length);  // 6
var message = "toto a dit \"je suis malade\".";
```

Il est possible de transtyper une valeur en chaîne avec `String()`.

Exemple

```
var v1 = 10;    alert(String(v1));  // "10"
var v2 = true;  alert(String(v2));  // "true"
var v3 = null;  alert(String(v3));  // "null"
var v4;         alert(String(v4)); // "undefined"
```

String

Il existe de nombreuses fonctions sur les chaînes de caractères.

Exemple

```
var s = "hello world";
alert(s.length); // 11
alert(s.charAt(1));  // "e"
alert(s.charCodeAt(1));  // 101
alert(s.slice(3)); // "lo world"
alert(s.slice(-3)); // "rld"
alert(s.substring(3,7)); // "lo w"
alert(s.indexOf("o")); // 4
alert(s.lastIndexOf("o")); // 7
alert(s.toUpperCase()); // HELLO WORLD
alert(s + " !"); // hellow world !
```

Math

Il s'agit d'un objet définissant de nombreuses constantes et fonctions mathématiques.

Exemple

```
alert(Math.E); // la valeur de e
alert(Math.PI); // la valeur de pi
alert(Math.min(5,12)); // 5
alert(Math.max(23,5,7,130,12)); // 130
alert(Math.ceil(25.3)); // 26
alert(Math.floor(25.8)); //25
alert(Math.random()); // valeur aléatoire entre 0 et 1
var n = Math.floor(Math.random()*nb + min);
alert(n); // valeur aléatoire entre min et min+nb (exclus)
```

D'autres fonctions :

Math.abs(x)	Math.exp(x)	Math.log(x)
Math.pow(x,y)	Math.sqrt(x)	
Math.sin(x)	Math.cos(x)	Math.tan(x)

Opérateurs

Typiquement, ceux de C, C++ et java:

- incrémentation/décrémentation (++, −−)
- arithmétiques (+,−,∗,/,%)
- relationnels (>,<,>=,<=,==,! =) et (===,! ==)
- logique (!,&&,||)
- affectation (=,+=,−=,*=,/=,%=)
- bit à bit (&,|,<<,>>, ...)

Exemple

```
var age = 10;
age++;
alert(age); // 11
alert(age > 10 && age < 20); // true
alert(26 % 5); // 1
age*=2;
alert(age); // 22
```

Outline

Structures de contrôle

Elles sont très proches de celles de langages tels que C, C++ et Java. Pour rappel, les structures de contrôles sont de trois types :

- **Séquence** : exécution séquentielle d'une suite d'instructions séparées par un point-virgule
- **Alternative** : structure permettant un choix entre divers blocs d'instructions suivant le résultat d'un test logique
- **Boucle** : structure itérative permettant de répéter plusieurs fois la même bloc d'instructions tant qu'une condition de sortie n'est pas avérée

Remarque

Toute condition utilisée pour une alternative ou boucle sera toujours placée entre parenthèses.

Warning

Il ne faut pas confondre $x == y$ (test d'égalité) avec $x = y$ (affectation).

Alternative

L'instruction **if** sans partie else :

```
if (condition) instruction;
```

```
if (condition) {
  instruction;
}
```

```
if (condition) {
  instruction1;
  instruction2;
  ...
}
```

Exemple

```
if (x >= 0) alert("valeur positive ou nulle");
...
if (note > 12 && note <= 14) {
  alert("bravo");
  mention="bien";
}
```

Alternative

L'instruction **if...else** :

```
if (condition) instruction1;
else instruction2;

if (condition) {
  instructions1;
} else {
 instructions2;
}
```

```
if (condition1) {
  instructions1;
} else if (condition2) {
  instructions2;
} else {
  instructions3;
}
```

Exemple

```
if (rank == 1)
  medaille="or";
else if (rank == 2)
  medaille="argent";
else if (rank == 3)
  medaille="bronze";
```

Alternative

L'instruction **switch** :

```
switch (expression) {
  case valeur1 :
    instructions1;
    break;
  case valeur2 :
    instructions2;
    break;
  ...
  case valeurN :
    instructionsN;
    break;
  default:
    instructionsDefault;
}
```

Remarque

Le branchement par défaut n'est pas obligatoire.

Alternative

L'opérateur ternaire ?: permet de remplacer une instruction if...else simple. Sa syntaxe (lorsqu'utilisée pour donner une valeur à une variable) est :

```
variable = condition ? expressionIf : expressionElse;
```

Elle est équivalente à :

```
if (condition) variable=expressionIf;
else variable=expressionElse;
```

Exemple

```
var civilite = (sexe == "F") ? "Madame" : "Monsieur";
```

Remarque

Cet opérateur est utile pour les expressions courtes.

Boucle

L'instruction **while** :

```
while (condition) instruction;
```

```
while (condition) {
  instruction1;
  instruction2;
  ...
}
```

Exemple

```
var num = 1;
while (num <= 5) {
  alert(num);
  num++;
}
```

Boucle

L'instruction **for** :

```
for (instructionInit; condition; instructionIter)  instruction;

for (instructionInit; condition; instructionIter) {
  instruction1;
  instruction2;
  ...
}
```

Exemple

```
for (var num = 1; num <= 5; num++)
  alert(num);
```

Remarque

Il n'est pas nécessaire de placer des parenthèses autour de la condition pour le for.

Boucle

L'instruction **do...while** :

```
do  {
  instruction1;
  instruction2;
  ...
}
while (condition);
```

L'instruction **for-in** pour les objets :

```
for (var prop in window)
  document.writeln(prop);
```

Boucle

Certaines instructions permettent un contrôle supplémentaire sur les boucles :

- **break** permet de quitter la boucle courante
- **continue** permet de terminer l'itération en cours de la boucle courante

Exemple

```
for (var i=0; i<5; i++) {
   for (var j=0; j<5; j++) {
    if ((i+j)%3 == 0)
      continue;
    for (var k=0; k<5; k++) {
      if ((i+j+k)%3 == 0)
        break;
      alert(i + " " + j + " " + k);
    }
  }
}
```

Exercice

Écrire le code Javascript qui détermine si un nombre entier x est parfait.

Definition

Un nombre est parfait ssi il est égal à la somme de ses diviseurs stricts.

6 est parfait car $6 = 1 + 2 + 3$.

Outline

Object

Utilisé pour stocker des données. Création d'une variable (de type Object) avec `new Object()` ou de manière littérale (énumération entre accolades).

Exemple

```
var person = new Object();
person.name = "Julien";
person.age = 23;

var person = {
   name : "Julien",
   age : 23
}
```

Remarque

Il est possible d'utiliser les crochets pour accéder à un champ.

```
alert(person.name);
alert(person["name"]);
var field="name"; alert(person[field]);
```

Array

Les tableaux peuvent contenir des données de nature différente.

Exemple

```
var colors = new Array(); // tableau vide
var colors2 = new Array(20); // tableau avec 20 cases
var colors3 = new Array("red","blue","green"); // 3 cases
var colors4 = ["red","blue","green"]; // notation littérale
```

Remarque

Le champ `length` indique la taille d'un tableau.

```
var colors = ["red","blue","green"];
alert(colors.length); // 3
colors[colors.length]="black"; // nouvelle couleur
colors[99]="pink";
alert(colors.length);  // 100
alert(colors[50]): // undefined
colors.length=10; // plus que 10 cases
```

Array

De nombreuses méthodes existent sur les tableaux.

Exemple

```
var colors = ["red","blue","green"];
alert(colors); // red,blue,green
alert(colors.join(";")); // red;blue;green
colors.push("black");
alert(colors); // red,blue,green,black
var item = colors.pop();
alert(item + " " + colors); // black red,blue,green
var item2= colors.shift();
alert(item2 + " " + colors); // red blue,green
```

Remarque

Il existe d'autres méthodes telles que `concat`, `slice` et `splice`.

Réordonner les tableaux

On peut utiliser `reverse` et `sort`.

Exemple

```
var values = [0, 1, 2, 3, 4];
alert(values.reverse()); // 4,3,2,1,0
values = [0, 1, 5, 10, 15];
alert(values.sort()); // 0,1,10,15,5



function compare(value1, value2) {
  ...
}
values = [0, 1, 5, 10, 15];
alert(values.sort(compare)); // 0,1,5,10,15
```

Exercice. Écrire le corps de la fonction `compare` ci-dessus qui retourne -1 lorsque le premier argument est inférieur au second, $+1$ lorsque le premier argument est supérieur au second, et 0 sinon.

Fonctions

La syntaxe pour définir une fonction est :

```
function name(arg0, arg1, ..., argN) {
  statements
}
```

Exemple

```
function sayHi(name) {
  alert("Hello " + name);
}


sayHi("Nicolas");
```

Exemple

```
function sum(num1, num2) {
  return num1+num2;
}


alert(sum(5,10)); //15
```

Remarque

Une fonction peut retourner un résultat avec l'instruction `return` même si rien ne l'indique au niveau de la signature (en-tête) de la fonction.

Arguments

Il existe toujours un tableau `arguments` implicite lors de l'appel à une fonction.

Exemple

```
function f() {
  alert(arguments.length);
  for (var i=0; i<arguments.length; i++)
    alert(arguments[i]);
}


f("Nicolas"); // affiche 1 Nicolas
f("Nicolas",25);  // affiche 2 Nicolas 25
```

Remarque

La possibilité de faire appel à une fonction avec un nombre variable de paramètres pallie l'impossibilité de surcharge (overloading).

Fonctions comme objets

Il y a deux manières (sensiblement équivalentes) de définir une fonction.

Exemple

```
function sum(n1, n2) {
  return n1+n2;
}

alert(sum(5,10)); //15
```

Exemple

```
var sum = function (n1, n2) {
  return n1+n2;
};

alert(sum(5,10)); //15
```

Remarque

Il est possible de référencer une fonction par deux variables différentes ou de passer une fonction comme paramètre à une autre fonction.

Outline

Window

L'objet `window` représente la fenêtre du navigateur.

Exemple

```
window.moveTo(0,0);
window.resizeTo(800,800);
var wroxWindow = window.open("http://www.wrox.com","wrox");
if (wroxWindow == null)
  alert("fenetre bloquee");
else {
  ...
  wroxWindow.close();
 }
```

Remarque

Le navigateur peut être configuré de manière à empêcher de modifier son emplacement, de modifier sa taille ou encore d'afficher une fenêtre pop-up.

Timer

timer.html

Il est possible de programmer l'exécution d'une méthode à un instant donné ou à des temps réguliers grâce à `setTimeout` et `setInterval`.

Exemple

```
function helloWorld() {
  alert("hello world");
}
var id = setTimeout(helloWorld, 1000);

var num=0, max=4, intervalId = setInterval(incrementNumber,500);

function incrementNumber() {
  if (++num == max) clearInterval(intervalId);
  else alert(num);
}
```

Remarque

Il est possible d'annuler avec `clearTimeout` et `clearInterval`.

Boites de dialogues

Des boites de dialogues peuvent être ouvertes en utilisant les méthodes `alert`, `confirm` et `prompt`.

Exemple

```
if (confirm("Are you sure?"))
  alert("I am glad");
else
  alert("I am sorry");

var name = prompt("What is your name?","Toto");
if (name != null)
  alert("Welcome " + name);
```

Remarque

Il existe deux autres méthodes asynchrones `print` et `find`.

En plus de `window`, il est possible d'utiliser les objets `location`, `navigator`, `screen` et `history`.

Exemple

```
location.href="http://www.wrox.com";
location.reload();
location.port=8080; // change le port
alert(navigator.appName); // nom du navigateur
alert(navigator.javaEnabled);
alert(screen.colorDepth);
alert(screen.width);
window.resizeTo(screen.availWidth,screen.availHeight);
history.go(2); // go forward 2 pages
history.back(); // go back one page
if (history.length == 0) alert("this is the first page");
```

Outline

DOM

DOM (Document Object Model) est une API pour manipuler les documents HTML et XML. L'objet `document` représente la page HTML chargée par le navigateur.

Exemple

```
var html = document.documentElement;
alert (html == document.firstChild); // true
var body = document.body;

document.title="nouveau titre";
var url = document.URL;
var domain = document.domain;
alert(url + " " + domain);
```

Méthode getElementById()

Cette méthode prend comme argument l'id d'un élement.

Exemple

```
// dom.html
  <head>
    ...
  </head>

  <body>
    <p id="p1"> An ordered list: </p>
    ...
    <script type="text/javascript" src="dom.js"> </script>
  </body>

// dom.js
var p = document.getElementById("p1");
p.style.color="blue";
```

Méthode getElementsByTagName()

dom.html

Cette méthode prend comme argument le nom de balise des éléments à récupérer.

Exemple

```
// dom.html
  <body>
    <ol>
      <li>XHTML</li>
      <li>CSS</li>
      ...
    </ol>
    <script type="text/javascript" src="dom.js"> </script>
  </body>

// dom.js
var list = document.getElementsByTagName("li");
for (var i=0; i< list.length; i++)
  list[i].style.color="green";
```

Propriété innerHTML

En utilisant cette propriété, il est possible de modifier le contenu d'un élément.

Exemple

```
var div1 = document.getElementById("d1");
var div2 = document.getElementById("d2");
var div3 = document.getElementById("d3");

div2.innerHTML=div1.innerHTML;
div3.innerHTML="je suis <strong> content </strong>";
```

Remarque

La propriété `innerText` (`textContent` pour Firefox) est similaire à `innerHTML`, mais ne traite que du texte simple.

Warning

Éviter d'utiliser `outerText` et `outerHTML`.

Il existe de nombreuses méthodes DOM pour créer dynamiquement des éléments.

Exemple

```
var table = document.createElement("table");
for (var i=1; i<=10; i++) {
  var row = document.createElement("tr");
  for (var j=1; j<=10; j++) {
    var cell = document.createElement("td");
    cell.appendChild(document.createTextNode(i*j));
    row.appendChild(cell);
  }
  table.appendChild(row);
}
document.getElementById("d1").appendChild(table);
```

Modifier le style dynamiquement

Tout élément HTML dispose d'un attribut `style` en JavaScript. Les noms des propriétés CSS doivent être convertis en camel case. Par exemple :

Propriété CSS	Propriété JavaScript
background-image	style.backgroundImage
color	style.color
font-family	style.fontFamily

Exemple

```
var div = document.createElement("myDiv");
myDiv.style.backgroundColor="red";
myDiv.style.width="100px";
myDiv.style.border="1px solid border";
```

Warning

La propriété `float` correspond à un mot réservé de Javascript. Il faut alors utiliser `cssFloat` (ou `styleFloat` pour IE).

Outline

Événements

Un événement est provoqué par une action de l'utilisateur ou du navigateur lui-même. Les événements ont des noms tels que `click`, `load` et `mouseover`. Une fonction appelée en réponse à un événement se nomme un écouteur (event handler ou event listener). Souvent, leur nom commence par `on` comme par exemple `onclik` ou `onload`.

Associer des écouteurs aux évènements possibles peut se faire de trois manières différentes:

- HTML
- DOM Level 0
- DOM Level 2 (pas présenté ici)

HTML Event Handlers

On utilise des attributs HTML pour déclarer les écouteurs. La valeur de ces attributs est le code JavaScript à exécuter lorsque l'événement est produit.

Exemple

```
<input type="button" value="but1" onclick="alert('clicked')" />

<script type="text/javascript">
  function showMessage() {
    alert("hello world");
  }
</script>

<input type="button" value="but2" onclick="showMessage()" />
```

Remarque

Il faut parfois échapper des caractères. Par exemple :

```
  onclick="alert(&quot;clicked&quot;)"
```

Dom Level 0 Event Handlers

dom0.html

On utilise les propriétés des éléments pour leur associer des écouteurs.

Exemple

```
// dom0.html
  <input type="button" id="but1" value="bouton 1" />


// dom0.js
  function but1Listener() {
    alert("but1 cliqued");
  }

  var but1 = document.getElementById("but1");
  but1.onclick=but1Listener;
```

Remarque

Pour éliminer l'écouteur, on place la valeur `null`. Par exemple :
`but1.onclick=null;`

Quand un événement se produit, toutes les informations le concernant sont enregistrées dans un objet appelé `event`. Il est possible de récupérer cet objet sous forme de paramètre d'une fonction écouteur.

Exemple

```
function but3Listener(event) {
  switch(event.type) {
    case "click" :
      alert("clicked at " + event.clientX + " " + event.clientY);
      break;
    case "mouseover" :
      this.style.backgroundColor="red"; break;
    case "mouseout" :
      this.style.backgroundColor=""; break;
  }
}
but3.onclick=but3Listener;
but3.onmouseover=but3Listener;
but3.onmouseout=but3Listener;
```

Événements souris

Ce sont :

click	dblclick
mousedown	mouseup
mouseover	mouseout
mousemove	

Les propriétés utiles et accessibles à partir de l'objet `event` sont :

clientX	clientY
screenX	screenY
shiftkey	ctrlKey
altKey	metaKey

Événements clavier

Ce sont :

keydown	keyup
keypress	

Les propriétés utiles et accessibles à partir de l'objet `event` sont :

shiftkey	ctrlKey
altKey	metaKey
keyCode	

Exemples de valeurs pour `keyCode`:

- 40 pour Down Arrow
- 65 pour A
- 112 pour F1

Événements HTML

Ce sont :

load	unload
abort	error
select	change
submit	reset
resize	scroll
focus	blur

Remarque

La plupart de ces évènements sont liés aux formulaires ou l'objet `window`.

Événements `load` et `unload`

Pour l'objet `window`, l'événement `load` se produit lorsque la page complète est chargée, incluant les ressources externes telles que les images, les fichiers JavaScript et CSS.

Exemple

```
function loadListener() {
    alert("loaded");
}

window.onload=loadListener;
```

Remarque

Il est possible d'associer cet événement à des éléments `img`.

Remarque

L'évènement `unload` se produit typiquement lorsqu'on change de page. Cela permet par exemple de libérer proprement certaines ressources.

Événements submit et reset

form1.html

Ils sont associés à un formulaire. Pour valider la soumission ou la ré-initialisation, la fonction écouteur doit retourner `true`.

Exemple

```
function submitListener() {
    if (this.name1.value.length == 0) return false;
    // autres controles à effectuer
    return true;
}

function resetListener() { return confirm("Are you sure?"); }

var form = document.getElementById("form1");
form.onsubmit=submitListener; form.onreset=resetListener;
```

Remarque

`this.name1.value` représente la valeur du champ `name1` de l'objet `this` qui est le formulaire auquel est associé l'écouteur.

Événements `focus` et `blur`

form2.html

Utiles par exemple pour contrôler les champs de saisie.

Exemple

```
function focusListener() {
  if (this.style.backgroundColor != "red")
    this.style.backgroundColor="yellow";
}

function blurListener() {
  if (this.value.match(/\d/g))
    this.style.backgroundColor = "red";
  else
    this.style.backgroundColor = "";
}

var field = document.getElementById("name1");
field.onfocus=focusListener;
field.onblur=blurListener;
```

Expressions régulières

La syntaxe est `/pattern/modifiers`
Modifiers are used to perform case-insensitive and global searches:

- i : perform case-insensitive matching
- g : perform a global match (find all matches rather than stopping after the first match)
- m : perform multiline matching

Le motif (pattern) est la séquence de caractères à rechercher. Il est possible d'utiliser des méta-caractères, crochets, etc.

Voir `http://www.w3schools.com/jsref/jsref_obj_regexp.asp`