

# Programmation JavaScript

Une version à jour et éditable de ce livre est disponible sur Wikilivres,  
une bibliothèque de livres pédagogiques, à l'URL :  
[https://fr.wikibooks.org/wiki/Programmation\\_JavaScript](https://fr.wikibooks.org/wiki/Programmation_JavaScript)

Vous avez la permission de copier, distribuer et/ou modifier ce document selon les termes de la Licence de documentation libre GNU, version 1.2 ou plus récente publiée par la Free Software Foundation ; sans sections inaltérables, sans texte de première page de couverture et sans Texte de dernière page de couverture.

Une copie de cette licence est incluse dans l'annexe nommée  
« Licence de documentation libre GNU ».

---

## Sections

---

- [1 Introduction](#)
  - [1.1 Références](#)
- [2 Programmer en deux minutes](#)
  - [2.1 Une minute](#)
  - [2.2 une minute trente : une fonction JavaScript](#)
  - [2.3 deux minutes](#)
  - [2.4 Voir aussi](#)
- [3 Programmer en deux minutes/Une page wiki](#)
- [4 Programmer en deux minutes/Compteur en temps réel](#)
- [5 Programmer en deux minutes/Tout un site web dans un seul document](#)
  - [5.1 Sans utiliser de JavaScript !](#)
  - [5.2 Avec jQuery, une librairie d'instructions javascript](#)
  - [5.3 Un menu déroulant en CSS](#)
- [6 Présentation](#)
  - [6.1 IDE](#)
    - [6.1.1 Libre](#)
    - [6.1.2 Propriétaire](#)
- [7 Syntaxe](#)
- [8 Ligne d'instruction](#)
  - [8.1 Composition d'une instruction](#)
  - [8.2 Délimitation des instructions](#)
  - [8.3 Regroupement des lignes d'instructions en blocs](#)
- [9 Ordre d'évaluation](#)
  - [9.1 Scripts](#)
  - [9.2 Instructions](#)
  - [9.3 Expressions](#)
- [10 Bloc d'instructions](#)
  - [10.1 Exécution séquentielle](#)
  - [10.2 Inclusion de blocs](#)
- [11 Identificateurs](#)
  - [11.1 Principales règles](#)
  - [11.2 Caractères autorisés](#)
  - [11.3 Caractères interdits](#)
  - [11.4 Standardisation](#)
  - [11.5 Notes de bas de page](#)
- [12 Expression littérale](#)
  - [12.1 Les chaînes sous forme littérale](#)
    - [12.1.1 Le mécanisme d'échappement](#)

- 12.1.1.1 L'échappement des guillemets simples (\') ou double (\")
- 12.1.1.2 Le retour à la ligne avec \n
- 12.1.1.3 La tabulation avec \t
- 12.1.1.4 L'inclusion de caractères unicode avec \u
- 12.2 Les nombres sous forme littérale
  - 12.2.1 Les nombres négatifs
  - 12.2.2 Les entiers
  - 12.2.3 Les réels
  - 12.2.4 L'octal
  - 12.2.5 L'hexadécimal
  - 12.2.6 La notation scientifique
  - 12.2.7 Taille des nombres
- 12.3 Les chaînes littérales dans les expressions associées aux évènements
- 13 Commentaire
  - 13.1 La syntaxe à double-slash.
  - 13.2 La syntaxe slash-étoile.
  - 13.3 Intégration dans les lignes d'instruction
- 14 Mots réservés
  - 14.1 Déclarations
    - 14.1.1 var
    - 14.1.2 let
    - 14.1.3 function
    - 14.1.4 void
    - 14.1.5 with
  - 14.2 Structures de contrôle
    - 14.2.1 return
    - 14.2.2 if
    - 14.2.3 else
    - 14.2.4 switch
    - 14.2.5 case
    - 14.2.6 break
    - 14.2.7 default
    - 14.2.8 for
    - 14.2.9 do
    - 14.2.10 while
    - 14.2.11 continue
  - 14.3 Gestions des erreurs
    - 14.3.1 throw
    - 14.3.2 try ... catch ... finally
  - 14.4 Opérateurs
    - 14.4.1 in
    - 14.4.2 new

- [14.4.3 instanceof](#)
- [14.4.4 typeof](#)
- [14.4.5 delete](#)
- [14.5 Valeurs spéciales](#)
  - [14.5.1 this](#)
  - [14.5.2 true](#)
  - [14.5.3 false](#)
  - [14.5.4 null](#)
  - [14.5.5 undefined](#)
- [14.6 Futurs mots réservés](#)
- [14.7 Mots réservés en jQuery](#)
- [14.8 Références](#)
- [15 Variables](#)
  - [15.1 Typage dynamique](#)
  - [15.2 Identificateur](#)
  - [15.3 Portée](#)
    - [15.3.1 Propriétés](#)
    - [15.3.2 Sous-entendre un objet](#)
    - [15.3.3 Variables locales](#)
  - [15.4 Types](#)
    - [15.4.1 undefined](#)
    - [15.4.2 number](#)
    - [15.4.3 string](#)
    - [15.4.4 boolean](#)
    - [15.4.5 object](#)
    - [15.4.6 function](#)
- [16 Opérateurs](#)
  - [16.1 Affectation \(= et ses dérivés\)](#)
  - [16.2 Concaténation \(+\)](#)
  - [16.3 Opérateurs arithmétiques](#)
    - [16.3.1 Addition \(+\)](#)
    - [16.3.2 Soustraction \(-\)](#)
    - [16.3.3 Multiplication \(\\*\)](#)
    - [16.3.4 Division \(/\)](#)
    - [16.3.5 Modulo \(%\)](#)
  - [16.4 Opérateurs logiques](#)
    - [16.4.1 ET \(&&\)](#)
    - [16.4.2 OU \(||\)](#)
    - [16.4.3 NON \(!\)](#)
  - [16.5 Opérateurs de bits](#)
    - [16.5.1 Et binaire \(&\)](#)

- 16.5.2 Ou binaire (|)
- 16.5.3 Ou exclusif binaire (^)
- 16.5.4 Non binaire (~)
- 16.5.5 Décalage de bits vers la gauche (<<)
- 16.5.6 Décalage de bits vers la droite (>>)
- 16.5.7 Décalage de bits vers la droite y compris le bit de signe (>>>)
- 16.6 Opérateurs de comparaison
  - 16.6.1 Égalité (==, ===)
  - 16.6.2 Inégalité (!=, !==)
  - 16.6.3 Ordre (<, <=, >=, >)
- 16.7 Opérateurs d'incrémentatation et décrémentation (++ --)
- 16.8 Opérateurs spéciaux
  - 16.8.1 Conditionnel ( ? : )
  - 16.8.2 Instanciation (new)
  - 16.8.3 Dé-référencement (delete)
  - 16.8.4 Propriété d'un objet (.)
  - 16.8.5 Type de variable (typeof)
  - 16.8.6 Type d'objet (instanceof)
  - 16.8.7 Appartenance à une liste (in)
  - 16.8.8 Juxtaposition (,)
- 17 Structures de contrôle
  - 17.1 Établissement d'une expression logique
  - 17.2 Le piège
  - 17.3 Branchement conditionnel
    - 17.3.1 if else
    - 17.3.2 ? :
    - 17.3.3 switch
  - 17.4 Contrôle d'itération (boucles)
    - 17.4.1 Utilisation de continue
    - 17.4.2 Utilisation de break
    - 17.4.3 Étiquettes
    - 17.4.4 for
      - 17.4.4.1 Description
      - 17.4.4.2 Boucle croissante
      - 17.4.4.3 Boucle à paliers
      - 17.4.4.4 Boucle décroissante
      - 17.4.4.5 État en sortie de boucle
      - 17.4.4.6 Boucle de parcours
      - 17.4.4.7 Éviter les pièges :
        - 17.4.4.7.1 Boucle infinie
        - 17.4.4.7.2 Variable modifiée
    - 17.4.5 while

- 17.4.6 do
- 18 Fonctions utilisateur
  - 18.1 Déclaration et identification
  - 18.2 Fonction sans paramètres
  - 18.3 Transmission de paramètres
    - 18.3.1 Transmission classique
    - 18.3.2 Autre mode de paramétrage
  - 18.4 Valeur renvoyée
    - 18.4.1 return vide
  - 18.5 Corps de la fonction
    - 18.5.1 Variables
    - 18.5.2 Sous-fonction
  - 18.6 Appel de la fonction
    - 18.6.1 Sans valeur de retour
    - 18.6.2 Avec valeur de retour
    - 18.6.3 Sans paramètre
    - 18.6.4 Avec paramètre
    - 18.6.5 Que deviennent les variables envoyées
      - 18.6.5.1 Primitives
      - 18.6.5.2 Instance d'un objet
  - 18.7 Fonction sans identificateur
- 19 Évènement
  - 19.1 L'objet Event
    - 19.1.1 type
    - 19.1.2 target
    - 19.1.3 currentTarget
    - 19.1.4 stopPropagation
    - 19.1.5 preventDefault
  - 19.2 Gestionnaires d'événements DOM-0
  - 19.3 Écouteurs d'événements
    - 19.3.1 addEventListener
    - 19.3.2 removeEventListener
  - 19.4 Événements
    - 19.4.1 onabort
    - 19.4.2 onblur
    - 19.4.3 onchange
    - 19.4.4 onclick
    - 19.4.5 ondblclick
    - 19.4.6 onerror
    - 19.4.7 onfocus

- [19.4.8 onkeydown](#)
- [19.4.9 onkeypress](#)
- [19.4.10 onkeyup](#)
- [19.4.11 onload](#)
- [19.4.12 onunload](#)
- [19.4.13 onmousedown](#)
- [19.4.14 onmouseup](#)
- [19.4.15 onmousemove](#)
- [19.4.16 onmouseout](#)
- [19.4.17 onmouseover](#)
- [19.4.18 onselect](#)
- [19.4.19 onreset](#)
- [19.4.20 onsubmit](#)
- [19.4.21 onresize](#)
- [19.4.22 onmove](#)
- [19.4.23 dragdrop](#)
- [19.4.24 javascript](#)
- [20 Objets prédéfinis](#)
- [21 Références/Objets/window](#)
  - [21.1 Propriétés](#)
  - [21.2 Méthodes](#)
- [22 Références/Objets/document](#)
  - [22.1 Propriétés](#)
  - [22.2 Méthodes](#)
- [23 Références/Objets/navigator](#)
  - [23.1 Propriétés](#)
  - [23.2 Méthodes](#)
- [24 Références/Objets/Array](#)
  - [24.1 Caractéristiques](#)
  - [24.2 Propriétés](#)
  - [24.3 Méthodes](#)
    - [24.3.1 Modification du contenu](#)
    - [24.3.2 Obtenir des données](#)
    - [24.3.3 Trier](#)
    - [24.3.4 Méthodes héritées](#)
  - [24.4 Manipulation](#)
    - [24.4.1 Instanciation](#)
    - [24.4.2 Adressage d'un élément](#)
    - [24.4.3 Utilisation d'une méthode](#)
  - [24.5 Références](#)
    - [24.5.1 Modifier le contenu du tableau](#)

- [24.5.1.1 concat\(\)](#)
  - [24.5.1.2 pop\(\)](#)
  - [24.5.1.3 push\(\)](#)
  - [24.5.1.4 shift\(\)](#)
  - [24.5.1.5 unshift\(\)](#)
  - [24.5.1.6 splice\(\)](#)
  - [24.5.1.7 reverse\(\)](#)
  - [24.5.2 Obtenir des données](#)
    - [24.5.2.1 join\(\)](#)
    - [24.5.2.2 slice\(\)](#)
  - [24.5.3 Trier le tableau](#)
    - [24.5.3.1 sort\(\)](#)
- [25 Références/Objets/Date](#)
  - [25.1 Méthodes](#)
  - [25.2 Exemples](#)
- [26 Références/Objets/Function](#)
  - [26.1 Propriétés](#)
  - [26.2 Exemples](#)
- [27 Références/Objets/Math](#)
  - [27.1 Propriétés](#)
  - [27.2 Méthodes](#)
- [28 Références/Objets/Number](#)
  - [28.1 Propriétés](#)
  - [28.2 Méthodes](#)
- [29 Références/Objets/String](#)
  - [29.1 Encodage](#)
  - [29.2 Propriétés](#)
  - [29.3 Méthodes](#)
  - [29.4 Références](#)
- [30 Références/Objets/Element](#)
  - [30.1 Propriétés](#)
  - [30.2 Méthodes](#)
  - [30.3 Références](#)
- [31 Références/Objets/RegExp](#)
  - [31.1 Syntaxe](#)
    - [31.1.1 Méthodes](#)
  - [31.2 Recherche](#)
  - [31.3 Remplacement](#)
  - [31.4 Références](#)
- [32 Fonctions prédéfinies](#)



- 33 Opérateurs de bits
  - 33.1 Opérations binaires
    - 33.1.1 Représentation binaire
    - 33.1.2 Opérations
    - 33.1.3 Utilité
  - 33.2 Les opérateurs
  - 33.3 Manipulation sur chaîne de caractères
  - 33.4 Décalage de bits
- 34 Programmation objet
  - 34.1 Programmation objet
  - 34.2 Mot clé new
  - 34.3 Object methods and fields
  - 34.4 Function et prototype
  - 34.5 mot clé this
  - 34.6 paradigme de programmation classe/objet
- 35 Notation JSON
  - 35.1 Tableau
  - 35.2 Objet
  - 35.3 Imbrications
  - 35.4 Voir aussi
- 36 Ajax
  - 36.1 Ajax : comment créer un sommaire
    - 36.1.1 Intérêt de l'utilisation d'Ajax
    - 36.1.2 Les fichiers
      - 36.1.2.1 Le fichier index.html
      - 36.1.2.2 Le fichier ajax.php
      - 36.1.2.3 Le fichier page1.html
      - 36.1.2.4 Le fichier page2.html
      - 36.1.2.5 Le fichier page3.html
      - 36.1.2.6 Le fichier page4.html
  - 36.2 Références
- 37 Dojo
- 38 Dojo/Hello World
  - 38.1 Notes
  - 38.2 Initialiser Dojo
  - 38.3 Pour commencer
  - 38.4 Créer un bouton gadget
- 39 Dojo/Widgets
  - 39.1 HTML valide W3C
- 40 Dojo/Ressources

- [40.1 Liens externes](#)
- [41 React](#)
  - [41.1 Syntaxe de base](#)
  - [41.2 Routes](#)
  - [41.3 Exemple](#)
  - [41.4 Références](#)
- [42 Débogage](#)
  - [42.1 Afficher des objets](#)
  - [42.2 Débogage au sein d'un navigateur](#)
    - [42.2.1 Firefox](#)
    - [42.2.2 Chrome](#)
    - [42.2.3 Internet Explorer](#)
  - [42.3 Exemples d'erreur](#)
    - [42.3.1 \\$ is not defined](#)
    - [42.3.2 addOnloadHook is not defined](#)
    - [42.3.3 missing \) after argument list](#)
    - [42.3.4 ReferenceError: invalid assignment left-hand side](#)
    - [42.3.5 uncaught exception: out of memory <inconnu>](#)
  - [42.4 Références](#)
  - [42.5 Voir aussi](#)
- [43 Références](#)
  - [43.1 Liens externes](#)
    - [43.1.1 Spécifications](#)
    - [43.1.2 Documentation](#)
    - [43.1.3 Outils](#)

## Introduction

**JavaScript** est un langage de [programmation](#) utilisé principalement par les navigateurs web. Il partage avec [Java](#) une syntaxe inspirée du langage C, mais leurs similitudes s'arrêtent là. Il fut créé par [Netscape](#) sous le nom LiveScript.

Le noyau du JavaScript est aujourd'hui défini par le standard [ECMA 262<sup>\[1\]</sup>](#), connu aussi sous l'appellation [ECMAScript](#) (ES).

Ce langage est intégré directement au sein des pages Web et s'exécute sur le client Web (par opposition au serveur Web) : c'est le navigateur Web qui prend en charge l'exécution de ces bouts de programme, manipulant leur environnement, appelés scripts.

Généralement, JavaScript sert à interagir avec le document [HTML](#) et ses feuilles de style CSS via l'interface DOM (Document Object Model ; on parle de HTML dynamique ou DHTML pour désigner cette intégration des trois langages). JavaScript est ainsi utilisé pour réaliser des services dynamiques en communication avec le serveur Web, la plupart du temps grâce à une technique appelée (abusivement) AJAX.

Il est souvent source de difficultés dues aux nombreuses versions différentes de l'interpréteur et du DOM, dépendantes des éditeurs de navigateurs. Chacun ayant développé sa propre variante supportant (presque) le standard ECMAScript, un ou deux des deux standards DOM, et possédant de surcroît des fonctionnalités supplémentaires et incompatibles.

## Références

---

1. <http://www.ecma-international.org/publications/standards/Ecma-262.htm>

# Programmer en deux minutes

JavaScript est un langage exécuté par le navigateur web, à l'affichage d'une page web. (il est surtout utilisé pour ça)

Créer ce document HTML (le nom du fichier se terminera généralement par **.html**) puis l'ouvrir avec un navigateur.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <script type="text/javascript">

      alert(prompt("bonjour le monde !"));

    </script>
  </head>

  <body>
    Corps de la page.
  </body>
</html>
```

Le navigateur prompt et alerte puis affiche le corps du document. **Félicitations**, vous avez écrit vos premières instructions JavaScript dans un document HTML !

Note : La plupart des navigateurs exécuteront sans problème ce code simplifié :

```
<body>
  Corps de la page.
  <script type="text/javascript">
    alert(prompt("bonjour le monde !"));
  </script>
</body>
```

## Une minute

---

Une page web est un document composé d'une tête et d'un corps (<html> <head></head> <body></body> </html>).

Le corps du document est affiché à l'écran tandis que la tête contient le titre de la page, son favicon, ses mots-clés, etc. (pour un exemple, afficher le code source de cette page (on peut souvent utiliser le raccourci clavier *Ctrl + U*))

La tête et le corps d'un document html peuvent contenir des instructions JavaScript :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
```

```
</head>

<body>
  Corps.

  <script type="text/javascript">
    alert(prompt("bonjour le monde !"));
  </script>
</body>
</html>
```

A l'ouverture de ce document, le navigateur affiche le corps puis prompt et alert.

## une minute trente : une fonction JavaScript

---

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html>
  <head>
    <script type="text/javascript">
      fonction Prompt_et_alert() {
        alert(prompt("bonjour le monde !"));
      }
    </script>
  </head>

  <body>
    <a onmouseout="Prompt_et_alert();" <!-- une ancre balise un hyperlien -->
    Corps.
    </a>

  </body>
</html>
```

La fonction est définie en tête de document et est exécutée lorsque la souris sort de l'ancre (onmouseout).

## deux minutes

---

Afin d'aller plus loin, munissez votre navigateur d'un outil tel [Firebug](#).

Pour apprendre à manipuler cet outil, introduisez une erreur dans votre document, par exemple en oubliant un des guillemets ", et vérifiez la présence de l'erreur dans la console JavaScript.

**Félicitation**, vous êtes paré pour écrire davantage d'instructions !

En deux minutes :

1. [Une page wiki](#)
2. [Compteur en temps réel des dépenses militaires](#).
3. [Tout un site web dans un seul document](#).
4. [Neige dans la fenêtre du navigateur](#).

## Voir aussi

---

- <https://jsfiddle.net/> : un interpréteur en ligne, pratique pour tester des commandes.

# Programmer en deux minutes/Une page wiki

Munis de votre éditeur de texte avec coloration syntaxique, créez ce document \*.html et ouvrez-le avec votre navigateur outillé d'une extension Firebug-like.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd">
<html lang="fr">
  <head>
    <script type="text/javascript">
      var texte;
      function Prompt_and_run() {
        texte = prompt("Bonjour. \nEntrez votre texte :");
        return affiche_dans_div();
      }
      function affiche_dans_div() {
        var wikitext = document.getElementById("wikitext");
        return wikitext.innerHTML= "Votre texte : " + texte;
      }
    </script>
  </head>

  <body>
    <a onmouseover="Prompt_and_run();"> <!-- la fonction est exécutée lorsque la
souris est sur l'ancre -->
    Ecrire.
    </a>

    <div id="wikitext"></div>

  </body>
</html>
```

Le navigateurrompte et affiche le texte dans une division du document.

Pour obtenir une wikipage, il faut communiquer les changements à un serveur, puis que le serveur enregistre les changements.

Vous l'avez compris, ceci nécessite d'autres technologies que le JavaScript. Rendez-vous sur Programmation Web/Programmer en deux minutes à la sous-page : Un wiki en JavaScript et PHP.

# Programmer en deux minutes/Compteur en temps réel

Document HTML :

```

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN" "http://www.w3.org/TR/html4
/strict.dtd">
<html lang="fr">
  <head>
  </head>

  <body>
    <div id="compteur"></div>

    <script type="text/javascript">
      /* Copier-coller le javascript ici,
       * ou créez un fichier *.js et déclarez-le dans ce document html.
       */
    </script>

  </body>
</html>

```

Le JavaScript peut sembler long mais est très simple : il exécute la fonction RunWarCost() qui se ré-exécute toutes les 100 millisecondes (window.setTimeout()).

Au passage cette fonction calcule le prix et l'affiche dans la division "compteur" du document. (document.getElementById().innerHTML)

Si vous êtes un utilisateur enregistré de Wikipédia, vous pouvez personnaliser votre JavaScript pour ajouter à votre interface plein de choses, dont bien sûr ce compteur. (voir [Gadget-WarCost.js](#))

```

////////////////////Start CUSTOMIZATION////////////////////////////////////
////Le budget affiché:
//var Gadget_WarCost_WhichWar="US in Afghanistan since 2001";
//var Gadget_WarCost_WhichWar="US in Iraq since 2003";
//var Gadget_WarCost_WhichWar="US in Wars since 2001";
var Gadget_WarCost_WhichWar="French military budget since 2009";

////Datas:
var totalIraq = 687000000000; // total dollars allocated to Iraq War.
var startOfIraqWar = new Date("Mar 20, 2003"); // start of Iraq War.
var IraqBudgetedThrough = new Date("Sept 30, 2009"); // current budget goes to

var totalAg = 2280000000000; // total dollars allocated to Afghanistan War.
var startOfAgWar = new Date("Oct 7, 2001"); // start of Afghanistan War.
var AgBudgetedThrough = new Date("Sept 30, 2009"); // current budget goes to

```



```

var totalFM=186000000000; //budget militaire français 2009 - 2014
var startOfFM=new Date("Jan 1, 2009");
var FMBudgetedThrough=new Date("Dec 31, 2014");

var separator = "";

//////////End CUSTOMIZATION//////////

RunWarCost(); // Exécute la fonction principale. //

function RunWarCost(){
if (Gadget_WarCost_WhichWar == "US in Wars since 2001") { //regarde quelles
données afficher.
    calculateAg();
    calculateIraq();
    calculateTotal();
    var OutNumberWarCost=costOfTotal;
}
else if (Gadget_WarCost_WhichWar == "US in Iraq since 2003") {
    calculateIraq();
    var OutNumberWarCost=costOfIraq;
}
else if (Gadget_WarCost_WhichWar == "US in Afghanistan since 2001") {
    calculateAg();
    var OutNumberWarCost=costOfAg;
}
else if (Gadget_WarCost_WhichWar == "French military budget since 2009") {
    calculateFrenchMilitary();
    WarCostmoney = " euros";
    var OutNumberWarCost=costOfFrenchMilitary;
}
else {alert("The value of parameter Gadget_WarCost_WhichWar is wrong.");}

if (WarCostmoney=="$") { var
OutTextCostOfWar=WarCostmoney+number_str(OutNumberWarCost);}
else { var OutTextCostOfWar=number_str(OutNumberWarCost)+WarCostmoney; } //La
chaîne de caractere du chiffre actuel

var compteur = document.getElementById("compteur");
compteur.innerHTML= "Votre compteur : " + OutTextCostOfWar + " = " +
Gadget_WarCost_WhichWar; //Affiche la chaîne de caractere

window.setTimeout(RunWarCost, 100); // Réexécute la fonction principale dans
100 millisecondes. //
}

function calculateFrenchMilitary () {
    calculateWarCost(startOfFM,FMBudgetedThrough,totalFM);
    costOfFrenchMilitary=costOfWarAmount;
}
function calculateIraq () {
    calculateWarCost(startOfIraqWar,IraqBudgetedThrough,totalIraq);
    costOfIraq=costOfWarAmount;
}
function calculateAg () {
    calculateWarCost(startOfAgWar,AgBudgetedThrough,totalAg);
    costOfAg=costOfWarAmount;
}

```

```

function calculateTotal () {
    costOfTotal = costOfAg + costOfIraq;
}

function calculateWarCost (startOfWar,BudgetedThrough,totalMoney) {
    var totalMS = BudgetedThrough - startOfWar;           // total MS for dollars
    var ratePerMS    = totalMoney / totalMS;           // the rate per MS of the war so
    var curDate = new Date();                          // today's date
    var diff = curDate - startOfWar;                   // MS between today and start of the
    costOfWarAmount = diff * ratePerMS;               // cost of war at this time
}

function number_str(n){
    var x=n.toString();
    var dot=x.lastIndexOf('.');
    x=x.substr(0,dot);
    var l=x.length;
    var res="";
    for(l-=3;l>0;l-=3){res=separator+x.substr(l,3)+res;}
    res=x.substr(0,l+3)+res;
    return res;
}

```

# Programmer en deux minutes/Tout un site web dans un seul document

Pour créer **Tout un site web dans un seul document**, écrivez un document HTML contenant des divisions non-affichées (`style="display:none"`) et une fonction JavaScript pour les afficher lorsqu'on navigue dans le document, en cliquant sur les ancres `<a onclick="afficher()">`, ce qui exécute le JavaScript suivant :

```
function afficher(a) {
    var e=document.getElementById(a);      /* e est l'élément du document d'identité a
*/
    e.style.display="inline";              /* Applique le style display:inline à l'element d'id
a. (affiche) */
    cacher_sauf(e);                        /* Cache les autres éléments. */
}

function cacher_sauf(e) {
    var a=document.getElementsByClassName('page'); /* Obtient tous les éléments de
classe page */
    for(i in a) {
        if (a[i] != e) {a[i].style.display="none";} /* Si différent de e, applique le
style display:none. (cache) */
    }
}
```

- Créez à l'aide d'un éditeur avec coloration syntaxique le fichier \*.html suivant, et ouvrez-le dans votre navigateur :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr" dir="ltr">
<head>
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<script type="text/javascript">
    alert(" Copier/coller ici les fonctions javascript afficher() et cacher_sauf().");
</script>

<style type="text/css">
    #sommaire {font-size:18pt; margin:3em; border-bottom:1px dotted grey;}
    #contenu {margin:1em;}
</style>
</head>

<body >
    <div id="sommaire">
```

```

        <a onclick="afficher('home');">Home</a> -
        <a onclick="afficher('page1');">Page 1</a> -
    <a onclick="afficher('page2');">Page 2</a> -
    <a onclick="afficher('contact');">Contact</a>
</div>

<div id="contenu">
<div class="page" id="home">
    Welcome on my document. Its <a onclick="afficher('page1');" href="#">page
1</a> is about...
    </div>

    <div class="page" id="contact" style="display:none;">
        contact page.
    </div>

    <div class="page" id="page1" style="display:none;">
        page1 text.
    </div>

    <div class="page" id="page2" style="display:none;">
        page2 text
    </div>
</div>
</body>
</html>

```

**Félicitation**, les divisions de classe *page* sont affichées et cachées lors des clics sur les ancres.

Remarquez que ce document pose un problème d'accessibilité : les éléments cachés ne s'afficheront pas si le navigateur n'exécute pas le javascript.

## Sans utiliser de JavaScript !

L'apparence déclarée à la pseudo-classe *hover* est active lorsque la souris est sur l'élément.

### CSS

```

.page .corps {      display:none;} /* Le corps n'est pas affiché. */
.page:hover .corps { display:block;} /* Lorsque la souris est sur la page, le corps
est affiché. */

```

- Créez ce document \*.html et ouvrez-le dans un navigateur :

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr" dir="ltr">
<head>
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

```

```

<meta name="robots" content="index,follow" />
<link rel="shortcut icon" href="/favicon.ico" />
<link rel="copyright" href="http://creativecommons.org/licenses/by-sa/3.0/" />
<style type="text/css">
#site {cursor:pointer;}
#site {position:relative;}
#site .page { float:left;
                margin:0 1em; padding:3px;
                border:1px solid gray;}
#site .page:hover {border:2px dotted gray; }

#site .page:hover .corps { display:block; }
#site .page .corps {      display:none;
                position: absolute;
                top: 1.5em;
                left: 1em; padding:1em;}
</style>
</head>
<body >

<div id="site">
  <div class="page"><span>Home</span>
    <div class="corps">
      Home. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt
pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis.
Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh.
Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus
imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique
metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam. In
laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus sapien
at erat.Etiam tellus magna, dignissim vel pulvinar non, auctor et neque. Aliquam erat
volutpat. Integer iaculis lorem eget tortor varius faucibus. Mauris ac lectus non metus
ornare laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi a urna mattis
rhoncus semper est mollis. Nunc velit elit, varius id adipiscing at, gravida at lectus.
Sed id odio justo, vel pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed
metus. Maecenas commodo libero id augue congue non luctus mauris luctus. Ut neque magna,
porttitor vel elementum eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus,
tortor velit bibendum sem, nec rutrum libero orci hendrerit velit. Aenean in eros ac
orci sollicitudin suscipit. Nunc lacinia gravida imperdiet. Morbi convallis mi at magna
lacinia id feugiat magna dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque
tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed
erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus.
Fusce aliquam tempus augue.
Nam nec tortor ligula. Pellentesque habitant morbi tristique senectus et netus et
malesuada fames ac turpis egestas. Pellentesque eleifend libero in lorem fermentum
auctor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos
himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam adipiscing dictum orci,
eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras auctor laoreet odio a
iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus ac. Vivamus placerat
quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut ut magna. Nullam tellus
orci, consectetur ac consectetur et, malesuada sed dolor. Donec non libero sit amet
risus venenatis laoreet.
Donec ultrices, sapien eget placerat consectetur, magna libero gravida dolor, at
condimentum dui lectus non lorem. Duis eget risus ante, at mattis ipsum. Vivamus nulla
augue, placerat ac molestie nec, sollicitudin nec ante. Sed euismod, risus at sodales
fringilla, lacus mi accumsan nunc, eget lobortis dolor neque sed arcu. Donec scelerisque
mauris vitae lectus faucibus feugiat. Aenean purus odio, volutpat et feugiat eu, posuere
sit amet nunc. Cras lectus orci, rhoncus ac mattis ut, ultrices id mi. Cras massa lorem,
porttitor sed tempus et, congue quis dui. Fusce tristique faucibus ultricies. In metus

```

```

felis, congue quis cursus nec, congue sit amet arcu. Quisque in molestie nibh. Curabitur
at odio et tortor faucibus commodo a non augue. Nunc a augue a magna vehicula suscipit.
    </div><!-- fin du corps -->
</div><!-- fin de la page -->

<div class="page"><span>About</span>
  <div class="corps">
    About
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt
pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis.
Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh.
Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus
imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique
metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam. In
laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus sapien
at erat. Etiam tellus magna, dignissim vel pulvinar non, auctor et neque. Aliquam erat
volutpat. Integer iaculis lorem eget tortor varius faucibus. Mauris ac lectus non metus
ornare laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi a urna mattis
rhoncus semper est mollis. Nunc velit elit, varius id adipiscing at, gravida at lectus.
Sed id odio justo, vel pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed
metus. Maecenas commodo libero id augue congue non luctus mauris luctus. Ut neque magna,
porttitor vel elementum eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus,
tortor velit bibendum sem, nec rutrum libero orci hendrerit velit. Aenean in eros ac
orci sollicitudin suscipit. Nunc lacinia gravida imperdiet. Morbi convallis mi at magna
lacinia id feugiat magna dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque
tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed
erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus.
Fusce aliquam tempus augue. Nam nec tortor ligula. Pellentesque habitant morbi tristique
senectus et netus et malesuada fames ac turpis egestas. Pellentesque eleifend libero in
lorem fermentum auctor. Class aptent taciti sociosqu ad litora torquent per conubia
nostra, per inceptos himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam
adipiscing dictum orci, eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras
auctor laoreet odio a iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus
ac. Vivamus placerat quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut ut
magna. Nullam tellus orci, consectetur ac consectetur et, malesuada sed dolor. Donec non
libero sit amet risus venenatis laoreet. Donec ultrices, sapien eget placerat
consectetur, magna libero gravida dolor, at condimentum dui lectus non lorem. Duis eget
risus ante, at mattis ipsum. Vivamus nulla augue, placerat ac molestie nec, sollicitudin
nec ante. Sed euismod, risus at sodales fringilla, lacus mi accumsan nunc, eget lobortis
dolor neque sed arcu. Donec scelerisque mauris vitae lectus faucibus feugiat. Aenean
purus odio, volutpat et feugiat eu, posuere sit amet nunc. Cras lectus orci, rhoncus ac
mattis ut, ultrices id mi. Cras massa lorem, porttitor sed tempus et, congue quis dui.
Fusce tristique faucibus ultricies. In metus felis, congue quis cursus nec, congue sit
amet arcu. Quisque in molestie nibh. Curabitur at odio et tortor faucibus commodo a non
augue. Nunc a augue a magna vehicula suscipit.</p>
  </div>
</div>

<div class="page"><span>contact page</span>
  <div class="corps">
    Contact page.
    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt
pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis.
Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh.
Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus
imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique
metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam. In
laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus sapien
at erat. Etiam tellus magna, dignissim vel pulvinar non, auctor et neque. Aliquam erat

```

```
volutpat. Integer iaculis lorem eget tortor varius faucibus. Mauris ac lectus non metus  
ornare laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi a urna mattis  
rhoncus semper est mollis. Nunc velit elit, varius id adipiscing at, gravida at lectus.  
Sed id odio justo, vel pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed  
metus. Maecenas commodo libero id augue congue non luctus mauris luctus. Ut neque magna,  
porttitor vel elementum eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus,  
tortor velit bibendum sem, nec rutrum libero orci hendrerit velit. Aenean in eros ac  
orci sollicitudin suscipit. Nunc lacinia gravida imperdiet. Morbi convallis mi at magna  
lacinia id feugiat magna dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque  
tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed  
erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus.  
Fusce aliquam tempus augue. Nam nec tortor ligula. Pellentesque habitant morbi tristique  
senectus et netus et malesuada fames ac turpis egestas. Pellentesque eleifend libero in  
lorem fermentum auctor. Class aptent taciti sociosqu ad litora torquent per conubia  
nostra, per inceptos himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam  
adipiscing dictum orci, eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras  
auctor laoreet odio a iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus  
ac. Vivamus placerat quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut ut  
magna. Nullam tellus orci, consectetur ac consectetur et, malesuada sed dolor. Donec non  
libero sit amet risus venenatis laoreet. Donec ultrices, sapien eget placerat  
consectetur, magna libero gravida dolor, at condimentum dui lectus non lorem. Duis eget  
risus ante, at mattis ipsum. Vivamus nulla augue, placerat ac molestie nec, sollicitudin  
nec ante. Sed euismod, risus at sodales fringilla, lacus mi accumsan nunc, eget lobortis  
dolor neque sed arcu. Donec scelerisque mauris vitae lectus faucibus feugiat. Aenean  
purus odio, volutpat et feugiat eu, posuere sit amet nunc. Cras lectus orci, rhoncus ac  
mattis ut, ultrices id mi. Cras massa lorem, porttitor sed tempus et, congue quis dui.  
Fusce tristique faucibus ultricies. In metus felis, congue quis cursus nec, congue sit  
amet arcu. Quisque in molestie nibh. Curabitur at odio et tortor faucibus commodo a non  
augue. Nunc a augue a magna vehicula suscipit.</p>
```

```
</div>
```

```
</div>
```

```
<div class="page">
```

```
<span>presentation</span>
```

```
<div class="corps">
```

```
Présentation du contenu.
```

```
<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt  
pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis.  
Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh.  
Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus  
imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique  
metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam. In  
laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus sapien  
at erat. Maecenas risus nisl, malesuada vel lobortis sed, malesuada dapibus urna.  
Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio accumsan et.  
Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus, feugiat a lacinia  
vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus gravida. Pellentesque ut  
leo ligula, sit amet tincidunt nulla. Pellentesque ut odio eros, a euismod dui. Aenean  
faucibus turpis et justo vestibulum eget ultricies risus facilisis. Donec sodales  
dignissim mauris eget euismod. Etiam tellus magna, dignissim vel pulvinar non, auctor et  
neque. Aliquam erat volutpat. Integer iaculis lorem eget tortor varius faucibus. Mauris  
ac lectus non metus ornare laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi  
a urna mattis rhoncus semper est mollis. Nunc velit elit, varius id adipiscing at,  
gravida at lectus. Sed id odio justo, vel pretium velit. Proin ac dui ut risus rhoncus  
commodo dapibus sed metus. Maecenas commodo libero id augue congue non luctus mauris  
luctus. Ut neque magna, porttitor vel elementum eu, congue vitae sapien. Nunc ornare,  
enim a luctus rhoncus, tortor velit bibendum sem, nec rutrum libero orci hendrerit  
velit. Aenean in eros ac orci sollicitudin suscipit. Nunc lacinia gravida imperdiet.  
Morbi convallis mi at magna lacinia id feugiat magna dictum. Vivamus eleifend nibh eu
```



```
nunc imperdiet tempus. Quisque tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus. Fusce aliquam tempus augue. Nam nec tortor ligula. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Pellentesque eleifend libero in lorem fermentum auctor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.</p>
</div></div>
```

```
<div class="page">
  <span>page1</span>
  <div class="corps">
    Page 1 :<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis. Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh. Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam. In laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus sapien at erat. Nam nec tortor ligula. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Pellentesque eleifend libero in lorem fermentum auctor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam adipiscing dictum orci, eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras auctor laoreet odio a iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus ac. Vivamus placerat quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut magna. Nullam tellus orci, consectetur ac consectetur et, malesuada sed dolor. Donec non libero sit amet risus venenatis laoreet.</p>
  </div>
</div>
```

```
<div class="page">
  <span>page2</span>
  <div class="corps">
    Page 2 : <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis. Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh. Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam. In laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus sapien at erat. Maecenas risus nisl, malesuada vel lobortis sed, malesuada dapibus urna. Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio accumsan et. Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus, feugiat a lacinia vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus gravida. Pellentesque ut leo ligula, sit amet tincidunt nulla. Pellentesque ut odio eros, a euismod dui. Aenean faucibus turpis et justo vestibulum eget ultricies risus facilisis. Donec sodales dignissim mauris eget euismod. Etiam tellus magna, dignissim vel pulvinar non, auctor et neque. Aliquam erat volutpat. Integer iaculis lorem eget tortor varius faucibus. Mauris ac lectus non metus ornare laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi la urna mattis rhoncus semper est mollis. Nunc velit elit, varius id adipiscing at, gravida at lectus. Sed id odio justo, vel pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed metus. Maecenas commodo libero id augue congue non luctus mauris luctus. Ut neque magna, porttitor vel elementum eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus, tortor velit bibendum sem, nec rutrum libero orci hendrerit velit. Aenean in eros ac orci sollicitudin suscipit. Nunc lacinia gravida imperdiet. Morbi convallis mi at magna lacinia id feugiat magna dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus. Fusce aliquam tempus augue. Nam nec tortor ligula.
```



```

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis
egestas. Pellentesque eleifend libero in lorem fermentum auctor. Class aptent taciti
sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vestibulum id
est dui, ac adipiscing nisi. Nullam adipiscing dictum orci, eget faucibus dolor dapibus
et. Ut euismod sodales aliquet. Cras auctor laoreet odio a iaculis. Vivamus dapibus
egestas massa, a lobortis neque cursus ac. Vivamus placerat quam lorem.</p>
</div>
</div>

<div class="page">
  <span>Album</span>
  <div class="corps">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt
    pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis.
    Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh.
    Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus
    imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique
    metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet. Etiam tellus magna,
    dignissim vel pulvinar non, auctor et neque. Aliquam erat volutpat. Integer iaculis
    lorem eget tortor varius faucibus. Mauris ac lectus non metus ornare laoreet et sed
    elit. Vivamus nec quam lacus. Nunc lobortis mi a urna mattis rhoncus semper est mollis.
    Nunc velit elit, varius id adipiscing at, gravida at lectus. Sed id odio justo, vel
    pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed metus. Maecenas commodo
    libero id augue congue non luctus mauris luctus. Ut neque magna, porttitor vel elementum
    eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus, tortor velit bibendum sem,
    nec rutrum libero orci hendrerit velit. Aenean in eros ac orci sollicitudin suscipit.
    Nunc lacinia gravida imperdiet. Morbi convallis mi at magna lacinia id feugiat magna
    dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque tellus augue, rutrum a
    placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed erat ultrices imperdiet.
    Donec sit amet ante ut orci suscipit eleifend sed vel risus. Fusce aliquam tempus augue.
    Nam nec tortor ligula. Pellentesque habitant morbi tristique senectus et netus et
    malesuada fames ac turpis egestas. Pellentesque eleifend libero in lorem fermentum
    auctor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos
    himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam adipiscing dictum orci,
    eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras auctor laoreet odio a
    iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus ac. Vivamus placerat
    quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut magna. Nullam tellus
    orci, consectetur ac consectetur et, malesuada sed dolor. Donec non libero sit amet
    risus venenatis laoreet.
    Donec ultrices, sapien eget placerat consectetur, magna libero gravida dolor, at
    condimentum dui lectus non lorem. Duis eget risus ante, at mattis ipsum. Vivamus nulla
    augue, placerat ac molestie nec, sollicitudin nec ante. Sed euismod, risus at sodales
    fringilla, lacus mi accumsan nunc, eget lobortis dolor neque sed arcu. Donec scelerisque
    mauris vitae lectus faucibus feugiat. Aenean purus odio, volutpat et feugiat eu, posuere
    sit amet nunc. Cras lectus orci, rhoncus ac mattis ut, ultrices id mi. Cras massa lorem,
    porttitor sed tempus et, congue quis dui. Fusce tristique faucibus ultricies. In metus
    felis, congue quis cursus nec, congue sit amet arcu. Quisque in molestie nibh. Curabitur
    at odio et tortor faucibus commodo a non augue. Nunc a augue a magna vehicula suscipit.
  </div><!-- fin du corps -->
</div><!-- fin de la page -->
</div><!-- fin du site -->
</body>
</html>

```

## Avec jQuery, une librairie d'instructions javascript

Nous allons créer un site défilant dans une division d'id galerie.

La division #galerie contient trois divisions :

- deux boutons (div#next et div#previous) à droite et à gauche,
- une très large division (div#slide {width:10000px;}) contenant les pages à faire défiler horizontalement.

## CSS

```
div#galerie {height:700px;
  position:relative; /* Ses divisions seront positionnées par rapport à ce
parent. */
  width:1000px;
  margin:auto;
  overflow:hidden; /* Ce qui dépasse sera caché. */}

div#next {right:0;}
div#previous {left:0;}

.defile {position:absolute; /* Par rapport au parent, on se positionne de manière
absolue. */
  top:40%; width: 40px; border:1px solid gray;z-index:5;
  cursor:pointer;background:white;}
.defile:hover {color:gray; border:2px dotted gray;}

div#slide {position:relative; /* Les pages seront positionnées par rapport à ce
parent. */
  width:10000px; /* Cette division est très large, elle contient les pages. */
  margin:40px 0em 0 0em;}
.page { float:left; /* Les pages sont à gauche les unes des autres. */
  padding:0 0em;
  width:1000px;}
```

## Javascript

La librairie jQuery est déclarée en tête du document (<script src=http://). Ensuite viennent nos instructions utilisant l'objet jQuery \$().

L'objet \$(document) est le document, l'objet \$("#ex") est l'élément du document d'id="ex". Il existe des méthodes associées à ces objets :

- \$("#divisionId").width() <- retourne la largeur de la division, en pixels.
- \$("#divisionId").hide() <- attribut style="display:none" à la division.
- \$(document).ready() <- s'exécute lorsque le document est prêt.

```
$(document).ready(function() { // Lorsque le document est prêt,
  $ = new slider("#galerie"); // crée l'objet slider().
});

var slider = function(id){ // L'objet slider est une fonction.

  var self=this;
```

```
    this.div = $(id);
    this.saut=this.div.width();

    this.next=this.div.find('#next');
    this.prev=this.div.find('#previous');
    this.prev.hide(); // Cache le bouton previous en attendant que l'utilisateur clique
sur next.

    this.largeurDesPages=0; // Largeur totale des div.page contenues dans #slide
    this.slide=this.div.find('#slide');
    this.slide.find('.page').each(function(){
        self.largeurDesPages+=$(this).width();
    });

    this.nbSaut=Math.ceil(this.largeurDesPages/this.saut - 1);

    this.courant=0; // Le numéro de la page courante.

    //Evenement lorsqu'on clique sur next
    this.next.click(function(){
        if (self.courant<self.nbSaut){
            self.courant++;
            if (self.courant==1) {self.prev.show();}
            self.slide.animate({
                left:-self.saut*self.courant,
            }, 900);
            //alert(self.courant);
            if (self.courant>=self.nbSaut) {
                self.next.animate({
                    width:"0",
                    border:"0"
                }, 1100 );
            }
        }
    });

    //Evenement lorsqu'on clique sur previous
    this.prev.click(function(){
        if (self.courant>0){
            if (self.courant==self.nbSaut) {self.next.animate({
                width:"40px",
                border:"1px solid gray"
            }, 700 );}

            self.courant--;
            self.slide.animate({
                left:-self.saut*(self.courant),
            }, 900);
            //alert(self.courant);
        } else {
            self.prev.hide("slow");
        }
    });

    alert("largeur totale des pages : "+this.largeurDesPages+"px\n"
        +"saut : "+this.saut+"px\n"
        +"nbSaut: "+this.nbSaut
        );
```

```
}
}
```

Ces instructions exécutent la méthode ready() de l'objet \$(document). (\$(document).ready(function(){}));

Lorsque le document est prêt c'est-à-dire que tout est chargé, la fonction est exécutée. Cette fonction crée l'objet slider().

slider() modifie les styles des divisions #slide, #next et #previous en utilisant les méthodes jQuery animate(), show() et hide().

- Créez ce document \*.html et ouvrez-le dans un navigateur :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr" dir="ltr">
<head>
  <title></title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
  <meta name="robots" content="index, follow" />
  <link rel="shortcut icon" href="/favicon.ico" />
  <link rel="copyright" href="http://creativecommons.org/licenses/by-sa/3.0/" />

  <!-- load jQuery library -->
  <script src="http://ajax.googleapis.com/ajax/libs/jquery/1.3.2/jquery.min.js"
type="text/javascript"></script>

  <script type="text/javascript">
  //
  alert(' Copie ici les instructions javascript'
    +' ou declare ton fichier javascript externe avec &lt;script src=""
type="text/javascript"&gt;&lt;/script&gt;.' );
  //]]&gt;
  &lt;/script&gt;

  &lt;style type="text/css"&gt;
  Copie les déclarations de style ici ou crée un fichier css et utilise &lt;style src=""
type="text/css"&gt;&lt;/script&gt;
  &lt;/style&gt;
&lt;/head&gt;

&lt;body &gt;

&lt;div id="galerie"&gt;
  &lt;div id="previous" class="defile"&gt;prev&lt;/div&gt;&lt;!-- premier bouton, --&gt;

  &lt;div id="next" class="defile"&gt;next&lt;/div&gt;    &lt;!-- deuxieme bouton, --&gt;

  &lt;div id="slide"&gt;                                &lt;!-- et le slide contenant plusieurs
pages. --&gt;

    &lt;div class="page" id="apropos"&gt;
      About page.
      &lt;p&gt;Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed tincidunt
pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem placerat quis.
Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel dictum nibh.
Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed lectus</pre>
</div>
<div data-bbox="25 932 122 949" data-label="Page-Footer">
<p>28 sur 161</p>
</div>
<div data-bbox="795 932 979 949" data-label="Page-Footer">
<p>16/09/2018 à 22:19</p>
</div>
```

```
imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec tristique  
metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam. In  
laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus sapien  
at erat. Maecenas risus nisl, malesuada vel lobortis sed, malesuada dapibus urna.  
Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio accumsan et.  
Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus, feugiat a lacinia  
vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus gravida. Pellentesque ut  
leo ligula, sit amet tincidunt nulla. Pellentesque ut odio eros, a euismod dui. Aenean  
faucibus turpis et justo vestibulum eget ultricies risus facilisis. Donec sodales  
dignissim mauris eget euismod. Etiam tellus magna, dignissim vel pulvinar non, auctor et  
neque. Aliquam erat volutpat. Integer iaculis lorem eget tortor varius faucibus. Mauris  
ac lectus non metus ornare laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi  
a urna mattis rhoncus semper est mollis. Nunc velit elit, varius id adipiscing at,  
gravida at lectus. Sed id odio justo, vel pretium velit. Proin ac dui ut risus rhoncus  
commodo dapibus sed metus. Maecenas commodo libero id augue congue non luctus mauris  
luctus. Ut neque magna, porttitor vel elementum eu, congue vitae sapien. Nunc ornare,  
enim a luctus rhoncus, tortor velit bibendum sem, nec rutrum libero orci hendrerit  
velit. Aenean in eros ac orci sollicitudin suscipit. Nunc lacinia gravida imperdiet.  
Morbi convallis mi at magna lacinia id feugiat magna dictum. Vivamus eleifend nibh eu  
nunc imperdiet tempus. Quisque tellus augue, rutrum a placerat tincidunt, pharetra nec  
odio. Donec suscipit lacus sed erat ultrices imperdiet. Donec sit amet ante ut orci  
suscipit eleifend sed vel risus. Fusce aliquam tempus augue. Nam nec tortor ligula.  
Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis  
egestas. Pellentesque eleifend libero in lorem fermentum auctor. Class aptent taciti  
sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vestibulum id  
est dui, ac adipiscing nisi. Nullam adipiscing dictum orci, eget faucibus dolor dapibus  
et. Ut euismod sodales aliquet. Cras auctor laoreet odio a iaculis. Vivamus dapibus  
egestas massa, a lobortis neque cursus ac. Vivamus placerat quam lorem. Aliquam ut magna  
porta enim lobortis sollicitudin ut magna. Nullam tellus orci, consectetur ac  
consectetur et, malesuada sed dolor. Donec non libero sit amet risus venenatis laoreet.  
Donec ultrices, sapien eget placerat consectetur, magna libero gravida dolor, at  
condimentum dui lectus non lorem. Duis eget risus ante, at mattis ipsum. Vivamus nulla  
augue, placerat ac molestie nec, sollicitudin nec ante. Sed euismod, risus at sodales  
fringilla, lacus mi accumsan nunc, eget lobortis dolor neque sed arcu. Donec scelerisque  
mauris vitae lectus faucibus feugiat. Aenean purus odio, volutpat et feugiat eu, posuere  
sit amet nunc. Cras lectus orci, rhoncus ac mattis ut, ultrices id mi. Cras massa lorem,  
porttitor sed tempus et, congue quis dui. Fusce tristique faucibus ultricies. In metus  
felis, congue quis cursus nec, congue sit amet arcu. Quisque in molestie nibh. Curabitur  
at odio et tortor faucibus commodo a non augue. Nunc a augue a magna vehicula suscipit.
```

```
</p>
```

```
</div>
```

```
<div class="page" id="contact">
```

```
contact page. <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.  
Sed tincidunt pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem  
placerat quis. Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel  
dictum nibh. Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed  
lectus imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec  
tristique metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam.  
In laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus  
sapien at erat. Maecenas risus nisl, malesuada vel lobortis sed, malesuada dapibus urna.  
Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio accumsan et.  
Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus, feugiat a lacinia  
vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus gravida. Pellentesque ut  
leo ligula, sit amet tincidunt nulla. Pellentesque ut odio eros, a euismod dui. Aenean  
faucibus turpis et justo vestibulum eget ultricies risus facilisis. Donec sodales  
dignissim mauris eget euismod. Etiam tellus magna, dignissim vel pulvinar non, auctor et  
neque. Aliquam erat volutpat. Integer iaculis lorem eget tortor varius faucibus. Mauris  
ac lectus non metus ornare laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi
```

```
la urna mattis rhoncus semper est mollis. Nunc velit elit, varius id adipiscing at,
gravida at lectus. Sed id odio justo, vel pretium velit. Proin ac dui ut risus rhoncus
commodo dapibus sed metus. Maecenas commodo libero id augue congue non luctus mauris
luctus. Ut neque magna, porttitor vel elementum eu, congue vitae sapien. Nunc ornare,
enim a luctus rhoncus, tortor velit bibendum sem, nec rutrum libero orci hendrerit
velit. Aenean in eros ac orci sollicitudin suscipit. Nunc lacinia gravida imperdiet.
Morbi convallis mi at magna lacinia id feugiat magna dictum. Vivamus eleifend nibh eu
nunc imperdiet tempus. Quisque tellus augue, rutrum a placerat tincidunt, pharetra nec
odio. Donec suscipit lacus sed erat ultrices imperdiet. Donec sit amet ante ut orci
suscipit eleifend sed vel risus. Fusce aliquam tempus augue. Nam nec tortor ligula.
Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis
egestas. Pellentesque eleifend libero in lorem fermentum auctor. Class aptent taciti
sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Vestibulum id
est dui, ac adipiscing nisi. Nullam adipiscing dictum orci, eget faucibus dolor dapibus
et. Ut euismod sodales aliquet. Cras auctor laoreet odio a iaculis. Vivamus dapibus
egestas massa, a lobortis neque cursus ac. Vivamus placerat quam lorem. Aliquam ut magna
porta enim lobortis sollicitudin ut magna. Nullam tellus orci, consectetur ac
consectetur et, malesuada sed dolor. Donec non libero sit amet risus venenatis laoreet.
Donec ultrices, sapien eget placerat consectetur, magna libero gravida dolor, at
condimentum dui lectus non lorem. Duis eget risus ante, at mattis ipsum. Vivamus nulla
augue, placerat ac molestie nec, sollicitudin nec ante. Sed euismod, risus at sodales
fringilla, lacus mi accumsan nunc, eget lobortis dolor neque sed arcu. Donec scelerisque
mauris vitae lectus faucibus feugiat. Aenean purus odio, volutpat et feugiat eu, posuere
sit amet nunc. Cras lectus orci, rhoncus ac mattis ut, ultrices id mi. Cras massa lorem,
porttitor sed tempus et, congue quis dui. Fusce tristique faucibus ultricies. In metus
felis, congue quis cursus nec, congue sit amet arcu. Quisque in molestie nibh. Curabitur
at odio et tortor faucibus commodo a non augue. Nunc a augue a magna vehicula suscipit.
</p>
</div>
```

```
<div class="page" id="presentation">
```

```
Présentation du contenu. <p>Lorem ipsum dolor sit amet, consectetur
adipiscing elit. Sed tincidunt pharetra tempor. Morbi dignissim sollicitudin orci, non
molestie lorem placerat quis. Quisque et eros turpis, vel suscipit neque. Nulla
facilisi. Mauris vel dictum nibh. Vestibulum et erat erat, ut dictum tellus. Quisque vel
mi magna. Nunc sed lectus imperdiet diam vestibulum facilisis eu a enim. Duis vitae
magna urna, nec tristique metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit
amet nunc quam. In laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu
volutpat metus sapien at erat. Maecenas risus nisl, malesuada vel lobortis sed, malesuada
dapibus urna. Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio
accumsan et. Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus,
feugiat a lacinia vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus
gravida. Pellentesque ut leo ligula, sit amet tincidunt nulla. Pellentesque ut odio
eros, a euismod dui. Aenean faucibus turpis et justo vestibulum eget ultricies risus
facilisis. Donec sodales dignissim mauris eget euismod. Etiam tellus magna, dignissim vel
pulvinar non, auctor et neque. Aliquam erat volutpat. Integer iaculis lorem eget tortor
varius faucibus. Mauris ac lectus non metus ornare laoreet et sed elit. Vivamus nec quam
lacus. Nunc lobortis mi a urna mattis rhoncus semper est mollis. Nunc velit elit, varius
id adipiscing at, gravida at lectus. Sed id odio justo, vel pretium velit. Proin ac dui
ut risus rhoncus commodo dapibus sed metus. Maecenas commodo libero id augue congue non
luctus mauris luctus. Ut neque magna, porttitor vel elementum eu, congue vitae sapien.
Nunc ornare, enim a luctus rhoncus, tortor velit bibendum sem, nec rutrum libero orci
hendrerit velit. Aenean in eros ac orci sollicitudin suscipit. Nunc lacinia gravida
imperdiet. Morbi convallis mi at magna lacinia id feugiat magna dictum. Vivamus eleifend
nibh eu nunc imperdiet tempus. Quisque tellus augue, rutrum a placerat tincidunt,
pharetra nec odio. Donec suscipit lacus sed erat ultrices imperdiet. Donec sit amet ante
ut orci suscipit eleifend sed vel risus. Fusce aliquam tempus augue. Nam nec tortor
ligula. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac
turpis egestas. Pellentesque eleifend libero in lorem fermentum auctor. Class aptent
```



```

taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos.
Vestibulum id est dui, ac adipiscing nisi. Nullam adipiscing dictum orci, eget faucibus
dolor dapibus et. Ut euismod sodales aliquet. Cras auctor laoreet odio a iaculis.
Vivamus dapibus egestas massa, a lobortis neque cursus ac. Vivamus placerat quam lorem.
Aliquam ut magna porta enim lobortis sollicitudin ut ut magna. Nullam tellus orci,
consectetur ac consectetur et, malesuada sed dolor. Donec non libero sit amet risus
venenatis laoreet. Donec ultrices, sapien eget placerat consectetur, magna libero gravida
dolor, at condimentum dui lectus non lorem. Duis eget risus ante, at mattis ipsum.
Vivamus nulla augue, placerat ac molestie nec, sollicitudin nec ante. Sed euismod, risus
at sodales fringilla, lacus mi accumsan nunc, eget lobortis dolor neque sed arcu. Donec
scelerisque mauris vitae lectus faucibus feugiat. Aenean purus odio, volutpat et feugiat
eu, posuere sit amet nunc. Cras lectus orci, rhoncus ac mattis ut, ultrices id mi. Cras
massa lorem, porttitor sed tempus et, congue quis dui. Fusce tristique faucibus
ultricies. In metus felis, congue quis cursus nec, congue sit amet arcu. Quisque in
molestie nibh. Curabitur at odio et tortor faucibus commodo a non augue. Nunc a augue a
magna vehicula suscipit. </p>

```

```
</div>
```

```
<div class="page" id="page1">
```

```

page1 text. <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit.
Sed tincidunt pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem
placerat quis. Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel
dictum nibh. Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed
lectus imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec
tristique metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam.
In laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus
sapien at erat. Maecenas risus nisl, malesuada vel lobortis sed, malesuada dapibus urna.
Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio accumsan et.
Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus, feugiat a lacinia
vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus gravida. Pellentesque ut
leo ligula, sit amet tincidunt nulla. Pellentesque ut odio eros, a euismod dui. Aenean
faucibus turpis et justo vestibulum eget ultricies risus facilisis. Donec sodales
dignissim mauris eget euismod.
Etiam tellus magna, dignissim vel pulvinar non, auctor et neque. Aliquam erat volutpat.
Integer iaculis lorem eget tortor varius faucibus. Mauris ac lectus non metus ornare
laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi a urna mattis rhoncus
semper est mollis. Nunc velit elit, varius id adipiscing at, gravida at lectus. Sed id
odio justo, vel pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed metus.
Maecenas commodo libero id augue congue non luctus mauris luctus. Ut neque magna,
porttitor vel elementum eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus,
tortor velit bibendum sem, nec rutrum libero orci hendrerit velit. Aenean in eros ac
orci sollicitudin suscipit. Nunc lacinia gravida imperdiet. Morbi convallis mi at magna
lacinia id feugiat magna dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque
tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed
erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus.
Fusce aliquam tempus augue. Nam nec tortor ligula. Pellentesque habitant morbi tristique
senectus et netus et malesuada fames ac turpis egestas. Pellentesque eleifend libero in
lorem fermentum auctor. Class aptent taciti sociosqu ad litora torquent per conubia
nostra, per inceptos himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam
adipiscing dictum orci, eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras
auctor laoreet odio a iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus
ac. Vivamus placerat quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut ut
magna. Nullam tellus orci, consectetur ac consectetur et, malesuada sed dolor. Donec non
libero sit amet risus venenatis laoreet. Donec ultrices, sapien eget placerat
consectetur, magna libero gravida dolor, at condimentum dui lectus non lorem. Duis eget
risus ante, at mattis ipsum. Vivamus nulla augue, placerat ac molestie nec, sollicitudin
nec ante. Sed euismod, risus at sodales fringilla, lacus mi accumsan nunc, eget lobortis
dolor neque sed arcu. Donec scelerisque mauris vitae lectus faucibus feugiat. Aenean
purus odio, volutpat et feugiat eu, posuere sit amet nunc. Cras lectus orci, rhoncus ac

```

```
mattis ut, ultrices id mi. Cras massa lorem, porttitor sed tempus et, congue quis dui.
Fusce tristique faucibus ultricies. In metus felis, congue quis cursus nec, congue sit
amet arcu. Quisque in molestie nibh. Curabitur at odio et tortor faucibus commodo a non
augue. Nunc a augue a magna vehicula suscipit.      </p>
```

```
</div>
```

```
<div class="page" id="page2">
```

```
page2 text<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed
tincidunt pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem
placerat quis. Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel
dictum nibh. Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed
lectus imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec
tristique metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam.
In laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus
sapien at erat.Maecenas risus nisl, malesuada vel lobortis sed, malesuada dapibus urna.
Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio accumsan et.
Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus, feugiat a lacinia
vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus gravida. Pellentesque ut
leo ligula, sit amet tincidunt nulla. Pellentesque ut odio eros, a euismod dui. Aenean
faucibus turpis et justo vestibulum eget ultricies risus facilisis. Donec sodales
dignissim mauris eget euismod.
```

```
Etiam tellus magna, dignissim vel pulvinar non, auctor et neque. Aliquam erat volutpat.
Integer iaculis lorem eget tortor varius faucibus. Mauris ac lectus non metus ornare
laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi a urna mattis rhoncus
semper est mollis. Nunc velit elit, varius id adipiscing at, gravida at lectus. Sed id
odio justo, vel pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed metus.
Maecenas commodo libero id augue congue non luctus mauris luctus. Ut neque magna,
porttitor vel elementum eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus,
tortor velit bibendum sem, nec rutrum libero orci hendrerit velit. Aenean in eros ac
orci sollicitudin suscipit. Nunc lacinia gravida imperdiet. Morbi convallis mi at magna
lacinia id feugiat magna dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque
tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed
erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus.
Fusce aliquam tempus augue.Nam nec tortor ligula. Pellentesque habitant morbi tristique
senectus et netus et malesuada fames ac turpis egestas. Pellentesque eleifend libero in
lorem fermentum auctor. Class aptent taciti sociosqu ad litora torquent per conubia
nostra, per inceptos himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam
adipiscing dictum orci, eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras
auctor laoreet odio a iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus
ac. Vivamus placerat quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut ut
magna. Nullam tellus orci, consectetur ac consectetur et, malesuada sed dolor. Donec non
libero sit amet risus venenatis laoreet.Donec ultrices, sapien eget placerat
consectetur, magna libero gravida dolor, at condimentum dui lectus non lorem. Duis eget
risus ante, at mattis ipsum. Vivamus nulla augue, placerat ac molestie nec, sollicitudin
nec ante. Sed euismod, risus at sodales fringilla, lacus mi accumsan nunc, eget lobortis
dolor neque sed arcu. Donec scelerisque mauris vitae lectus faucibus feugiat. Aenean
purus odio, volutpat et feugiat eu, posuere sit amet nunc. Cras lectus orci, rhoncus ac
mattis ut, ultrices id mi. Cras massa lorem, porttitor sed tempus et, congue quis dui.
Fusce tristique faucibus ultricies. In metus felis, congue quis cursus nec, congue sit
amet arcu. Quisque in molestie nibh. Curabitur at odio et tortor faucibus commodo a non
augue. Nunc a augue a magna vehicula suscipit.      </p>
```

```
</div>
```

```
<div class="page" id="photos">
```

```
Album<p>Lorem ipsum dolor sit amet, consectetur adipiscing elit. Sed
tincidunt pharetra tempor. Morbi dignissim sollicitudin orci, non molestie lorem
placerat quis. Quisque et eros turpis, vel suscipit neque. Nulla facilisi. Mauris vel
dictum nibh. Vestibulum et erat erat, ut dictum tellus. Quisque vel mi magna. Nunc sed
lectus imperdiet diam vestibulum facilisis eu a enim. Duis vitae magna urna, nec
```



```

tristique metus. Nunc vel nisi erat, nec commodo felis. Pellentesque sit amet nunc quam.
In laoreet, urna vitae elementum rutrum, velit eros blandit lacus, eu volutpat metus
sapien at erat. Maecenas risus nisl, malesuada vel lobortis sed, malesuada dapibus urna.
Suspendisse potenti. Suspendisse lacinia pharetra libero, non semper odio accumsan et.
Aenean quis augue eu ipsum tempus molestie. Suspendisse risus tellus, feugiat a lacinia
vitae, hendrerit quis odio. Aenean vulputate luctus dui rhoncus gravida. Pellentesque ut
leo ligula, sit amet tincidunt nulla. Pellentesque ut odio eros, a euismod dui. Aenean
faucibus turpis et justo vestibulum eget ultricies risus facilisis. Donec sodales
dignissim mauris eget euismod.
Etiam tellus magna, dignissim vel pulvinar non, auctor et neque. Aliquam erat volutpat.
Integer iaculis lorem eget tortor varius faucibus. Mauris ac lectus non metus ornare
laoreet et sed elit. Vivamus nec quam lacus. Nunc lobortis mi a urna mattis rhoncus
semper est mollis. Nunc velit elit, varius id adipiscing at, gravida at lectus. Sed id
odio justo, vel pretium velit. Proin ac dui ut risus rhoncus commodo dapibus sed metus.
Maecenas commodo libero id augue congue non luctus mauris luctus. Ut neque magna,
porttitor vel elementum eu, congue vitae sapien. Nunc ornare, enim a luctus rhoncus,
tortor velit bibendum sem, nec rutrum libero orci hendrerit velit. Aenean in eros ac
orci sollicitudin suscipit. Nunc lacinia gravida imperdiet. Morbi convallis mi at magna
lacinia id feugiat magna dictum. Vivamus eleifend nibh eu nunc imperdiet tempus. Quisque
tellus augue, rutrum a placerat tincidunt, pharetra nec odio. Donec suscipit lacus sed
erat ultrices imperdiet. Donec sit amet ante ut orci suscipit eleifend sed vel risus.
Fusce aliquam tempus augue. Nam nec tortor ligula. Pellentesque habitant morbi tristique
senectus et netus et malesuada fames ac turpis egestas. Pellentesque eleifend libero in
lorem fermentum auctor. Class aptent taciti sociosqu ad litora torquent per conubia
nostra, per inceptos himenaeos. Vestibulum id est dui, ac adipiscing nisi. Nullam
adipiscing dictum orci, eget faucibus dolor dapibus et. Ut euismod sodales aliquet. Cras
auctor laoreet odio a iaculis. Vivamus dapibus egestas massa, a lobortis neque cursus
ac. Vivamus placerat quam lorem. Aliquam ut magna porta enim lobortis sollicitudin ut ut
magna. Nullam tellus orci, consectetur ac consectetur et, malesuada sed dolor. Donec non
libero sit amet risus venenatis laoreet. Donec ultrices, sapien eget placerat
consectetur, magna libero gravida dolor, at condimentum dui lectus non lorem. Duis eget
risus ante, at mattis ipsum. Vivamus nulla augue, placerat ac molestie nec, sollicitudin
nec ante. Sed euismod, risus at sodales fringilla, lacus mi accumsan nunc, eget lobortis
dolor neque sed arcu. Donec scelerisque mauris vitae lectus faucibus feugiat. Aenean
purus odio, volutpat et feugiat eu, posuere sit amet nunc. Cras lectus orci, rhoncus ac
mattis ut, ultrices id mi. Cras massa lorem, porttitor sed tempus et, congue quis dui.
Fusce tristique faucibus ultricies. In metus felis, congue quis cursus nec, congue sit
amet arcu. Quisque in molestie nibh. Curabitur at odio et tortor faucibus commodo a non
augue. Nunc a augue a magna vehicula suscipit.</p>
    </div><!-- fin de la derrière .page du slide -->

</div>    <!-- fin de la division #slide -->
</div>    <!-- fin de la division #galerie -->
</body>
</html>

```

## Un menu déroulant en CSS

Le CSS permet également d'interagir avec la souris, via notamment la pseudo-classe `hover`.

L'apparence déclarée à la pseudo-classe `hover` est active lorsque la souris est sur l'élément.

### CSS declarations

```

div#menu {width:100px;}
div#menu ul {padding: 0; width: 100px; border:1px solid gray; margin:0px; }

```

```

div#menu ul li {position:relative; list-style: none; border-bottom:1px solid gray;}

div#menu ul ul {position: absolute; top: 0; left: 100px; display:none;} /* ul ul =>
display:none */

div#menu ul.niveau1 li.sousmenu:hover ul.niveau2,
li.sousmenu:hover => display */
div#menu ul.niveau2 li.sousmenu:hover ul.niveau3 {display:block;}

div#menu li a {text-decoration: none; color:gray;}
div#menu li.sousmenu {background: #F0F0F0;}

```

*ul.niveau2* est affiché (*display:block;*) lorsque la souris est dessus *li.sousmenu* (*li.sousmenu:hover*) contenue dans *div#menu ul.niveau1*.

### Créez ce document \*.html et ouvrez-le dans votre navigateur

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org
/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr" dir="ltr">
<head>
<title></title>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta name="robots" content="index,follow" />
<link rel="shortcut icon" href="/favicon.ico" />
<link rel="copyright" href="http://creativecommons.org/licenses/by-sa/3.0/" />
<script type="text/javascript">
function afficher(a) {
    var e=document.getElementById(a);
    e.style.display="inline"; /* Applique display:inline à l'element d'id
a */
    cacher_sauf(e); /* Applique display:none aux autres
éléments. */
}

function cacher_sauf(a) {
    var els=document.getElementsByClassName('page');
    for(i in els) {
        if (els[i] != a) {els[i].style.display="none";}
    }
}
</script>
<style type="text/css">
a{cursor:pointer;}
#sommaire { font-size:18pt;
margin:2em;
margin-right:0;
border-bottom:1px dotted grey;}
#contenu {margin:1em;}

div#menu {width:100px;}
div#menu ul {padding: 0; width: 100px; border:1px solid gray; margin:0px; }
div#menu ul li {position:relative; list-style: none; border-bottom:1px solid gray;}
div#menu ul ul {position: absolute; top: 0; left: 100px; display:none;} /* ul ul =>

```

```

display:none */



```

```
    </div>
<div class="page" id="page1" style="display:none;">
  page1 text.
</div>
<div class="page" id="page2" style="display:none;">
  page2 text
</div>
<div class="page" id="photos" style="display:none;">
  Album
</div>
</div>
</body>
</html>
```

# Présentation

## IDE

---

Un environnement de développement intégré (EDI ou IDE en anglais pour Integrated Development Environment) est un programme regroupant un éditeur de texte, des outils automatiques de fabrication, et souvent un débogueur. Il en existe pour pratiquement tous les langages informatique et le Javascript n'échappe pas à la règle. Les IDE pour javascript sont souvent pourvus de fonctionnalité pour écrire en HTML.

---

Les utilités principales des IDE sont :

- **L'analyse du code (Syntaxique, fonctionnel, objet...)**

Permet d'éviter de faire des erreurs et de parcourir rapidement les différentes parties du code.

- **Le déboggeur**

Indique où se trouve les erreurs et de quel type d'erreur il s'agit.

- **L'auto complétion**

Permet de visionner les méthodes et paramètres des objets javascript que vous utilisez.

- **La documentation**

L'aide de chaque fonction est accessible rapidement.

- **Les générateurs de code**

Des exemples de code permettent d'obtenir des programmes très rapidement.

## Libre

- Aptana (Basé sur Eclipse)
- NetBeans

## Propriétaire

- Dreamweaver

# Syntaxe

Les deux grandes structures du langage sont les instructions et les structures de contrôle, qui toutes deux contiennent des expressions.

La syntaxe du JavaScript est largement inspirée des langages C et Java. Il faut ajouter à cela le fait que la plupart des opérateurs utilisent les mêmes noms ou symboles que dans ces langages.

L'avantage de cette similarité de forme est qu'elle permet de s'appuyer sur ce que l'on connaît déjà dans ces langages très répandus pour coder ou pour lire en JavaScript, ou réciproquement.

L'inconvénient est que cette similarité de forme dissimule des différences nombreuses, subtiles ou fondamentales, et des fonctionnalités inexistantes dans les langages classiques auxquels on est tenté de comparer le JavaScript. Et inversement, on peut vouloir appliquer en JavaScript des mots et des syntaxes propres à ces langages qui lui ressemblent, et être déçu de leur absence.

# Ligne d'instruction

## Composition d'une instruction

---

Une instruction JavaScript est constituée d'une combinaison d'expressions (identificateurs, variables, opérateurs, expressions littérales, appels de fonctions), parfois précédée d'une déclaration (var, void). Elle peut être suivie ou entrecoupée de commentaires.

## Délimitation des instructions

---

Une instruction s'achève lorsqu'une autre instruction commence.

En règle générale, on rencontre une ligne d'instruction par ligne de texte. Pour écrire plusieurs instructions sur une même ligne, il est nécessaire de les séparer avec un point-virgule ( ; ). Par habitude, on met souvent des point-virgules à la fin de toutes les instructions en JavaScript mais il faut savoir que leur absence n'est pas une erreur de syntaxe.

### Juxtaposition d'instructions

```
s1="trois "; s2="mots "; s3="concaténés"; alert(s1+s2+s3);
```

Les mêmes instructions sur plusieurs lignes, sans point-virgules (JavaScript "comprend" que chaque nouvelle ligne est une instruction différente) :

### Fin d'instruction automatique en fin de ligne

```
s1="trois "  
s2="mots "  
s3="concaténés"  
alert(s1+s2+s3)
```

Une instruction répartie sur plusieurs lignes (JavaScript "comprend" que chaque nouvelle ligne prolonge l'instruction)

### Répartition possible sur plusieurs lignes

```
alert(  
  "trois "  
  +"mots "  
  +"concaténés")
```

Intégration de commentaires :

### Commentaires

```
alert( "trois " // Commentaire en fin de ligne
```

```
+ "mots " /* Commentaire multiligne */  
+ /* peut être placé au milieu d'une instruction */ "concaténés"  
);
```

Ces quatre exemples provoquent l'affichage d'une boîte d'avertissement munie de la chaîne "trois mots concaténés". Leur présentation diffère en fonction des besoins d'expressivité et de lisibilité du code tel que voulu par le programmeur.

## Regroupement des lignes d'instructions en blocs

Les lignes d'instructions sont regroupées en blocs d'instructions délimités par des accolades.

**Note :** JavaScript **différencie les majuscules et les minuscules**. Il convient donc d'être vigilant sur ce point lors de l'écriture des lignes d'instructions pour s'éviter de longues séances de débogage.



# Ordre d'évaluation

## Scripts

---

Lors du chargement d'une page Web, dès que le navigateur rencontre un script, il l'exécute avant de charger la suite du document HTML. Si le script contient des instructions qui doivent être évaluées immédiatement et qui manipulent le document, ce dernier risque de ne pas être prêt.

Lorsque des balises `<script>` possèdent l'attribut `defer`, le navigateur est supposé télécharger ces scripts en parallèle (donc plus rapidement), et attendre d'avoir fini le chargement de la page avant de les exécuter dans l'ordre des balises.

Lorsque qu'elles possèdent l'attribut `async`, le navigateur les télécharge et exécute en parallèle, dans un ordre qui dépend du contexte, donc aléatoire et potentiellement avant la fin du chargement de la page. Il ne faut donc jamais l'utiliser pour charger des bibliothèques ayant des dépendances, comme jQuery.

S'il faut manipuler le document au moment du chargement, il est donc fortement recommandé de déclencher le code *après* avoir déclaré toute partie du document qui doit être utilisée. Il reste bien sûr possible de déclarer la plupart du code dans des fonctions avant le corps du document, comme cela est fait couramment, et soit d'appeler les fonctions dans un script ultérieur, soit de placer ces instructions dans la méthode `document.onload`.

## Instructions

---

À la base, chaque fichier ou fragment de code JavaScript est évalué de haut en bas : chaque instruction est exécutée avant de passer à la suivante. Bien entendu, les fonctions, structures de contrôle, et d'autres éléments du langage font que certaines instructions sont évaluées à un autre moment, et/ou plusieurs fois, ou pas du tout.

## Expressions

---

Les expressions sont évaluées de gauche à droite, en respectant la priorité des opérateurs et les parenthèses. Lorsqu'une expression est composée d'opérations de priorité identique (par exemple, plusieurs additions) la première opération en partant de la gauche est effectuée (et ses opérandes sont évaluées), puis son résultat est utilisé comme opérande de gauche de l'opération suivante (et l'opérande de droite est évaluée), et ainsi de suite.

```
str = 1 + 3 + "5" + -1 + 2
alert( str ) // affiche "45-12"
```

L'exemple ci-dessus montre que la première opération entre deux nombres se comporte normalement (1 et 3 font 4), la seconde se transforme en concaténation car une des opérandes est de type "string", et à partir de là le résultat de chaque opération est une chaîne donc toutes les additions suivantes se transforment en concaténations. Ce serait différent avec des parenthèses :

```
str = 1 + 3 + "5" + (-1 + 2)
alert( str ) // affiche "451"
```

Les opérations de priorité identique ne sont pas évaluées simultanément (techniquement ce ne serait pas possible d'ailleurs) et marginalement, cela peut avoir une influence sur le résultat. Par conséquent, les opérations que l'on appelle commutatives en mathématiques ne sont pas strictement commutatives en JavaScript.

```
u = v + (v = u) // équivalent à trois instructions séparées pour calculer la suite de
Fibonacci
// est différent de :
u = (v = u) + v // façon tordue d'écrire u*=2
```

Dans cet exemple, l'addition n'est pas commutative. Si l'opération  $v=u$  est placée à gauche de l'addition, elle est évaluée en premier et modifie l'autre opérande. En revanche, placée à droite, elle est évaluée *après*  $v$ , et donc la valeur de  $v$  utilisée par l'addition est l'ancienne valeur.

# Bloc d'instructions

Les blocs d'instructions JavaScript sont constitués d'instructions placées entre une accolade ouvrante ( { ) et une accolade fermante ( } ).

Un bloc d'instructions est obligatoirement introduit par ces déclarations (les accolades sont nécessaires même pour une seule instruction) :

- une déclaration de fonction
- une déclaration d'objet implicite with
- les déclarations try, catch et finally
- et par la structure de contrôle switch

Ces autres structures sont généralement suivies d'un bloc d'instruction (mais dans le cas d'une seule instruction les accolades sont facultatives) :

- une déclaration de boucle (for, while, do)
- un branchement conditionnel (if et else)

Dans tout autre cas, les accolades sont considérées comme les délimiteurs de l'écriture littérale d'un objet et non pas d'un bloc d'instructions.

## Exécution séquentielle

---

À l'intérieur d'un bloc, les instructions sont exécutées séquentiellement, à moins de l'apparition du mot réservé *return* (dans une fonction) ou du mot réservé *break* (dans certaines structures de branchement conditionnelles) ou de l'occurrence d'une erreur.

## Inclusion de blocs

---

Un bloc d'instructions peut lui-même contenir d'autres blocs, généralement des structures conditionnelles ou itératives, mais aussi des déclarations de fonctions imbriquées.

### Inclusion de blocs

```
function main()
{ // Le début du bloc principal de la fonction
  function locale()
  { // Début d'un bloc imbriqué
    alert("Exécution d'une sous fonction")
  }
}
```

# Identificateurs

La programmation javascript, comme tous les langages informatiques, demande au développeur de définir des identificateurs, et ceci dans deux cas :

- pour nommer une fonction utilisateur
- pour nommer une variable

Pour construire notre identificateur nous pouvons utiliser les caractères autorisés, et devons proscrire les caractères interdits. De plus, certaines combinaisons sont interdites. Notre identificateur ne doit pas correspondre à un mot réservé. Enfin, il est utile de se conformer à certaines recommandations en ce qui concerne le choix et l'orthographe de l'identificateur, pour renforcer l'expressivité du code.

## Principales règles

---

- Ne pas commencer par un chiffre,
- Ne pas utiliser de caractère interdit,
- Ne pas faire suivre un caractère accentué par un chiffre,
- Rester monobloc. Pas de caractère espace au milieu,
- Vérifier que l'identificateur n'est pas un mot réservé,
- Garder en tête que javascript différencie les majuscules des minuscules.

## Caractères autorisés

---

Tous les caractères sont autorisés, sauf ceux qui contreviennent aux règles précédentes et aux caractères interdits.

- Les lettres de l'alphabet, majuscules et minuscules,
- Les chiffres, sous réserve de ne pas commencer avec,
- Les caractères accentués, avec limitations,
- Certains caractères spéciaux : \$ \_
- Les séquences d'échappement unicode de forme `\uMMMM`

## Caractères interdits

---

- Les ponctuations et autres symboles spéciaux : - , . " ' @ : / + \* % = ~ ! ? < > ( ) [ ] { } # & | ` ^ ` €
- Les caractères de contrôles obtenus par séquence d'échappement,
- Les espaces.

Expression littérale >>

## Standardisation

---

Les identificateurs sont en particulier standardisés par l'ECMA 262<sup>[1]</sup>, au chapitre §7.6. Ce standard s'appuie sur la chapitre 5 de l'Unicode version 3<sup>[2]</sup>.

Sont en particulier autorisés les échappement Unicode, s'ils conduisent à un identificateur autorisé. Enfin, deux

identificateurs équivalent ne sont considérés identique que s'ils sont identiques binairement<sup>[3]</sup>.

Les versions suivants d'Unicode peuvent fonctionner mais ne sont pas nécessairement portables et compatibles.

## Notes de bas de page

---

1. Librement disponible sur internet
2. Lire éventuellement À la découverte d'Unicode.
3. Par exemple la séquence [ e; accent aigu ] (en deux points de code ) est différent de [ e accent aigu ] (en un seul point de code)

# Expression littérale

Dans un programme javascript, il est courant de devoir introduire des données sous forme littérale. Cela se produit principalement quand on renseigne une variable, ou quand on appelle une fonction.

Ce qui suit est valide pour le code javascript écrit dans les fichiers externes, ou entre les balises `<script>` `</script>`. En ce qui concerne le code associé aux événements (onclick, onload...), l'approche est différente.

```
ma_var = "une chaîne exprimée littéralement";  
mon_nombre = 12; // un nombre littéral;  
alert("Ce message est une chaîne littérale");
```

Ces expressions littérales peuvent être des chaînes, ou des nombres.

## Les chaînes sous forme littérale


Elles sont constituées de lettres, chiffres, ponctuation et symboles avec un mécanisme (échappement) permettant d'étendre cet ensemble à des caractères de contrôle, de formatage et des caractères spéciaux.

Une chaîne exprimée sous forme littérale est délimitée par des guillemets double ( " ) ou simple ( ' ).

La règle veut qu'une chaîne ouverte avec un guillemet double doit être fermée avec un guillemet double. De même, une chaîne ouverte avec un guillemet simple doit être fermée avec un guillemet simple.

La première complication se produit quand nous désirons inclure des guillemets dans notre chaîne.

Appelons une fonction avec une chaîne en paramètre :

 Ce code contient **une erreur volontaire** !

```
alert("le titre de l'article : "les joies de l'informatique");  
// ne marche pas
```

... à la place, on peut écrire:

```
alert("le titre de l'article : " +  
      "' + "les joies de l'informatique" + "' );
```

... mais c'est affreux.

Pour utiliser une solution élégante, nous avons recours au mécanisme d'échappement.

## Le mécanisme d'échappement

Le mécanisme d'échappement consiste à inclure dans les chaînes, des anti-slash ( \ ) suivis d'un caractère choisi pour sa fonction.

les caractères échappés sont en nombre limités et facile à mettre en œuvre.

## L'échappement des guillemets simples (') ou double (")


Précédé d'un anti-slash, les guillemets ne feront plus office de délimiteur de chaîne, et ceci permet de les inclure comme de simples symboles de ponctuation.

```
alert("Mon nom est \"Personne\""); // affiche : Mon nom est "Personne"
```

## Le retour à la ligne avec \n

Il est interdit d'aller à la ligne à l'intérieur d'une chaîne de caractère. Cependant, il peut être utile de formater une chaîne en la présentant sur plusieurs lignes. La séquence d'échappement \n réalise cela. Elle inclut en effet le caractère de contrôle "linefeed" (valeur 10 en décimal, 0A en hexadécimal) qui réalise le retour à la ligne avec la plupart des navigateurs graphique.

Par exemple, il n'est pas possible de formater une chaîne de caractère en utilisant le code suivant :

 Ce code contient **une erreur volontaire** !

```
alert("première,  
deuxième  
et troisième ligne");  
// ne marche pas
```

Il faudra utiliser à la place l'écriture suivante :

```
alert("première, \ndeuxième \net troisième ligne");
```

... affiche :

```
première,  
deuxième  
et troisième ligne
```

## La tabulation avec \t

La séquence d'échappement \t insère un caractère de tabulation dans la chaîne, et permet ainsi de formater l'affichage. Le caractère de tabulation est codé 9 en décimal et 09 en hexadécimal.

```
alert("colonne 1 \tcolonne 2 \tcolonne 3");
```

## L'inclusion de caractères unicode avec \u

Javascript, quand il est codé dans un fichier distinct, ou quand il est codé entre les balises <script> </script> ne supporte

pas les entités de la forme `&lt;` ou `&#160;`, ou toute autre forme d'entité.

L'échappement `\u` offre un mécanisme de remplacement dont la mise en œuvre nécessite de se référer à une table associant les codes hexadécimaux au symboles correspondant (par exemple).

Ces codes **hexadécimaux** sont présentés avec *4 (ou 5) chiffres hexadécimaux*, et la séquence d'échappement `\u` nécessite d'écrire ces 4 chiffres sous peine d'un déclenchement d'erreur.

```
alert("Rodrigue as-tu du c\u0153ur?");
```

... affiche Rodrigue as-tu du cœur?

ce qui est plus conforme à l'original que l'anti-typographique "coeur", utilisé à longueur d'internet.

L'inclusion de caractères Unicode permet l'accès aux symboles et à la citation de fragments utilisant d'autres alphabets que le nôtre.

De plus, il permet également l'inclusion des caractères de contrôle. Ainsi, `\n` et `\u000A` sont équivalents.

[Table Unicode \(wikipédia\)](#)

Unicode est toutefois plus vaste et peut être découvert au travers du livre Wikibooks [À la découverte d'Unicode](#).

## Les nombres sous forme littérale

---

Les nombres sont constitués de chiffres et de signes à l'exclusion du caractère espace. JavaScript permet d'exprimer littéralement les nombres entiers, réels, octals, hexadécimaux, exponentiels, sous forme positive ou négative.

### Les nombres négatifs

En plaçant le signe moins ( - ) devant le nombre, on obtient ce nombre en négatif, et ceci qu'il soit noté comme entier, réel, octal, hexadécimal ou exponentiel.

```
ma_var = -5;
```

### Les entiers

Ils s'écrivent tel quel, c'est la représentation par défaut, employant les chiffres 0 à 9.

```
ma_var = 12;
```

### Les réels

Le signe de séparation de la partie entière est le point ( . )

```
ma_var = 8.65;
```



## L'octal

Un nombre commençant par zero et ne comprenant que les chiffres de 0 à 7 sera interprété en octal.

```
ma_var = 0257;
```

## L'hexadécimal

Un nombre commençant par 0x (ou 0X) comprenant les chiffres et les lettres abcdef en minuscule, majuscule ou les deux, sera interprété en hexadécimal.

```
ma_var = 0xf044;
```

## La notation scientifique

En faisant suivre un nombre entier ou réel de la lettre e (ou E), elle-même suivie d'un entier positif ou négatif, on obtient un nombre en **notation scientifique**.

```
ma_var = 2.5e40; // Un grand nombre : 2,5 * 10 exposant 40  
ma_var = 50e-30 // Un tout petit nombre
```

## Taille des nombres

Le type **number** est au format 64 bits double précision.

## Les chaînes littérales dans les expressions associées aux évènements

A l'intérieur des balises html, il est possible d'intégrer du code JavaScript associé aux évènements, (comme onclick, onmouseover...). Javascript est à ce moment là interprété en dehors de son territoire propre, c'est à dire entre les balises <script> et </script> ou dans un fichier séparé. Sa rédaction obéit alors aux règles de rédaction des attributs des balises, qui n'autorisent pas tout.

```
<div onclick="alert('Je suis cliqué')">clicquez moi!</div>
```

Html, comme xhtml, ne reconnaît pas les séquences de type \uHHHH. Nous devons nous limiter aux caractères échappés classiques, amputés de (\" ) et (\' ). En effet, ces deux caractères ne seront pas interprétables comme valeur de la propriété de la balise.

Si nous voulons afficher **Je suis "cliqué"**, nous devons recourir aux entités.

Les entités sont un mécanisme permettant d'insérer n'importe quel caractère dans une chaîne.

Selon la référence du doctype html, nous aurons accès à des entités munies d'un identificateur. Sinon, sous réserve de disposer d'une **table adéquate**, nous pouvons reproduire n'importe quel caractère.

Sans aller plus-avant dans la description des entités, nous pouvons les décrire comme suit :

L'entité commence par le caractère & (esperluette ou "et commercial") et finit par un point-virgule. Entre ces deux symboles, on rencontre soit un identificateur ( **&lt;**, **&gt;**, **&amp;**), soit le code du caractère exprimé en **décimal** et précédé du signe sharp (#) (**&#160;**, **&#339;**).

```
<div onclick="alert('Je suis &#34;cliqué&#34;')"> // Affiche: Je suis "cliqué"  
    cliquez moi!  
</div>
```

... Mission accomplie.

# Commentaire

JavaScript permet d'insérer des commentaires dans les programmes, et ceci de deux manières :

- Syntaxe à double-slash (//)
- Syntaxe slash-étoile (/\* \*/)

## La syntaxe à double-slash.

---

Les caractères placés à droite du double-slash sont ignorés par l'interpréteur jusqu'à l'occurrence d'un retour à la ligne.

```
alert("message"); // Commentaire expliquant le rôle de l'instruction
```

## La syntaxe slash-étoile.

---

Les commentaires sont ouverts avec (/\*) et fermés avec (\*/) et donc ignorés par l'interpréteur.

```
/* Commentaire sur une ligne */  
/* Commentaire  
sur plusieurs  
lignes */  
alert("message"); // Cette ligne est exécutée
```

## Intégration dans les lignes d'instruction

---

Les deux syntaxes de commentaire permettent de multiples modes d'inclusion dans le code:

```
alert("deux " /*Le premier*/ + "mot" /*Le deuxième*/); // Inclusion dans les paramètres  
alert("deux " // Le premier  
+ "mot" /* Le deuxième*/); // Deux inclusions de commentaires
```

# Mots réservés

Cet article va passer en revue les mots réservés et constantes prédéfinies de JavaScript.

La norme [ECMA 262](http://www.ecma-international.org/publications/standards/Ecma-262.htm) (<http://www.ecma-international.org/publications/standards/Ecma-262.htm>), définit les mots réservés de JavaScript en deux catégories :

- les mots réservés (*break else new var case finally return void catch for switch while continue function this with default if throw delete in try do instanceof typeof*)
- les futurs mots réservés (*abstract enum int short boolean export interface static byte extends long super char final native synchronized class float package throws const goto private transient debugger implements protected volatile double import public*).

On ne doit pas utiliser un mot réservé comme identificateur. Cela provoque une erreur. Mais comme le langage différencie les majuscules et minuscules, il est possible de mettre en majuscule une lettre pour rendre l'identificateur utilisable.

On peut, avec la plupart des navigateurs utiliser un futur mot réservé comme identificateur, mais cela est déconseillé, car le programme ne fonctionnera probablement plus à l'apparition d'une version future de JavaScript.

Voici une description sommaire des mots réservés :

## Déclarations

---

### var

Déclaration d'une variable locale, avec ou sans affectation. À placer en début d'instruction.

```
var ma_variable;  
var ma_variable = "Je suis affecté";
```

Une variable locale est détruite automatiquement à la fin de la fonction où elle a été créée (sauf dans le cas des *closures*). Seul le code écrit dans cette fonction peut y accéder, et dans ce code, la variable est prioritaire sur tout autre identifiant homonyme. En revanche, elle ne peut être détruite par l'opérateur delete.

### let

Comme "var" mais restreint la portée au bloc, comme si ce bloc était situé dans une autre fonction. Exemple :

```
var a = 1;  
var b = 2;  
{  
  var a = 11;  
  let b = 22;  
}  
console.log(a); // 11  
console.log(b); // 2
```

## function

Déclaration d'une fonction utilisateur. C'est aussi un opérateur.

```
function ma_fonction()
{
    alert("Je suis exécuté dans ma_fonction");
}
```

## void

**void** est une déclaration placée obligatoirement en début d'instruction. **void** ne fait rien, ne retourne rien, et n'est utile que dans un seul contexte : les pseudo-URL `javascript:`. En effet, si l'instruction qui compose cet URL retourne une valeur, cette valeur remplace le contenu de la page, ce qui est souvent indésirable. On place alors la déclaration **void** pour que l'instruction n'ait aucune valeur.

```
<a href="javascript:void(maFonction())">...</a>
```

## with

Permet de simplifier la référence aux propriétés et méthodes des objets en précédant un bloc d'instruction dans lequel un objet devient implicite. La déclaration **with** rajoute en fait un niveau tout en haut de la pile de *scopes*.

```
with(navigator)
{
    alert(appName + " " + appVersion);
}
```

est équivalent à

```
alert(navigator.appName + " " + navigator.appVersion);
```

## Structures de contrôle

---

### return

Employé dans une fonction, provoque le retour au programme appelant avec renvoi facultatif d'un résultat.

```
function ma_fonction(aparam)
{
    s = "";
    if (aparam == "")
        return; // Sortie sans renvoyer de résultat
    else
        s = "Le résultat";
    return s; // Renvoie le contenu de s
}
```

```
}  
}
```

## if

Permet de définir l'exécution conditionnelle d'instructions ou d'un bloc d'instructions. Peut-être utilisé conjointement à `else`.

```
a = 5;  
if (a==5)  
    alert("la condition est remplie");
```

## else

Utilisé conjointement à `if`, permet d'exécuter des instructions alternativement au résultat de la condition spécifiée par `if`.

```
a = 5  
if (a==5)  
    alert("a est égal à 5");  
else  
    alert("a est différent de 5");
```

## switch

Utilisé conjointement à `case`, permet d'implanter un sélecteur de cas.

```
function test(condition)  
{  
    switch(condition)  
    {  
        case "1":  
            alert("condition='1'");  
            break;  
        case "3":  
            alert("condition='3'");  
            break;  
        case "5":  
            alert("condition='5'");  
            break;  
    }  
}
```

## case

Utilisé à l'intérieur d'un bloc switch, constitue un des choix du selecteur.

## break

Utilisé à l'intérieur d'un bloc switch, permet de sortir du sélecteur sans procéder tests suivants. Utilisé à l'intérieur d'une

boucle telle un bloc switch, permet de quitter la boucle immédiatement.

## default

Utilisé à l'intérieur d'un bloc switch, constitue l'option par défaut du sélecteur.

## for

Introduit une boucle itérative.

```
for (i=0;i<5;i++)
{
    alert(i);
}
```

For each :

```
var array = [1, 2, 3];
for (line in array)
{
    alert(line);
}
```

En tableau associatif :

```
var array = { "a" : 1, "b" : 2, "c": 3 };
for (line in array)
{
    alert('Clé : ' + line + ', valeur : ' + array[line]);
}
```

## do

Introduit une boucle itérative conditionnelle avec test effectué à chaque tour par "while" :

```
i=0
do
{
    i++
    alert(i);    // Affichera 1, puis 2, puis 3, puis 4, puis 5
}
while(i<5);
```

## while

Introduit une boucle itérative conditionnelle:

```
i=0
```

```
while(i<5)
{
    i++;      // Incréments i
    alert(i); // Affiche 1, puis 2, puis 3, puis 4, puis 5
}
```

## continue

Utilisé dans une boucle "for", "while" et "do", permet de sauter un tour.

```
for (i=-2;i<=2;i++)
{
    if (i==0) continue; // Si i == 0, on passe directement à 1
    alert(5/i); // On évite ainsi la division par zéro.
}
```

## Gestions des erreurs

---

### throw

Provoque une erreur personnalisée.

```
throw new Error("votre erreur");// a le même effet visuel que alert("votre erreur");
```

### try ... catch ... finally

Évalue un bloc (**try**) en capturant les erreurs, tente de les gérer (**catch**) si elles se produisent, et quoiqu'il arrive, évalue un dernier bloc (**finally**).

## Opérateurs

---

### in

Opérateur qui détermine l'appartenance à un objet. Dans une boucle for... in, permet de répéter des instructions pour chaque propriété d'un objet.

### new

Opérateur qui permet l'instanciation d'un objet.

```
var mon_tableau = new Array();
```

### instanceof



Cet opérateur permet de tester si une valeur est une instance d'un objet :

```
t = new Array();  
alert(t instanceof Array); // Affiche true  
alert(t instanceof Date); // Affiche false
```

## typeof

Cet opérateur renvoie le type de la variable placée à droite.

Les valeurs renvoyées par **typeof** correspondent aux types de variables JavaScript, soit : boolean string number function object undefined.

## delete

Cet opérateur permet de supprimer une propriété.

## Valeurs spéciales

---

### this

Dans le constructeur ou une méthode d'un objet, c'est une référence à l'objet. Hors de ce contexte, **this** référence l'objet global *window*.

### true

Valeur booléenne "vrai".

```
if (true)  
{  
    alert("Je m'affiche toujours");  
}
```

### false

Valeur booléenne "faux".

```
if (ma_bool == false)  
    alert("C'est pas vrai...")
```

### null

Valeur de type "object" qui ne référence rien.

## undefined

C'est la valeur de toutes les variables inexistantes ou supprimées.

## Futurs mots réservés

---

Les versions futures de JavaScript intégreront peut-être un ou plusieurs mots réservés parmi la liste suivante.

```
abstract enum int short boolean export interface static byte
extends long super char final native class float
package const goto private debugger
implements protected double import public
```

Il est déconseillé de les employer comme identificateurs dans les programmes actuels, sous peine de risquer un comportement bizarre dans le futur.

## Mots réservés en jQuery

---

### Liste des mots spécifiques en jQuery<sup>[1]</sup>

Nom	Fonction
<code>\$()</code> , <code>jQuery()</code>	Fonction de sélection, et d'initialisation
<code>holdReady()</code>	Retiens ou relâche un évènement jQuery
<code>noConflict()</code>	Supprime un mot réservé
<code>sub()</code>	Crée une copie modifiable d'un objet jQuery
<code>when()</code>	Planifie des objets
<code>each()</code>	Sélectionne chaque élément d'un objet
<code>parents()</code>	Sélectionne tous les éléments parents
<code>parent()</code>	Sélectionne le parent direct
<code>children()</code>	Sélectionne tous les enfants
<code>closest()</code>	Sélectionne l'élément le plus proche avec une certaine caractéristique (évite des <code>parent().parent().parent()</code> ).
<code>hasClass()</code>	Renvoie si l'élément à une classe
<code>addClass()</code>	Ajoute une classe
<code>removeClass()</code>	Retire une classe
<code>toggleClass()</code>	Échange deux classes
<b>Sélecteurs</b>	
<code>:animated</code>	Sélectionne les éléments animés
<code>:hidden</code>	Sélectionne les éléments cachés
<code>:visible</code>	Sélectionne les éléments visibles
<code>:first</code>	Sélectionne le premier élément
<code>:last</code>	Sélectionne le dernier élément
<code>:even</code>	Sélectionne les éléments pairs
<code>:odd</code>	Sélectionne les éléments impairs
<code>:eq(index)</code>	Sélectionne le n <sup>ème</sup> élément
<code>:gt(index)</code>	Sélectionne les éléments supérieurs au n <sup>ème</sup>
<code>:lt(index)</code>	Sélectionne les éléments inférieurs au n <sup>ème</sup>
<code>:input</code>	Sélectionne les champs modifiables
<code>:text</code>	Sélectionne les champs texte
<code>:radio</code>	Sélectionne les boutons radio
<code>:checkbox</code>	Sélectionne les cases à cocher
<code>:checked</code>	Sélectionne les boutons radio et cases à cocher cochés

Nom	Fonction
:first-child	Sélectionne le premier élément enfant
:last-child	Sélectionne le dernier élément enfant
parent > child	Sélectionne le premier enfant d'un parent
Méthodes <sup>[2]</sup>	
.attr()	Affiche ou remplace la valeur d'un attribut
.val()	Affiche ou remplace la valeur d'un champ
.html()	Affiche ou remplace l'objet par de l'HTML
.text()	Affiche ou remplace l'objet par du texte
.prepend()	Ajoute le paramètre avant l'objet, dans la balise
.append()	Ajoute le paramètre après l'objet, dans la balise
.prependTo()	Ajoute l'objet avant le paramètre, dans la balise
.appendTo()	Ajoute l'objet après le paramètre, dans la balise
.before()	Ajoute le paramètre avant l'objet
.after()	Ajoute le paramètre après l'objet
.insertBefore()	Ajoute l'objet avant le paramètre
.insertAfter()	Ajoute l'objet après le paramètre
.wrap()	Ajoute les balises en paramètre autour de l'objet
.wrapInner()	Ajoute les balises en paramètre autour de l'objet à l'intérieur
.wrapAll()	Ajoute les balises en paramètre autour des objets
.clone()	Duplique un objet
.empty()	Vide un objet
.remove()	Retire un objet
.removeAttr()	Retire un attribut
.replaceWith()	Remplace l'objet par le paramètre
.replaceAll()	Remplace le paramètre par l'objet

Entrer `$( '.data' )` dans la console du navigateur (F12) affiche les éléments de la page dans un format pertinent au JS.

## Références

---

- <http://api.jquery.com/category/core/>
- <http://tutorials.jenkov.com/jquery/dom.html>



# Variables

Les variables permettent d'instancier un objet et de manipuler cette instance à l'aide d'un identificateur (le nom de la variable). Tous les identificateurs en JavaScript sont des références à un objet. Javascript étant un langage au typage dynamique, le type des objets n'est pas déclaré explicitement et il peut changer automatiquement.

## Typage dynamique

---

Il est donc possible de déclarer une variable sans lui définir de type. Ou plutôt, une variable déclarée de telle manière sera du type `undefined`.

Tout au long de son utilisation, cette variable pourra changer de type sans que l'interpréteur n'y trouve rien à redire.

```
var ma_versatile; // est undefined
ma_versatile = "Je deviens une chaîne"; // devient string
ma_versatile = 5; // devient number
ma_versatile = new Array(); // devient object
```

Il n'est en fait pas nécessaire de déclarer une variable (avec le mot-clé **var**). La différence est expliquée ci-après.

## Identificateur

---

Les règles d'élaboration des identificateurs sont décrites à [ce lien](#). Pour résumer, les identificateurs comportent des caractères alphanumériques et le tiret soulignant (`_`). Ils ne peuvent commencer par un chiffre, et les majuscules comptent.

## Portée

---

### Propriétés

L'instanciation directe (sans mot-clé `var`) d'un objet avec un identificateur donne souvent l'impression de créer une variable globale. En réalité, la notion de "variable globale" n'existe pas en JavaScript. Toutes les variables sont, sauf quand on les déclare avec **var**, des propriétés d'un objet. Par défaut, cet objet est implicitement l'objet **window** (et non, **window** n'est pas une variable globale qui ferait exception à la règle car il est une de ses propres propriétés, c'est-à-dire que **window** est une propriété de **window**).

Comme **window** reste l'objet implicite la plupart du temps, une variable créée ainsi pourra être employée comme si elle était globale :

```
ma_variable = "une valeur";

function maFonction()
{
  mon_autre_variable = "une autre valeur"
}

maFonction()
```

```
alert(mon_autre_variable) // affiche "une autre valeur"
```

La variable `mon_autre_variable` est certes déclarée dans une fonction, mais il faut lire `"window.mon_autre_variable"` donc c'est une propriété de `window` et elle ne sera pas détruite automatiquement à la fin de la fonction.

## Sous-entendre un objet

Pour économiser du code, il peut être intéressant de sous-entendre un autre objet que **window**. Cela se fait explicitement avec un bloc **with**. Cette déclaration ne fonctionne pas aussi simplement qu'en Basic, par exemple. L'objet est en fait placé en haut de la *scope chain* dans le bloc d'instruction, presque comme si ses propriétés étaient des variables locales d'une fonction (voir [Programmation JavaScript/Scope chain](#)). Lorsqu'on utilise un identificateur dans un bloc **with**, la résolution se fait aussitôt qu'une des conditions suivantes est remplie :

1. l'identificateur est une propriété existante de l'objet sous-entendu
2. l'identificateur existe ailleurs dans la *scope chain*, en partant du haut (par exemple, une variable locale)
3. l'identificateur est une propriété existante de `window`

Si aucune condition n'est remplie et que l'expression affecte une valeur à la variable, cette variable est créée comme propriété de l'objet global. Si l'expression ne fait qu'utiliser la valeur, elle retournera simplement **undefined**.

```
c=true
d="propriété de window"

monObjet={a: 0, d: "propriété de monObjet"}

with(monObjet)
{
  alert(a) // affiche la valeur de monObjet.a : "0"
  a=2 // monObjet.a existe et va être utilisé
  // si on veut utiliser window.a, il faut le faire explicitement
  window.a=5

  b="texte" // b n'est encore défini nulle part donc window.b est créé
  // si on veut créer monObjet.b, il faut le faire explicitement
  monObjet.b="autre chose"

  delete c // monObjet.c n'existe pas, mais window.c va être utilisé

  alert(d) // affiche "propriété de monObjet"
  // si on veut l'autre, il faut être explicite
  alert(window.d) // affiche "propriété de window"
}
```

## Variables locales

À l'aide de la déclaration **var**, il est possible de définir une variable locale, qui n'est pas une propriété d'un autre objet. Cette déclaration peut avoir lieu n'importe où dans le corps d'une fonction, pas nécessairement au début, mais seules les utilisations ultérieures de l'identificateur feront référence à la variable locale puisqu'elle n'existait pas avant.

Comme toute variable locale, elle est automatiquement détruite à la fin de la fonction, bien que sa valeur puisse être

sauvegardée dans certains cas (voir [Programmation JavaScript/Closure](#)). Par contre, elle ne peut pas être détruite par l'opérateur `delete`.

```
function foo()
{
  var loc_1 = 2           // n'a pas d'existence en dehors de foo()
  for (i = 0 ; i < loc_1; i++)
  {
    var loc_2 = 2 * loc_1; // loc_2 est disponible pour toute la fonction foo()
    alert(loc_2);
  }
  alert(loc_2);           // S'affiche sans problème
}
alert(loc_2); // Provoque une erreur
```

Si la déclaration `var` est utilisée en-dehors de toute fonction, la variable sera utilisable dans tout le script.

## Types

Javascript est faiblement typé, mais typé quand même. Voici une revue des types que peut prendre une variable. Il est possible à tout moment de vérifier le type avec l'opérateur `typeof`.

### undefined

Une variable est du type `undefined` dans trois cas:

- Après déclaration `var` sans affectation

```
var maNouvelleVar; // Déclarée mais pas affectée
alert(typeof maNouvelleVar); // Affiche undefined
```

- Si on lui affecte explicitement la valeur `undefined`, ou qu'on lui affecte le résultat d'une expression qui retourne cette valeur.

```
var jeSaisPas = undefined
var caNExistePas = "impossible".enFrancais
alert (typeof jeSaisPas) // Affiche undefined
alert (typeof caNExistePas) // Affiche undefined
```

- Si elle n'a pas été déclarée ni utilisée, ou bien qu'elle a été effacée par l'opérateur `delete`.

```
alert(typeof varPasDeclaree); // Affiche undefined (si elle n'a effectivement pas été affectée)
monEphemere = "exister";
delete monEphemere; // la vie est courte
alert (typeof monEphemere); // Affiche undefined

alert(varPasDeclaree) // erreur !!
```

À noter : dans les deux premiers cas, la variable existe et possède une valeur (`undefined`). On peut lire cette valeur



sans provoquer d'erreur. Dans le troisième cas cependant, la variable n'existe pas : l'opérateur `typeof` retourne "undefined", mais lire directement la valeur de la variable provoque une erreur.

## number

JavaScript réunit en un seul type les entiers, petits et grands, les décimaux, et les réels, flottants ou pas.

```
r = 1/3;
alert(r);           // affiche 0.333333333333
alert(typeof r);   // affiche number

n = 4/2;
alert(n);           // affiche 2
alert(typeof n);   // affiche number

3.6e5               // 360 000 en notation scientifique
0x40                // 64 en notation hexadécimale
0100                // 64 en notation octale
```

Concrètement, les nombres sont stockés sur 64 bits, avec une mantisse de 53 bits. Cela permet des valeurs entières jusqu'à 9 007 199 254 740 991, à partir duquel on commence à perdre de la précision jusqu'à  $2^{1024}-1$  qui est la valeur flottante la plus élevée représentable en JavaScript.

En fait, JavaScript inclut également dans le type **number** deux valeurs spéciales. **Infinity**, un nombre signé, est renvoyé dès que le résultat dépasse la capacité de stockage de **number**, mais aussi dans des cas limites simples. Dans les cas indéfinissables, ou bien lorsqu'une opération qui retourne un nombre ne peut pas le faire, c'est la valeur **NaN** (Not A Number) qui est renvoyée (voir les [opérateurs arithmétiques](#), [les méthodes de Number](#) et [les méthodes de Math](#) pour les cas qui produisent ces deux valeurs).

```
1/0 == Infinity
-2 / Infinity == 0
0/0 == NaN
Number("texte") == NaN
```

## string

Le type **string** stocke une chaîne de caractères. C'est aussi le type le plus "faible" du langage : n'importe quelle valeur peut être convertie en chaîne, et en cas de doute avec l'opérateur `+`, c'est une concaténation qui aura lieu, pas une addition.

```
s = "Une chaîne de caractères";
alert(typeof s); // Affiche "string"
```

Tous les objets possèdent une méthode générique `toString()` "magique" qui est appelée lorsqu'il faut convertir leur valeur en chaîne. Elle peut bien sûr être écrasée par une méthode plus spécifique.

Les chaînes de caractères en JavaScript sont encadrées par des guillemets droits (*double quotes*, `"`) ou des *single quotes* (`'`), au choix. Il n'y a pas de distinction entre des chaînes définies avec l'un ou l'autre, mais si on commence une chaîne

par un type de guillemet il faut la finir par le même type. À l'intérieur d'une chaîne, les séquences d'échappement connues fonctionnent comme partout :

- `\n` : *line feed*, saut de ligne Unix
- `\r` : *carriage return*, saut de ligne Mac OS classique
- `\t` : tabulation
- `\\` : *backslash*
- `'` ou `"` : un guillemet si la chaîne commence par ce caractère

etc.

Enfin, JavaScript utilise des chaînes de caractères Unicode multi-octets. Quel que soit l'encodage du fichier où est écrit le script, il est toujours possible d'inclure n'importe quel caractère dans une chaîne, soit en récupérant une chaîne d'un autre fichier, soit en créant une chaîne avec la méthode statique `String.fromCharCode()`, soit en utilisant la séquence d'échappement :

- `\xHH` : HH est le code hexadécimal du caractère dans l'encodage courant
- `\uHHHH` : HHHH est le code hexadécimal du caractère dans le tableau Unicode. Il ne peut y avoir moins de 4 chiffres après la séquence `\u`.

## boolean

Une variable de type boolean accepte deux valeurs : vrai et faux.

```
b = true;
alert(typeof b); // Affiche "boolean"
c = (5 == 3); // est faux, mais l'expression est booléenne.
alert(typeof c); // Affiche "boolean"
```

Dans une comparaison, les valeurs suivantes sont équivalentes (opérateur `==`) à **false** :

- 0 : le nombre zéro et ses variations, du style 0.000, 0x0, 0e12, etc.
- "" : la chaîne vide
- "0" : la chaîne contenant le nombre zéro, et toutes ses variations, comme pour le nombre
- [] ou **new Array()** : un tableau vide, et en fait tout objet dont la conversion en chaîne est l'une des valeurs ci-dessus

Par ailleurs, les valeurs suivantes donnent **false** si on les convertit (explicitement ou pas) en booléen, bien qu'elle ne soient pas toutes équivalentes par comparaison :

- 0 et ses variations
- ""
- null
- NaN
- undefined

Comme ces listes le montrent, une valeur peut être équivalente à **false** sans être considérée comme fautive elle-même ! Et inversement, une valeur considérée comme fautive n'est pas forcément équivalente au booléen **false**.

Cela peut sembler surprenant et illogique à première vue (et source de quelques bugs) mais il faut comprendre l'origine

de ces différences :

- les valeurs **null**, **NaN** et **undefined** refusent simplement d'être équivalentes à n'importe quoi sauf elles-mêmes. Donc l'opérateur `==` retournera systématiquement **false** si une des opérands est **null**, **NaN** ou **undefined** et que l'autre n'est pas strictement identique. Cependant, à choisir entre vrai ou faux (lors d'une conversion en **Boolean**, il n'y a pas d'autre choix), ces valeurs sont plutôt **false**.
- les chaînes "0" et ses variantes sont... des chaînes, et **false** est un booléen. JavaScript ne peut pas comparer directement deux types différents. Il pourrait convertir la chaîne en booléen, mais préfère convertir les deux en **number** (le type **number** est plus "fort"). Le résultat est `0 == 0` la comparaison est donc vraie. Cependant, "0" reste une chaîne non-vide, et si on la convertit en booléen, on obtient **true**.
- la chaîne "false" est aussi une chaîne, mais ne peut pas être convertie en **number**. JavaScript, en dernier recours, convertit alors l'autre opérande en **string** et se retrouve à comparer `"false"=="false"`, ce qui est vrai. Par contre, comme pour "0", la chaîne n'est pas vide et sa conversion en booléen donne **true**.
- le tableau vide [] est un tableau, donc JavaScript le convertit en chaîne pour le comparer à **false**. Or, la méthode `toString()` d'un tableau retourne la fusion de tous les éléments, séparés par des virgules. Comme il n'y a aucun élément dans le tableau, le résultat est la chaîne vide "", qui est équivalente à **false**. Par contre, un tableau est de type objet et sa conversion en booléen donne **true**.

Une valeur qui n'est pas équivalente à **false** ne sera pas pour autant équivalente à **true** par comparaison, à commencer par **null**, **NaN** et **undefined** mais aussi toute valeur qui n'est pas équivalente à l'une des conversions possibles de **true** (la chaîne "true", la chaîne ou le nombre 1). Par contre, toutes les valeurs qui ne donnent pas **false** par conversion en **boolean** donnent **true**, sans exception.

Attention, les valeurs de type **object**, y compris les instances des classes **String**, **Number** et **Boolean** créées avec le constructeur **new** sont toujours converties en **true**, même lorsque leur valeur est "", 0 ou **false** (la seule exception est **null** qui est de type **object** et dont la conversion en booléen est **false**). Cependant la comparaison avec **false** fonctionne de la même façon qu'avec des valeurs littérales.

## object

Une variable de type **object** assume pleinement sa nature. Les autres types de variables, dits "scalaires", se comportent comme s'ils étaient eux-mêmes une valeur. Ainsi, lorsqu'on passe une variable scalaire à une expression, sa valeur est clonée. En revanche, passer une variable de type **object** ne fait que copier la référence, et la valeur n'est pas clonée.

L'existence de "classes" en JavaScript est discutée. Le langage ne possède pas de déclaration de classe en tant que telle. Pour cette raison, l'utilisation de ce terme sera mise entre guillemets.

Il existe trois manières de créer une variable de type **object**.

- implicitement par l'expression littérale d'une valeur non-scalaire (valable uniquement quand une expression littérale existe pour la "classe" voulue)
- explicitement avec **new** et le constructeur (valable pour toutes les "classes" sauf les objet du DOM)
- indirectement pour les objets du DOM (et on ne peut pas utiliser **new** avec eux)

L'instanciation explicite d'un objet grâce à un constructeur s'effectue avec l'opérateur **new**.

```
{}t = new Array();
```

```
alert(typeof t) // Affiche "object"
```

Les expressions littérales de chaînes, nombres et booléens créent des variables scalaires alors que le constructeur de ces types crée un objet (explicitement de type **object** et donc strictement différents d'une valeur scalaire équivalente). Mais les expressions littérales de fonctions, tableaux et expressions rationnelles créent des objets, car ce ne sont pas des types scalaires. Ces objets, instanciés sans utilisation de **new**, sont néanmoins strictement identiques à leur équivalent produit par l'appel d'un constructeur. Par exemple :

```
// valeur scalaire vs. objet
"chaîne" !== new String("chaîne")
2.34 !== new Number(2.34)

// objet vs. objet... c'est pareil !
["a", "b", "c"] === new Array("a", "b", "c")
/(\w+) ?\(/i === new RegExp("(\\w+) ?\\(", "i")
```

On peut créer un objet générique (qui sera simplement une instance de la "classe" **Object**) avec la notation littérale. Cela consiste à écrire, entre accolades, une liste de couples clé : valeur séparés par des virgules, la clé pouvant être un identifiant ou bien une chaîne, et la valeur pouvant être n'importe quelle valeur. La clé et la valeur sont séparés par deux points (':').

```
objetLitteral = {
  nom : "Mac OS X",
  version : 10.5,
  "nom de code" : "Leopard"
}
```

Attention, dans l'exemple ci-dessus, on ne pourra accéder à la dernière propriété ("nom de code") de l'objet qu'avec la notation tableau (avec des crochets), pas la notation objet (avec des points) car ce n'est pas un identifiant valide (il y a des espaces dedans).

Le DOM (aussi bien W3C que IE) définit certaines "classes" : **HTMLElement**, **HTMLTextNode**, **HTMLDocument**, **CSSRuleSet**, **CSSRule**, etc. On doit utiliser les méthodes du DOM (comme `document.createElement()`) pour instancier ces objets (le "DOM 0" fait exception à cette règle et à bien d'autres).

Les autres "classes" d'objets ne peuvent être créés que par un constructeur. Il y a des constructeurs natifs tels que **Date**, et on peut définir un constructeur soi-même. Pour plus de détails, voir le chapitre consacré aux objets.

## fonction

La différence essentielle entre une variable de type fonction et une variable de type object, est qu'on peut appeler une fonction. Dans les deux sens du terme :

- une fonction peut être *nommée*, par un identifiant autre que le nom de la variable qui la référence
- une fonction peut être *invquée*

Attention : ne pas confondre le type de variable ("object", "function") avec les constructeurs homonymes (**Object**, **Function**). Une variable de type "object" peut avoir n'importe quel constructeur sauf **Function** et bien des

comportements différents, tous différents du comportement d'une fonction. Mais cette section est uniquement consacrée au **type** "function".

Contrairement aux autres types, le type "function" n'a pas de valeur spéciale (telle que **NaN** pour "number" ou **null** pour "object"). Il n'y a aucun moyen de convertir directement un autre type en fonction.

Une fonction est créée par l'expression littérale (exemple ci-dessous) ou bien par le constructeur `Function`.

```
function fibonacci( n )
{
  if(isNaN(n) || n<0)
    return undefined

  n=Math.floor(n)
  var u = 1, v = 0

  while(n-- )
    u = v + (v = u)

  return v
}
```

Cette déclaration crée une variable "fibonacci" qui référence la fonction, mais c'est aussi une opération qui retourne la référence de la fonction, que l'on peut ainsi affecter à une autre variable. Il est d'ailleurs possible de déclarer une fonction anonyme. De plus, toute nouvelle fonction reçoit automatiquement une propriété **prototype** identique au prototype de la fonction `Object`.

L'appel se fait en ouvrant et fermant une paire de parenthèses juste après une expression dont la valeur est de type "function". Cette expression n'est pas obligatoirement le nom d'une fonction, ni même un identifiant. Cette signification des parenthèses est prioritaire sur la signification générale (forcer une expression à être évaluée avant celles qui l'entourent).

```
f = fibonacci
delete fibonacci // efface seulement la variable, pas la fonction, qui est toujours
référéncée par f
x = f(10); // Constitue l'appel de la fonction fibonacci, en lui passant un argument
alert(f) // affiche le code de la fonction, y compris le nom donné dans sa déclaration,
"fibonacci"
```

Les opérateurs et méthodes qui acceptent des objets comme opérande ou argument fonctionnent pareil avec des fonctions, notamment le point pour accéder aux propriétés de la fonction. L'opérateur new n'accepte que des fonctions comme opérande.

Par défaut, la conversion d'une fonction en chaîne retourne le code source de la fonction, *préalablement reformaté par l'interpréteur JavaScript* : les point-virgules, sauts de ligne et indentations seront normalisés.

# Opérateurs

Les opérateurs javascript reprennent la plupart des opérateurs du langage C.

## Affectation (= et ses dérivés)

---

Le principal opérateur d'affectation est le signe égal (=). Il permet d'affecter une valeur à une variable.

```
ma_var = "La valeur affectée";
```

Comme en C, l'opérateur d'affectation copie la valeur de l'expression qui le suit (opérande de droite) dans la variable qui le précède (opérande de gauche). Contrairement au C, l'opération d'affectation ne retourne généralement pas `true` ou `false`, mais la valeur qui a été affectée. Dans le cas où l'affectation est impossible, JavaScript lance une exception.

Lorsque l'identifiant à gauche du signe = n'est pas une propriété existante, cette propriété est ajoutée automatiquement à l'objet global (`window`) et l'affectation se déroule normalement. Il n'y a jamais d'erreur lorsqu'on affecte une valeur à une variable d'un type différent car ce sont les valeurs elles-mêmes qui ont un type en JavaScript, pas les variables. L'affectation à une constante prédéfinie (comme `Math.PI`) ne provoque pas d'erreur mais la constante ne sera pas modifiée.

La seule cause d'erreur possible lors d'une affectation est lorsque l'opérande de gauche n'est pas une variable valide ou devient autre chose qu'une variable avant l'affectation :

```
2 = "valeur" // erreur de syntaxe
2 + a = "valeur" // erreur car l'addition est évaluée en premier et son résultat
n'est pas une variable
2 + (a = "valeur") // correct car l'affectation a lieu avant l'addition.
```

JavaScript reprend les raccourcis sténographiques du langage C en proposant les autres opérateurs d'affectation qui font précéder le signe égal par un opérateur binaire arithmétique (+, -, \*, / ou %), de concaténation (+), bit-à-bit (&, | ou ^) ou de décalage de bits (<<, >> ou <<<).

`ma_var = ma_var + 2` est remplacé avantageusement par `ma_var += 2`

`ma_var = ma_var / 2` peut s'écrire `ma_var /= 2`

`ma_var = ma_var << 2` peut s'écrire `ma_var <<= 2`

`ma_var = ma_var ^ 1` peut s'écrire `ma_var ^= 1`

et ainsi de suite.

## Concaténation (+)

---

Surchargé avec l'opérateur arithmétique d'addition, l'opérateur de concaténation permet de joindre des chaînes, ou des chaînes avec d'autres objets comme des nombres, qui sont alors automatiquement convertis en chaîne. Le résultat d'une concaténation est du type **string**.

La concaténation ne provoque jamais d'erreur car en JavaScript, absolument tout peut être converti en chaîne.

Attention, l'opérateur + fonctionne ainsi dès qu'au moins une des opérandes est de type "string" (y compris la chaîne vide ""), "function" ou "object" (à l'exception de `null`). Une chaîne représentant un nombre, par exemple la chaîne "17", sera considérée comme une chaîne et ne sera pas convertie automatiquement en nombre par cet opérateur.

#### Exemple de concaténation de chaînes

```
s1 = "un ";
s2 = "bout";
alert(s1 + s2); // Affiche "un bout"
```

#### Exemple de concaténation d'une chaîne avec un nombre

```
s = "Valeur du nombre : ";
n = 10;
alert(s + n); // Affiche "Valeur du nombre : 10"
```

#### Autre exemple de concaténation d'une chaîne avec un nombre

```
s = "17"; // Une chaîne représentant un nombre
n = 89;
alert(s + n); // Affiche "1789" et non pas 106
```

#### Exemple de concaténation d'objets divers

```
t = Array(1,2,3);
d = Date();
alert(t+d); // Affiche "1,2,3Fri Jan 20 17:13:32 2006"
```

## Opérateurs arithmétiques

---

### Addition (+)

En présence de deux opérandes de type **number** ou **boolean**, d'objets construits avec `new Number`, de l'objet `null`, ou de la valeur `undefined`, l'opérateur d'addition va effectuer la somme arithmétique des deux valeurs, après les avoir converties en nombre le cas échéant. Le résultat de l'addition est de type **number** :

Exemple :

- `3 + 1` donne 4

Dans le cas limite `Infinity + -Infinity`, le résultat est `NaN`.

**Attention** : si au moins l'un des deux objets ne fait pas partie de ceux mentionnés ci-dessus, alors l'opérateur se comporte comme un opérateur de concaténation :

Exemple :

- `3 + "bonjour"` donne `"3bonjour"` (et non 0)

Accessoirement, l'opérateur `+` avec une seule opérande à sa droite se comporte comme un opérateur unaire de conversion en nombre, comme la méthode `Number()`. C'est très utile pour forcer justement les autres `+` à additionner au lieu de concaténer :

```
"3" + 10 + "5" === "3105";  
+"3" + 10 + +"5" === 18 // les chaînes sont converties explicitement en nombres
```

## Soustraction (-)

Retourne la différence arithmétique entre deux valeurs de type **number** ou converties dans ce type.

Le résultat est lui-même de type `number`.

Comme pour tous les opérateurs arithmétiques, si l'une des deux opérandes, (ou les deux) n'est pas un nombre (**NaN**, soit à l'origine, soit parce que la conversion en nombre n'a pas été possible), l'opérateur renvoie **NaN** (Not a Number). Dans le cas limite `Infinity - Infinity`, le résultat est également `NaN`.

Accessoirement, le signe `-` avec une seule opérande à sa droite devient un opérateur unaire qui en change le signe (après l'avoir convertie en nombre si nécessaire).

## Multiplication (\*)

Retourne le produit arithmétique de deux valeurs de type **number** ou converties dans ce type.

Le résultat est lui-même de type `number`.

Comme pour tous les opérateurs arithmétiques, si l'une des deux opérandes, (ou les deux) n'est pas un nombre (**NaN**, soit à l'origine, soit parce que la conversion en nombre n'a pas été possible), l'opérateur renvoie **NaN** (Not a Number). Dans le cas limite `Infinity*0`, le résultat est également `NaN`.

## Division (/)

L'opérateur de division permet d'effectuer la division de deux entités de type **number** ou converties dans ce type.

Le résultat est lui-même de type `number`.

Comme pour tous les opérateurs arithmétiques, si l'une des deux opérandes, (ou les deux) n'est pas un nombre (**NaN**, soit à l'origine, soit parce que la conversion en nombre n'a pas été possible), l'opérateur renvoie **NaN** (Not a Number). Dans les cas limite `Infinity/Infinity` et `0/0`, le résultat est également `NaN`.

À noter : la division d'un autre nombre par zéro est possible et retourne le nombre **Infinity** sans provoquer d'erreur.

Attention : le signe `/` est également utilisé pour délimiter les expressions rationnelles. En cas d'ambiguïté (même en cas de saut de ligne sans point-virgule), JavaScript privilégie la division.

## Modulo (%)



L'opérateur '%' permet d'effectuer l'opération mathématique *modulo*

Il retourne l'opérande de gauche modulo l'opérande de droite, et les deux opérandes sont d'abord converties en **number** si elles étaient d'un autre type.

Le résultat est lui-même de type number.

Comme pour tous les opérateurs arithmétiques, si l'une des deux opérandes, (ou les deux) n'est pas un nombre (**NaN**, soit à l'origine, soit parce que la conversion en nombre n'a pas été possible), l'opérateur renvoie **NaN** (Not a Number).

De plus, n'importe quel nombre modulo zéro retournera NaN.

À noter : l'opérateur % accepte sans problème des nombres non-entiers pour les deux opérandes mais dans ce cas il y a souvent un problème de précision, par exemple `4%1.6` affichera `0.7999999999999998` au lieu de `0.8`.

### Exemple de modulo

```
nb = 7 % 3;    // nb prend la valeur 1 (car (7 - (2*3)) = 1)
nb = -7 % 3;   // nb prend la valeur -1
nb = 7 % -3;   // nb prend la valeur 1
nb = 4 % 1.6; // nb prend (presque) la valeur 0.8
```

Attention : le signe % est utilisé par l'encodage des URL. S'il est suivi de deux chiffres hexadécimaux, le navigateur convertira la séquence %xx en un caractère si le code se trouve dans une URL `javascript:`. Pour éviter ce problème, mettez l'opérande de droite entre parenthèses ou bien séparez-la de l'opérateur par un espace.

## Opérateurs logiques

### ET (&&)

L'opérateur logique &&, aussi appelé "prérequis", détermine si deux expressions sont "vraies" (au sens booléen). Le résultat est l'une des deux opérandes, et la conversion booléenne du résultat est identique au résultat de l'opérateur && dans les langages où il retourne simplement un booléen.

Cet opérateur est utilisé fréquemment dans les structures de contrôle if, while, etc :

```
if ((a == true) && (b==false)) alert("C'est vrai!");
```

L'opérateur && en JavaScript retourne l'opérande de gauche si elle est fausse (c'est-à-dire si sa conversion en booléen est égale à **false**) *sans évaluer l'expression de droite*, et sinon, évalue et retourne l'opérande de droite. Par conséquent, son résultat n'est généralement *pas un booléen* et && n'est pas strictement commutatif :

```
"chaîne" && 25    // retourne 25
25 && "chaîne"    // retourne "chaîne"
NaN && new Date() // retourne NaN et ne crée pas d'objet Date
new Date() && NaN // retourne NaN après avoir créé un objet Date
null && ""        // retourne null
"" && null        // retourne ""
```

Cela peut provoquer quelques comportements inattendus, par exemple :

```
if(("vrai" && "vrai") == true)    // cette condition ne sera jamais vraie, alors que ce
serait le cas en C
```

Mais on peut aussi en tirer partie pour condenser un code tel que celui-ci :

```
if( erreur )
    x=prompt( "Échec de l'opération. Recommencer ?" )
```

en le remplaçant par celui-ci :

```
erreur && (x=prompt( "Échec de l'opération. Recommencer ?" ))
```

D'où le nom "prérequis" : l'opérande de droite *requiert* que l'opérande de gauche soit vraie pour être évaluée et retournée.

## OU (||)

L'opérateur logique ||, aussi appelé "défaut", détermine si au moins une expression est "vraie" (au sens booléen). Le résultat est l'une des deux opérandes, et la conversion booléenne du résultat est identique au résultat de l'opérateur || dans les langages où il retourne simplement un booléen.

```
if ((a==true)|| (b==true)) alert("L'une des deux propositions est vrai. Peut-être les
deux");
```

L'opérateur || en JavaScript retourne l'opérande de gauche si elle est vraie (c'est-à-dire si sa conversion en booléen est égale à **true**) *sans évaluer l'expression de droite*, et sinon, évalue et retourne l'opérande de droite. Par conséquent, son résultat n'est généralement *pas un booléen* et || n'est pas strictement commutatif :

```
"chaîne" || 25    // retourne "chaîne"
25 || "chaîne"    // retourne 25
NaN || new Date() // retourne new Date()
new Date() || NaN // retourne new Date()
null || ""       // retourne ""
"" || null       // retourne null
close() || new Date() // ferme la fenêtre
new Date() || close() // retourne new Date() et ne ferme pas la fenêtre
```

Cela peut provoquer quelques comportements inattendus, par exemple :

```
if(("vrai" || "faux") == true)    // cette condition ne sera jamais vraie, alors que ce
serait le cas en C
```

Mais on peut aussi en tirer partie pour condenser un code tel que celui-ci :

```
function augmenter(n)
```

```
{
  if(!n) n=1
  ...
}
```

en le remplaçant par celui-ci :

```
function augmenter(n)
{
  n=n||1
  ...
}
```

D'où le nom "défaut" : si l'opérande de gauche a une valeur non-nulle, elle est utilisée, mais sinon l'opérande de droite est utilisée comme valeur *par défaut*. Cela permet à la fonction dans ce dernier exemple de prévoir une valeur par défaut pour son argument 'n', au cas où il ne serait pas fourni.

## NON (!)

L'opérateur de négation ! contredit logiquement l'expression qu'il précède. Il en convertit d'abord la valeur en booléen si elle était d'un autre type, et retourne le booléen contraire.

```
if (!true) alert("alors c'est faux!");
```

Si on l'utilise deux fois de suite (!! ) il permet simplement de convertir la valeur de son opérande en booléen :

```
"chaîne" != true
```

mais

```
!! "chaîne" === true
```

## Opérateurs de bits

Les opérateurs bit-à-bit et de décalage de bits commencent par convertir leurs opérandes en entiers de 32bits, après les avoir converties en **number** si c'était nécessaire. Les nombres NaN et Infinity donnent le nombre entier *zéro*.

Les opérateurs logiques &, |, ^ et ~ effectuent un test logique séparément sur chaque bit, de la manière illustrée par l'exemple suivant ou l'opérateur est un ET bit-à-bit :

```
var n = 26 & 12 ;
```

```

1  1  0  1  0  (26 en binaire)
0  1  1  0  0  (12 en binaire)
-----
1&0 1&1 0&1 1&0 0&0 <- opérations bit à bit
0  1  0  0  0  <- résultat (8 en binaire)
```

Les autres opérateurs de bits sont les opérateurs de décalage des bits (vers la droite ou la gauche).

## Et binaire (&)

Cet opérateur retourne pour deux opérands entiers (a et b) un nombre entier c dont chaque bit est calculé par la table de vérité suivante :

```
a & b = c
0  0  0
0  1  0
1  0  0
1  1  1
```

## Ou binaire (|)

Cet opérateur retourne pour deux opérands entiers (a et b) un nombre entier c dont chaque bit est calculé par la table de vérité suivante :

```
a | b = c
0  0  0
0  1  1
1  0  1
1  1  1
```

## Ou exclusif binaire (^)

Cet opérateur retourne pour deux opérands entiers (a et b) un nombre entier c dont chaque bit est calculé par la table de vérité suivante :

```
a ^ b = c
0  0  0
0  1  1
1  0  1
1  1  0
```

## Non binaire (~)

Cet opérateur retourne pour un opérande entier a un nombre entier c dont chaque bit est l'opposé du bit original :

```
~a = c
0  1
1  0
```

## Décalage de bits vers la gauche (<<)

Cet opérateur retourne l'opérande de gauche après avoir décalé ses bits vers la gauche du nombre de bits spécifié par

L'opérande de droite, en ajoutant de nouveaux bits de valeur nulle et en supprimant les bits qui dépassent de l'autre côté.

Exemple :

```
var c = 15 << 3;
```

```
0 0 0 0 1 1 1 1 (15 en binaire)
0 1 1 1 1 0 0 0 (décalé de 3 bits vers la gauche en ajoutant des zéros)
```

Attention : le bit le plus à gauche détermine le signe du nombre, et il est aussi affecté par le décalage.

## Décalage de bits vers la droite (>>)

Cet opérateur retourne l'opérande de gauche après avoir décalé ses bits vers la droite du nombre de bits spécifié par l'opérande de droite, en ajoutant de nouveaux bits de valeur nulle à droite et en supprimant des bits de poids faible (les plus à gauche dans la représentation binaire).

Exemple :

```
var c = 15 >> 3;
```

```
0 0 0 0 1 1 1 1 (15 en binaire)
0 0 0 0 0 0 0 1 (décalé de 3 bits vers la droite en supprimant les bits 1 1 1)
```

Le bit de signe (le plus à gauche) est préservé.

## Décalage de bits vers la droite y compris le bit de signe (>>>)

Cet opérateur retourne l'opérande de gauche après avoir décalé ses bits vers la droite du nombre de bits spécifié par l'opérande de droite, en ajoutant de nouveaux bits de valeur nulle à droite et en supprimant des bits de poids faible (les plus à gauche dans la représentation binaire).

Exemple :

```
var c = -15 >>> 3;
```

```
1 1 1 1 1 ... 1 1 0 0 0 1 (-15 en binaire)
0 0 0 1 1 ... 1 1 1 1 1 0 (décalé de 3 bits vers la droite en supprimant les bits 0 0 1)
```

Attention : le bit le plus à gauche détermine le signe du nombre, et il est aussi affecté par le décalage. Le résultat de l'exemple ci-dessus est 536870910.

## Opérateurs de comparaison

---

## Égalité (==, ===)

L'opérateur d'égalité == renvoie la valeur vrai si les deux termes à comparer sont équivalents.

```

{
  valeur="50";
  if (valeur == 50) alert("valeur vaut 50");
  // l'alerte sera affichée
}

```

L'opérateur === compare les valeurs et leur type. Le résultat est vrai (true) si les deux valeurs sont égales et de même type.

```

{
  valeur="50";
  if (valeur === 50) alert("valeur vaut 50 (nombre entier)");
  // l'alerte ne sera pas affichée (chaîne et entier sont deux types différents)
}

```

Si les deux opérandes sont des références (types object, fonction...), c'est les références qui sont comparées.

```

{
  if ([] == []) alert("les deux listes son identiques");
  // l'alerte ne sera pas affichée (chaque opérande est une référence différente même si
  // les listes [] et [] sont semblables)
}

```

Quelques valeurs spéciales :

- undefined est seulement équivalent, mais pas identique, à null.
- La chaîne vide "" est équivalente au nombre zéro, au booléen false, et à une liste vide [].
- Tous les nombres sont équivalents à leurs représentations sous forme de chaîne.
- Le nombre 1 est, de plus, équivalent au booléen true.
- Par contre le nombre NaN n'est pas équivalent à la chaîne "NaN".

Ces équivalences sont commutatives.

## Inégalité (!=, !==)

L'opérateur d'inégalité != renvoie la valeur vrai si les deux termes à comparer sont différents.

```

{
  valeur="50";
  if (valeur != 50) alert("valeur ne vaut pas 50");
  // l'alerte ne sera pas affichée
}

```

L'opérateur !== compare les valeurs et leur type. Le résultat est vrai (true) si les deux valeurs sont différentes ou de type différent.

```

{
  valeur="50";
  if (valeur !== 50) alert("valeur ne vaut pas 50 (nombre entier)");
  // l'alerte sera affichée (chaîne et entier sont deux types différents)
}

```

## Ordre (<, <=, >=, >)

Respectivement inférieur, inférieur ou égal, supérieur ou égal, supérieur, permet de comparer des valeurs.

```
if (10 > 5) alert("Normal, c'est le double");
```

À l'instar de l'opérateur +, les opérateurs de comparaison peuvent comparer numériquement ou alphabétiquement. Si les deux opérandes sont de types **number** ou **boolean**, **object** construit par `new Number`, ou les valeurs **null** ou **undefined** (les mêmes valeurs qui font que l'opérateur + les additionnerait), alors les valeurs sont converties en nombres et comparées en tant que telles. Sinon, les valeurs sont converties en chaînes et comparées alphabétiquement.

## Opérateurs d'incrément et de décrémentation (++ --)

Ces deux opérateurs proposent un moyen simple d'incrémenter ou de décrémentation une variable de type **number**. Si l'opérande n'est pas de type number, elle est convertie automatiquement. Si ce n'est pas une variable du tout, cela provoque une erreur comme pour l'affectation.

### Incrément et décrémentation

```
a = 0;
a++; // Réalise a = Number(a) + 1;
alert(a); // Affiche 1

a--;
alert(a); // Affiche 0
```

Note : Appliqué sur une valeur qui ne peut être convertie, comme une chaîne qui ne représente pas un nombre, l'opérateur abîme définitivement le contenu de la variable. Contrairement à l'opérateur +, l'opérateur ++ ne concatène jamais.

### Application éronnée sur une chaîne

```
s = "une chaîne";
s++; // Number("une chaîne") donne NaN, et NaN+1 reste NaN.
alert(s); // Affiche "NaN", et la chaîne de départ est perdue.
```

**Remarque :**

En langage C cette instruction peut être utilisée car une chaîne de caractères en C est un pointeur, donc fondamentalement un nombre entier. Ce n'est pas le cas en JavaScript.

Comme dans les autres langages, les opérateurs ++ et -- peuvent être placés avant ou après leur opérande. Cela change la valeur retournée par l'opération :

- Lorsque l'opérateur est placé à droite, l'opération retourne d'abord la valeur de départ de la variable *puis* l'incrémente (*Post-incrémentation*);
- Lorsque l'opérateur est à gauche, l'opération incrémente d'abord la variable *puis* retourne sa valeur modifiée (*Pré-incrémentation*).

#### Post-Incrémentation

```
a = 0;
alert( a++ ); // Affiche 0, mais a vaut 1 car incrémenté après retour de sa valeur
```

#### Pré-Incrémentation

```
a = 0;
alert( ++a ); // Affiche 1, a vaut bien 1 car incrémenté avant retour de sa valeur
```

## Opérateurs spéciaux

---

### Conditionnel ( ? : )

L'opérateur conditionnel est ternaire (il prend trois opérandes). Il est représenté par le signe ? qui sépare les deux premières, et : qui sépare les deux dernières. L'opération retourne l'une des deux opérandes qui entourent les deux points (:) après avoir déterminé si l'expression précédant le point d'interrogation (?) est vraie ou fausse.

```
(X ? A : B)
```

Si X est vrai (autrement dit si sa conversion en booléen donne **true**), l'opération évalue et retourne A sans évaluer B. Sinon, elle évalue et retourne B sans évaluer A.

On peut se représenter l'opérateur avec une structure if...else classique :

```
if(X)
  resultat = A
else
  resultat = B
```

### Instanciation (new)

L'opérateur **new** crée un nouvel objet qui hérite du prototype de la fonction qui le suit, puis appelle cette fonction dans laquelle le mot-clé *this* sera une référence à ce nouvel objet. La fonction qui sert d'opérande à new (le *constructeur*) peut



prendre des arguments. Le résultat de l'opération est l'objet ainsi créé, sauf si le constructeur retourne autre chose.

```
maintenant = new Date() // crée un objet avec la fonction prédéfinie Date
```

```
function Livre( nom, auteur ) // définition d'un constructeur
{
  this.nom = nom
  this.auteur = auteur || "personne" // définit une valeur par défaut si auteur est
  undefined, null, false, 0 ou ''
}

hp = new Livre( "Harry Potter et les Reliques de la Mort", "JK Rowling" )
```

Il est possible, dans l'opération d'instanciation, d'omettre les parenthèses après le nom de la fonction (comme si on ne l'appelait pas) lorsqu'on ne lui passe aucun argument :

```
maintenant = new Date // crée un objet avec la fonction prédéfinie Date
```

Bien sûr, une référence du constructeur est suffisante pour instancier un objet, il n'est pas nécessaire d'utiliser le nom de la fonction tel que défini dans la déclaration **function** :

```
o = Date
maintenant = new o() // crée un objet avec la fonction prédéfinie Date
```

Il est même possible de déclarer le constructeur dans la même ligne que new, et même d'utiliser une fonction anonyme :

```
singleton = new ( function () { this.prop = "valeur" } )()
```

Si l'opérande n'est pas de type "function", l'opération provoque une erreur.

## Dé-référencement (delete)

L'opérateur **delete** supprime la propriété qui le suit. Si la propriété contient une référence à un objet, seule la référence est supprimée : l'objet existe toujours et, s'il est référencé par d'autres variables, ne sera pas supprimé.

```
a = 3
b = ["une", "liste"]
c = { prop: "valeur" }

delete a // a n'existe plus, ni sa valeur
delete b // supprime la propriété b ; la liste elle-même sera libérée car elle n'a plus
aucune référence

delete c.prop
alert("prop" in c) // affiche false
```

Note : seul l'opérateur **delete** efface la propriété. Affecter **null** ou **undefined** à une variable déréférence l'objet qu'elle référençait s'il y en avait un, mais la variable existe toujours :

```

a = new Date()
c = { prop: "valeur" }

a = undefined
alert(a) // affiche "undefined"

delete a
alert(a) // erreur !

c.prop = undefined
alert("prop" in c) // affiche true

delete c.prop
alert("prop" in c) // affiche false

```

**delete** peut supprimer toute propriété existante (y compris prédéfinies telles que `Math`, `Object`...), mais pas les variables locales déclarées avec **var** ni les arguments d'une fonction, ni les variables **NaN** et **undefined**. Dans les cas où la variable ne peut pas être supprimée, l'opération retourne **false**. **delete** accepte aussi des propriétés inexistantes, et des valeurs anonymes (dans ce cas, il n'a aucun effet).

```

a = "valeur"
var b = "valeur"

delete a // true
delete b // false, et b n'est pas détruit
delete 17 // true, totalement inutile, et on peut toujours utiliser le nombre 17
delete Math.PI // false, c'est une propriété en lecture seule
delete Math // true, et c'est dommage parce qu'on ne peut plus accéder à l'objet Math !

```

## Propriété d'un objet (.)

L'opérateur `.` (point) permet simplement d'accéder à la propriété, nommée par l'opérande de droite, de l'objet référencé par l'opérande de gauche. L'opérande de droite doit être un identifiant ; l'opérande de gauche peut être n'importe quelle expression qui retourne un objet différent de **null**.

```

window . document . URL // affiche la propriété "URL" de l'objet référencé par la
propriété "document" de l'objet window
Math . sin( 1.5 ) // appelle la méthode référencée par la propriété "sin" de l'objet
Math
new Date() . getTime() // appelle la méthode référencée par la propriété "getTime" d'un
objet construit par Date

```

Si l'opérande de gauche est une valeur scalaire (string, boolean ou number), elle est convertie en un objet correspondant. Par exemple, une chaîne est convertie en un objet construit par `String` :

```

"chaîne".charAt(2) // affiche "a"

```

L'opérande de droite n'a pas besoin de désigner une propriété existante de l'objet. Il suffit (et il faut) que ce soit un identifiant valide.

L'opération provoque une erreur si l'opérande de gauche est **undefined** ou **null**, ou si l'opérande de droite n'est pas un identifiant.

Attention, le point est également le séparateur entre parties entière et décimale d'un nombre. S'il est immédiatement précédé par un nombre entier (sans espace), il ne fonctionnera pas comme opérateur.

## Type de variable (typeof)

L'opérateur **typeof** retourne une chaîne représentant le type de la valeur de l'expression à sa droite. Les possibilités sont "undefined", "boolean", "number", "string", "object" et "function".

```
if(typeof arg) != "number")
  alert("arg n'est pas un nombre !")
```

## Type d'objet (instanceof)

L'opérateur **instanceof** détermine si l'opérande de gauche "hérite" de l'objet à droite. C'est-à-dire, pour parler JavaScript, si le prototype de l'objet à droite de **instanceof** se trouve dans la chaîne de prototypes de l'opérande de gauche. **instanceof** retourne un booléen. L'opérande de droite n'a pas besoin d'être le constructeur de celle de gauche pour que **instanceof** retourne **true**.

```
new Date() instanceof Date // true
new Date() instanceof Object // true
new Date() instanceof Array // false
[2,7,8] instanceof Array // true
```

**instanceof** retourne toujours **false** lorsque l'opérande de gauche est une valeur scalaire (types string, boolean, number), undefined ou null. Une erreur se produit si c'est l'opérande de droite qui est dans ce cas.

## Appartenance à une liste (in)

L'opérateur **in** détermine si l'expression à sa gauche est le nom d'une propriété de l'objet à sa droite. L'opérande de droite doit obligatoirement être de type "object" et différente de **null**, sinon l'opération provoque une erreur. L'opérande de gauche peut être n'importe quelle valeur car elle sera automatiquement convertie en chaîne.

Exemple :

```
var liste = [1, 5, "bleu", new Date()]
var perso = {nom : "Potter", prenom : "Harry"}

alert(2 in liste) // affiche true car liste[2] existe
alert("bleu" in liste) // affiche false car "bleu" est la valeur de liste[2] mais pas
une clé de la liste
alert("length" in liste) // affiche true car liste est un Array qui possède une
propriété length

alert("nom" in perso) // affiche true, l'objet perso possède bien une propriété "nom"
alert("Harry" in perso) // affiche false, "Harry" n'est pas une propriété mais la valeur
```

```
d'une propriété  
alert("toString" in perso) // affiche true, l'objet perso hérite la méthode toString du  
prototype de Object
```

On voit que **in** va chercher dans le prototype des objets s'il ne trouve pas la propriété dans l'objet lui-même. Pour savoir si une propriété est directement possédée par un objet, il y a la méthode `hasOwnProperty(name)` :

```
alert(perso.hasOwnProperty("nom")) // true  
alert(perso.hasOwnProperty("toString")) // false
```

Dans le contexte de la structure de contrôle **for... in**, l'opérateur **in** joue un rôle un peu différent.

## Juxtaposition (,)

L'opérateur "virgule" permet simplement de juxtaposer plusieurs expressions, et retourne la valeur de la dernière (celle de droite).

```
alert( (x=3, 1) )
```

Ce code va afficher le nombre 1, mais l'affectation `x=3` aura bien lieu d'abord.

C'est très utilisé dans les boucles **for**, où on doit parfois initialiser, tester ou incrémenter plusieurs variables différentes en une seule expression.

Le signe "virgule" fonctionne différemment dans certains contextes (notamment quand on énumère les éléments d'une liste ou d'un objet, ou les arguments d'une fonction, ce n'est plus qu'un simple séparateur).

```
alert( x=3, 1 )
```

affichera 3, parce que 3 et 1 seront considérés comme deux arguments de la méthode `alert` mais elle n'affiche que le premier.

# Structures de contrôle

Javascript dispose de structures de contrôle comparables à ce qu'on rencontre en langage C. La plupart des règles et habitudes acquises dans ce langage sont immédiatement transposables en javascript.

Avant de passer en revue les structures de contrôle, il est utile de lire le paragraphe [Établissement d'une expression logique](#) qui précise certains aspects de la méthode.

## Établissement d'une expression logique

---

L'usage de **if**, **while** et **do** met en œuvre des expressions logiques dont le résultat testé (vrai) conditionnera l'exécution de l'instruction ou du bloc d'instructions associé.

Ces expressions logiques sont inscrites entre parenthèses.

Du plus simple au plus complexe, on trouve :

- `(5 == 5)` est bizarre mais vrai
- `(true)` La constante true est toujours vraie
- `(false)` La constante false est toujours fausse
- `(var_bool)` Si `var_bool` est vrai, l'expression est vraie
- `(!var_bool)` Si `var_bool` est vrai, l'expression est fausse
- `(var_bool==false)` Si `var_bool` est faux, l'expression est vraie (et oui...)

Avec des variables numériques ou des variables chaînes, le principe est similaire :

- `(ma_var_num == 5)`
- `(ma_chaine == "foo")`

De même avec les opérateurs de comparaison :

- `(ma_var > 0)` `ma_var` non nulle
- `(ma_var <= 5)` inférieure ou égale à 5
- `(ma_chaine != "bar")` le contenu de `ma_chaine` différent de "bar"

etc..

À l'étape suivante, on peut combiner des expressions logiques avec des opérateurs logiques ET ( `&&` ) et OU ( `||` ).

- `((ma_var==5)&&(ma_chaine=="bar"))` Réaliser les deux conditions
- `((ma_var==5)||(ma_chaine=="bar"))` Réaliser l'une des conditions
- `((ma_var==5)&&(ma_chaine=="bar"))!(var_bool)`

... Et ainsi de suite... en faisant très attention aux parenthèses et à l'ordre de priorité des opérateurs, et à conserver une certaine lisibilité au code.

## Le piège

---

L'auteur de cet article ne vient pas du C , mais du pascal. Il commet donc encore aujourd'hui l'erreur de confondre l'opérateur d'affectation avec l'opérateur d'égalité logique. Ceci a pour conséquence d'introduire un bug à retardement dû à l'affectation de la variable de test.

**if (ma\_var = 0) Exécution du code**

L'expression `(ma_var = 0)` est fausse. L'expression `(ma_var = 5)` est vraie. Mais dans les deux cas, l'affectation a lieu, écrasant le contenu précédent de la variable.

Donc cette erreur, si elle n'est pas comprise, plonge le programmeur dans des abîmes de questions et de doutes, avec peu de chances de comprendre pourquoi. Bien entendu, la bonne façon est d'utiliser l'opérateur d'égalité logique (`==`) à la place de l'opérateur d'affectation (`=`). La valeur de l'opération d'affectation peut cependant être utilisée volontairement. Voir [la section sur l'opérateur =](#).

## Branchement conditionnel

---

### if else

La structure de contrôle **if** permet de subordonner l'exécution d'une ligne ou d'un bloc d'instructions à l'évaluation (vrai) d'une [expression logique](#).

La syntaxe est :

#### Exemple if

```
if (condition_vrai) // Exécution d'une ligne d'instructions
    alert("La condition est vraie");
...
if (condition_vrai) // Exécution d'un bloc d'instructions
{
    alert("La condition est vraie");
    alert("... je le confirme");
}
```

Le mot réservé **else** permet d'exécuter une ligne ou un bloc d'instructions alternativement au résultat de l'évaluation de l'expression logique.

#### Exemple if else

```
if (condition)
    alert("La condition est vraie");
else
    alert("la condition est fausse");
...
if (condition)
{
    alert("La condition est vraie");
    alert("... je le confirme");
}
else
{
    alert("la condition est fausse");
}
```

```
    alert("... je le confirme");
}
```

## ? :

Cet opérateur est remarquable par sa concision et peut se substituer à la structure **if... else**. Il est expliqué en détails [sur la page précédente](#).

Parfois, il permet d'économiser beaucoup de code. En théorie, presque toute structure if... else peut être remplacée par cet opérateur, du moment qu'elle ne contient pas de déclaration. En pratique, il vaut mieux le réserver à des cas simples.

## switch

Le mot réservé **switch** permet en conjonction avec **case** de mettre en place un sélecteur de cas d'une grande souplesse.

Cette structure remplace avantageusement une structure équivalente construite à partir de if else et if else imbriqués.

Le mécanisme de test ne fait pas appel à une expression logique, mais à une comparaison d'une variable de type scalaire avec des valeurs du même type.

Contrairement au langage C, qui nécessite que les valeurs de comparaison soient des constantes littérales, JavaScript, interprété, autorise l'usage de variables.

La structure **switch case** ne pourrait pas fonctionner correctement sans **break**. En effet, et cela est déconcertant au début, quand une condition case est vérifiée, l'interpréteur n'effectue plus de test et exécute tout ce qu'il trouve jusqu'à la fin en passant par dessus les **case** rencontrés.

Enfin, le mot réservé **default** couvre les cas différents de ceux traités par les **case**.

### switch exemple 1

```
switch (ma_var)
{
var egal_deux = 2
  case 1 :
    alert("Ma variable vaut 1");
    break;
  case egal_deux :
    alert("Ma variable vaut 2");
    break;
  default : alert("Ma variable vaut autre chose que 1 ou 2");
}
```

On remarque les **break** systématiques dans ce cas.

D'autre part, on illustre la possibilité de fournir des variables à **case**, ce qui n'est pas possible en langage C. Deuxième exemple : Nous allons regrouper plusieurs cas, et déclencher plusieurs exécutions d'instructions pour certaines valeurs.

### switch exemple 2

```
switch (ma_var)
```

```
{
  case 0:
    alert("Vraiment nulle, cette variable"); // Elle vaut zero
    break;
  case 1:
  case 3: alert("Ma variable vaut 1 ou 3"); // Et on continue
  case 5:
    alert("Ma variable est impaire et comprise entre 1 et 5");
    break;
  case 2:
  case 4:
  case 6:
    alert("Ma variable est paire et comprise entre 2 et 6");
    break;
  case 7:
    alert("Ma variable est égale à 7");
    break;
  default: alert("Ma variable est négative ou supérieure à 7")
}
```

On remarquera l'utilisation des **break** pour regrouper des cas entre eux...

## Contrôle d'itération (boucles)

JavaScript implémente les mêmes structures de contrôle d'itération que le langage c, à savoir les boucles **for**, **while** et **do**.

Avant de les examiner, nous allons regarder l'usage de **continue** et **break** appliqué aux boucles.

### Utilisation de continue

Cette instruction permet à volonté de sauter des tours. L'exemple suivant saute le passage à zero d'une itération comprise entre -2 et 2.

#### Exemple de continue

```
for (var i=-2; i <= 2; i++)
{
  if (i==0)
    continue;
  alert(i); // Affiche -2, puis -1, 1 et 2... mais pas le zero.
}
```

Cette fonctionnalité de **continue** est aussi applicable aux boucles **while** et **do**.

**continue** peut être suivi du nom d'une étiquette placée juste avant une boucle. Dans ce cas, l'exécution continue au niveau de cette boucle, pas de celle qui contient directement l'instruction **continue**.

### Utilisation de break



Les boucles **for**, **while** et **do** autorisent l'usage de **break** pour sortir de l'itération.

Chacune de ces boucles possède une condition d'arrêt, mais parfois il peut être souhaitable de sortir de la boucle pour d'autres raisons (en cas d'erreur, si on a trouvé ce qu'on cherchait avant la fin d'une recherche, si l'utilisateur a décidé d'annuler une longue opération...). Par défaut, **break** termine immédiatement la boucle dont il fait partie et le script continue après cette boucle.

De même que **continue**, **break** peut être suivi du nom d'une étiquette. Dans ce cas, l'exécution continue après la structure désignée par l'étiquette, pas après la boucle qui contient directement l'instruction **break**.

### Attention !

Quand on utilise la boucle `.each()`, elle exécute une fonction, donc il faut remplacer "break" par "return false".



Exemple pour sélectionner une option d'un menu déroulant :

```
$(document).ready(function(){
    $("#liste_deroulante > option").each(function() {
        if ($(this).val() == "valeur à sélectionner") {
            $(this).attr('selected', 'selected');
            return false;
        }
    });
});
```

## Étiquettes

Une étiquette est un identifiant suivi de ':' (deux points). On peut la placer avant une instruction ou une structure de contrôle.

Il n'y a **pas** d'instruction `goto` en JavaScript. Les étiquettes servent exclusivement à affiner l'utilisation de **break** et **continue** (voir plus haut).

```
str = "Liste des objets :\n"
annuler = false

parcoursListe : for(i in liste)
{
    if(!liste.hasOwnProperty(i))
        continue // passe à l'itération suivante dans cette boucle
    str+="\n• L'élément «"+i+"» de la liste contient :\n"
    if(annuler)
    {
        str+="\nopération annulée."
        break // reprend l'exécution immédiatement après cette boucle
    }
    parcoursProps : for(j in liste[i])
    {
        if(!liste[i].hasOwnProperty(j))
            continue // passe à l'itération suivante dans cette boucle
    }
}
```

```
        if(annuler)
        {
            str+="\nopération annulée."
            break parcoursListe // reprend l'exécution immédiatement après la boucle
extérieure
        }
        str+="- "+j+" = "+liste[i][j]+"\n"
    }
}
```

## for

### Description

La structure for permet d'effectuer plusieurs fois une ligne d'instructions ou un bloc d'instructions.

Les modalités d'exécution de l'itération sont indiquées entre les parenthèses précédant le mot réservé for. L'instruction ou le bloc à exécuter se situent après.

Syntaxe:

```
for (modalités) action;
ou
for (modalités){action1; action2;}
```

Par modalités, nous regroupons en fait trois choses distinctes qui sont:

- l'initialisation
- La condition pour exécuter la boucle
- Les changements à effectuer à chaque tour (généralement une incrémentation ou décrémentation).

Ces trois instructions sont séparées par des points-virgule et placées entre parenthèses après le mot réservé **for**.

Généralement, ces trois instructions s'appliquent à une variable chargée de contrôler l'itération, et qu'on nomme avec la lettre **i**.

### Boucle croissante

L'exemple le plus simple est le suivant :

#### Exemple for

```
for (i = 0; i < 5; i++)
{
    alert(i); // Affiche 0, puis 1, puis 2, puis 3, puis 4
}
```

Dans cet exemple, nous avons initialisé la variable **i** à 0, défini la condition pour exécuter la boucle (répéter tant que **i** est strictement inférieur à 5) et défini le changement à effectuer à chaque tour (incrémenter **i**).

Une fois assimilé le fonctionnement, on imagine aisément toutes les possibilités.

D'abord, on peut initialiser *i* avec la valeur de son choix. Commencer avec 1, ou un nombre négatif.

Ensuite, on est libre de l'expression de la condition : strictement inférieur à 5, inférieur ou égal à 5 ( $\leq$ ) à condition devienne fausse à un moment donné, pour sortir.

## Boucle à paliers

On n'est pas tenu exclusivement d'incrémenter *i*. On peut modifier la valeur par pas de 2.

### Autre exemple for

```
for (i = 4; i <= 10; i += 2)
    alert(i + "est un nombre pair compris entre 4 et 10"); // affiche 4, puis 6, 8 et 10
```

## Boucle décroissante

De la même manière, une boucle peut être décroissante :

### Exemple for, boucle décroissante

```
for (i = 5; i >= 0; i--)
    alert(i); // affiche 5, puis 4, 3, 2, 1 et 0
```

## État en sortie de boucle

En sortie de boucle, *i* contient la valeur résultante des modifications. Pour le dernier exemple, c'est -1.

## Boucle de parcours

Grâce à l'opérateur `in` il est possible d'utiliser une forme spéciale de la boucle `for` pour parcourir un tableau ou une table associative (clé -> valeur) par ses indices/clés :

### Exemple for, parcours d'un tableau

```
var tableau = [ "une", "boucle for", "spéciale" ];
for (var i in tableau)
    alert(tableau[i]); // affiche successivement tous les éléments du tableau
    // i valant successivement : 0, 1, 2
```

### Exemple for, parcours d'une table associative

```
var table = { "yes": "oui" , "no": "non" , "maybe": "peut-être" };
for (var i in table)
    alert(table[i]); // affiche successivement "oui", "non", "peut-être"
                    // i valant successivement : "yes", "no", "maybe"
```

## Éviter les pièges :

### Boucle infinie

Il est possible, par inattention, de programmer une boucle infinie. Cela se produit quand la condition de boucle reste vraie malgré la modification de la variable.

```
for (i=0; i >= 0; i++)
```

L'incréméntation de *i* ne changera pas le fait que la variable est supérieure ou égale à zéro...

### Variable modifiée

La structure `for` s'appuie sur une variable pour piloter le déroulement de la boucle. Seulement, elle ne rend pas la variable comme elle l'a reçue. Elle la modifie complètement. Pour éviter de modifier une variable par erreur, il est judicieux d'utiliser le mot réservé `var` pour déclarer une variable locale à la boucle :

```
for (var i=0; ....
```

## while

La structure `while` conditionne l'exécution répétée d'une instruction ou d'un bloc d'instructions au test d'une expression logique.

```
while (condition)
    action;

ou

while (condition)
{
    action1;
    action2;
}
```

Pour qu'il soit possible de sortir de la boucle, il faut que les instructions modifient à terme la condition pour la rendre fausse.

### Exemple while

```
var i = 0; // Initialiser i
```

```
while (i<5) // Tant que i est strictement inférieur à cinq
  i++;      // ... l'incrémenter.
```

En langage c, la boucle while peut-être volontairement infinie : while(true) toujours vrai, mais en JavaScript, le navigateur ne l'acceptera probablement pas.

## do

La structure do permet d'exécuter une instruction ou un bloc d'instructions et de répéter cette action tant qu'une expression logique surveillée par while est vraie.

```
do
  action
while (condition)
```

### Exemple do

```
var i = 0;
do
{
  i++
}
while(i < 5);
```

# Fonctions utilisateur

## Déclaration et identification

---

Dans tous les cas (sauf deux), la fonction nécessite un Identificateur établi avec les mêmes règles que pour les variables.

Pour déclarer une fonction, on emploie la syntaxe suivante:

Déclaration d'une fonction

```
function ma_fonction()
```

C'est à dire, utilisation du mot réservé>> **function** suivi de l'identificateur et de parenthèses. Ces dernières serviront à définir les paramètres de la fonction.

## Fonction sans paramètres

---

La plupart des fonctions servent à effectuer des opérations sur des variables données en argument, mais ce n'est pas une obligation.

Fonction sans paramètres

```
function init()
{
  if (!navigator.cookieEnabled)...
}
```

## Transmission de paramètres

---

### Transmission classique

Les paramètres sont transmis à la fonction entre les parenthèses de la déclaration. Comme javascript est un langage à typage dynamique, les paramètres se résument à de simples identificateurs séparés par des virgules.

Transmission de paramètres (mode classique)

```
function display_message(s1,s2)
{
  var result = s1 + " " + s2;
  alert(result);
}
```

## Autre mode de paramétrage

JavaScript offre beaucoup de souplesse dans l'envoi d'arguments à une fonction.

le langage ne vous oblige pas à rédiger des fonctions au nombre d'arguments préétablis.

Un mécanisme simple permet à tout moment de compter et d'accéder aux paramètres.

JavaScript autorise l'appel d'une fonction en omettant les paramètres. Ceci ne déclenche pas d'erreur, mais c'est une incivilité, source de bugs.

La fonction, elle, peut à tout moment se rendre compte de la présence, absence, nombre et nature des paramètres, et ceci à travers un tableau intégré à la fonction et nommé "**arguments**".

### Argument non-renseigné

```
function foo(param1)
{
  if (arguments.length==0)
    alert("fonction foo : paramètre manquant");
}

// Appel de la fonction sans paramètre :
foo(); // Affiche "paramètre manquant";
```

L'exemple suivant montre comment tirer partie du tableau **arguments** pour traiter des données dont on ne connaît pas le nombre au départ

### Propriété arguments (Array)

```
function concat_châînes()
{
  result = "";
  for(i = 0; i < arguments.length; i++) // Pour chaque argument
  {
    result += arguments[i] + " "; // Ajouter le contenu et une espace à
result
  }
  return result; // renvoyer le résultat
}
alert(concat_châînes("Concaténation","de","châînes")); // Affiche "Concaténation de
châînes "
```

## Valeur renvoyée

Avec le mot réservé **return**, la fonction est capable de renvoyer une valeur au programme appelant. Cette valeur est contenue dans une donnée en tout point comparable à une variable typée.

## return vide

Le mot réservé `return` peut s'employer sans valeur de retour, uniquement dans le but de "sortir" de la fonction. Il est possible d'utiliser plusieurs fois `return` dans une fonction, l'exécution sera interrompue au premier `return` rencontré.

### return multiples

```
function foo(p)
{
  var resultat = "";
  if (condition_1)
    return;           // Sortie de secours
  else
    resultat = traitement(p);
  return resultat;   // Sortie normale
}
```

Note : Bien que parfaitement valide, cette pratique est généralement déconseillée en algorithmie. Une fonction doit s'efforcer d'avoir une entrée et une sortie.

## Corps de la fonction

---

### Variables

La fonction peut avoir besoin de créer des variables pour mener à bien ses calculs. Dans ce cas, il est indispensable de garder en tête qu'une variable non-précédée du mot réservé **var** se révèle globale, et donc source de conflits. Une bonne pratique consiste à déclarer ses variables en début de fonction, même si rien ne vous y oblige.

### Variabes locales

```
function foo()
{
  var v1;
  var v2;
}
```

### Sous-fonction

Une fonction peut contenir des sous-fonctions qui lui sont propre et l'aide à accomplir sa tâche.

### Sous-fonction

```
function foo(n)
{
  ;
}
```



```
function bar(p)
{
    return p*p*p;
}

for (var i = 0; i < n; i++)
    bar(i);
}
```

## Appel de la fonction

---

Nous pouvons admettre quatre modèles de fonction : Avec ou sans paramètres, avec ou sans valeur de retour.

### Sans valeur de retour

Sans valeur de retour, la fonction est un verbe. Il suffit de la nommer avec ses paramètres, et elle s'exécute :

Fonction sans valeur de retour

```
<body onload="init()">
```

### Avec valeur de retour

Si l'on a prévu que la fonction renvoie une valeur, on peut recueillir celle-ci dans une variable, ou l'utiliser directement, ou l'ignorer si bon nous semble. Le résultat renvoyé par la fonction a toutes les caractéristiques d'une variable (type, valeur, utilisabilité avec des opérateurs...

Avec valeur de retour

```
var le_cube_de_trois = cube(3); // Valeur de retour recueillie dans une variable
alert("Cube de 3 : " + cube(3)); // Affiche "Cube de 3 : 27" (concaténation)
cube(3); // Appel sourd de la fonction, on ignore le résultat
```

### Sans paramètre

Les parenthèses demeurent nécessaire à l'identification complète de la fonction. Elles restent vides. (Les espaces sont possibles, ainsi que les commentaires slash-étoile)

Entre parenthèses

```
n = foo();
```

```
in = foo ( );
in = foo(/* J'occupe l'espace... */);
```

## Avec paramètre

Les paramètres envoyés à une fonction peuvent être sous forme de constante, de référence à une variable ou d'expression littérale.

### Types d'arguments

```
result = foo("Chaîne littérale", ma_var, true); // Littéral, variable et constante
booléenne
```

## Que deviennent les variables envoyées

Les variables envoyées en paramètre d'une fonction subissent ou ne subissent pas de modifications selon leur nature:

### Primitives

Si la variable est d'un type primitif comme string, bool et number, le mécanisme d'appel de la fonction envoie en fait des copies de la variable. Les modifications opérées par la fonction n'ont pas de répercussion sur la variable originale, sous réserve, bien entendu qu'il n'y ait pas de collision d'identification avec une variable globale.

### Variables préservées

```
function foo(str)
{
  str = "c'est pas si important";
}
s = "Trés important";
foo(s);
alert(s); // Affiche "Trés important" : pas d'altération
```

### Instance d'un objet

Il en est tout autrement avec les instances d'un objet transmis en paramètre. Dans ce cas, le mécanisme d'appel de la fonction envoie un pointeur sur l'objet, et la fonction a toute latitude pour opérer des modifications sur les propriétés ou données de l'objet.

### Objet modifié

```
function modif_array(tab)
{
    tab[0] = "c'est pas si important";
}
t = Array();
t[0] = "Très important";
modif_array(t);
alert(t[0]); // Affiche "c'est pas si important" : modification du contenu de l'objet
```

## Fonction sans identificateur

---

Une fonction déclarée sans identificateur est une expression de type "function" qui peut être utilisée comme argument d'une fonction, ou affectée à une variable. L'invocation d'une telle fonction se fait en utilisant le paramètre ou la variable.

### Fonction sans identificateur

```
var superieur = function(a,b){
    return a>b;
};
if (superieur(5,4)) alert("5 > 4");
```

### Fonction sans identificateur

```
function affiche(objet, f_tostring)
{
    alert( f_tostring(objet) );
}
var personne = { "nom":"clinton", "prenom":"bill" };
affiche(personne, function(o){
    return o.prenom + " " + o.nom;
});
```

# Évènement

Les **événements JavaScript** permettent d'intercepter les changements d'états de l'environnement provoqués par le document HTML, les scripts ou l'interaction du client.

## L'objet Event

---

Lorsqu'un événement survient, un objet Event permettant de le décrire est créé. Il se propage alors dans l'arbre DOM selon trois phases déterminées par la cible (l'objet depuis lequel l'événement est intercepté) :

- Capture : l'événement se propage de la racine du document (incluse) à la cible (exclue).
- Cible : l'événement atteint la cible.
- Bouillonnement : l'événement se propage dans le sens inverse : de la cible (exclue) à la racine du document (incluse).

Cet objet a été défini par le W3C, mais malheureusement Internet Explorer en a sa propre définition ce qui oblige le développeur à tenir compte du navigateur.

Certaines propriétés de cet objet concernent tous les types d'événements et d'autres, tels que le bouton de la souris, sont spécifiques à un ou plusieurs événements. Seuls les premières nous intéressent dans cette partie, nous verrons les autres dans la description des événements.

### type

Identique sous Internet Explorer.

Renvoie le type d'événement (onkeydown, onload...).

### target

**Équivalent Internet Explorer : srcElement.**

Permet de récupérer l'élément depuis lequel l'événement a été envoyé. Il ne s'agit pas forcément de celui auquel on associe la fonction, mais de l'élément qui a récupéré le focus ou qui le récupère au moment de l'action. Par exemple, lors du clique de la souris sur un bouton, c'est ce bouton qui est renvoyé. Ou bien lorsque l'on appuie sur une touche du clavier, c'est l'objet qui a le focus qui est renvoyé.

### currentTarget

Non supporté par Internet Explorer.

Permet de récupérer l'objet auquel l'événement a été rattaché. Équivaut à utiliser la référence this qui fonctionne dans certain cas sous Internet Explorer.

### stopPropagation

**Équivalent Internet Explorer : l'attribut cancelBubble.**

Cette méthode arrête la propagation de l'événement dans l'arbre DOM après le nœud sur lequel il se trouve. Il faut faire attention au fait qu'il s'agisse d'une méthode dans le W3C mais d'un attribut sous Internet Explorer.

## preventDefault

Équivalent Internet Explorer : l'attribut **event.returnValue = false**.

Empêche l'action normalement prévue de se dérouler. Par exemple, lors de l'appui sur la touche tabulation dans un champ texte, cela annulera le changement de focus et permettra l'insertion d'une indentation. Il est préférable d'utiliser l'expression `"return false;"` (sauf pour Internet Explorer) qui est mieux supportée.

## Gestionnaires d'événements DOM-0

---

Chaque événement peut être capté par les objets HTML concernés en leurs associant une fonction ou une commande JavaScript. Nous verrons plus loin comment fonctionnent les écouteurs (gestionnaires d'événements DOM-2). Ici nous utiliserons le type de gestionnaire d'événement DOM-0, plus simple et plus fiable : les événements sont des méthodes (le nom de l'événement avec le préfixe "on") qu'il suffit de définir. Ces méthodes peuvent prendre en paramètre un objet Event qui permettra de contrôler l'événement. Cependant, ce paramètre n'est pas toujours nécessaire.

Il existe deux façons de définir les événements :

1. Directement dans la balise de l'objet concerné :

L'appui sur le bouton "bt" envoie un message avec son id :

```
<FORM>
  <INPUT type="button" id="bt" onclick="alert('Vous avez cliqué sur '+this.id+'.');">
</FORM>
```

La commande peut aussi être une fonction que vous avez vous même définie ultérieurement.

2. En associant une fonction via JavaScript :

Le même effet est obtenu différemment :

```
<FORM>
  <INPUT type="button" id="bt">
</FORM>
```

La fonction javascript est définie après :

```
document.getElementById("bt").onclick = function(event)
{
  alert("Vous avez cliqué sur "+this.id+".");
}
```

Le paramètre Event n'est pas accessible avec Internet Explorer, il faut donc passer par la variable globale window.event. Autrement, il suffit de récupérer l'instance de l'Event en paramètre de la fonction.

```
//Capture la touche de clavier enfoncée
document.onkeydown = function(event)
{
  //Internet Explorer ne prend pas d'objet Event en paramètre, il faut donc aller le
chercher dans l'objet window
  if (typeof event == "undefined" ) event = window.event;
}
```

## Écouteurs d'événements

Il est possible d'intercepter le flux d'événements dans l'arbre DOM avec des écouteur d'événements. Pour ajouter un écouteur à un objet HTML il suffit d'utiliser la méthode addEventListener. Cet écouteur peut-être supprimé avec removeEventListener. Il s'agit du type de gestionnaire d'événement DOM-2.

### addEventListener

Équivalent Internet Explorer : attachEvent;

Cette méthode crée un écouteur pour un objet HTML. Il prend trois paramètres :

- type : le type d'événement. Le suffixe "on" n'est requis que pour Internet Explorer.
- EventListener : la fonction appelée lors de l'événement.
- useCapture : **true** pour la phase de capture et **false** pour celles de la cible et du bouillonnement. Ce paramètre n'est pas disponible sous Internet Explorer qui ne traite que la cible et le bouillonnement.

Le mot clé this n'est hélas pas reconnu dans cette méthode, c'est pourquoi il est préférable d'utiliser le gestionnaire d'événement DOM-0.

Lors de l'appuie sur le bouton "my\_button", on affiche le type d'évènement la cible courante et on efface l'écouteur de sorte à ce que cette action ne s'effectue qu'une seule fois.

```
function clickMe(event)
{
    //this.id renvoie "undefined" : this n'est pas reconnu ici
    alert("Type : "+event.type+"\nCible courante : "+this.id);
    if(navigator.appName == "Microsoft Internet Explorer")
    {
        //Ne fonctionne pas : this n'est pas reconnu ici
        this.detachEvent("onclick", clickMe);
    }
    else
    {
        this.removeListener("click", clickMe, false);
    }
}

if(navigator.appName == "Microsoft Internet Explorer")
{
    document.getElementById("my_button").attachEvent("onclick", clickMe);
}
else
{
    document.getElementById("my_button").addEventListener("click", clickMe, false);
}
```

## removeEventListener

Équivalent Internet Explorer : `detachEvent`;

Pour détruire un écouteur, il faut utiliser cette méthode avec les mêmes paramètres que `addEventListener` (ou `detachEvent`).

## Événements

---

### onabort

En cas d'interruption de chargement d'une image.

Objets concernés : Image.

### onblur

Lorsque l'utilisateur quitte l'objet et que celui-ci perd le focus.

**Objets concernés** : Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea et window.

### onchange

Lorsque l'utilisateur quitte l'objet après l'avoir modifié et que celui-ci perd le focus.

**Objets concernés** : FileUpload, Select, Submit, Text et TextArea.

## onclick

Lors d'un clique de souris sur l'objet.

**Objets concernés** : Button, document, Checkbox, Link, Radio, Reset, Select et Submit.

## ondblclick

Lors d'un double clique de souris sur l'objet.

**Objets concernés** : Button, document, Checkbox, Link, Radio, Reset, Select et Submit.

## onerror

Survient lors d'une erreur de chargement.

**Objets concernés** : Image et window.

## onfocus

Lorsque l'objet est sélectionné et prend le focus.

**Objets concernés** : Button, Checkbox, FileUpload, Layer, Password, Radio, Reset, Select, Submit, Text, TextArea et window.

## onkeydown

Lorsqu'on appuie sur une touche. Pour tous les évènements clavier, Event a pour attribut keyCode (Internet Explorer) ou which (Netscape) et contient le code de la touche enfoncée.

Dans cet exemple un message envoie le code de la touche appuyée :

```
document.onkeydown = function(event)
{
    //On vérifie le navigateur
    if(navigator.appName === "Microsoft Internet Explorer")
    {
        //On envoie un message avec la touche appuyée pour Internet Explorer
        alert(event.keyCode);
    }
    else
    {
        //On envoie un message avec la touche appuyée pour Netscape
        alert(event.which);
    }
}
```



**Objets concernés** : document, Image, Input (type=file, password ou text), Link et TextArea.

## onkeypress

Lorsqu'on maintient une touche appuyée. Pour tous les événements clavier, Event à pour attribut keyCode (Internet Explorer) ou which (Netscape) et contient le code de la touche enfoncée.

**Objets concernés** : document, Image, Link et TextArea.

## onkeyup

Lorsqu'on relâche sur une touche. Pour tous les événements clavier, Event à pour attribut keyCode (Internet Explorer) ou which (Netscape) et contient le code de la touche enfoncée.

**Objets concernés** : document, Image, Link et TextArea.

## onload

Lors du chargement de la page HTML, d'une frame ou d'une image.

**Objets concernés** : Image, Layer et window.

## onunload

En quittant le fichier. L'objet Event ne subit pas la phase de bouillonnement avec onunload.

**Objet concerné** : window.

## onmousedown

En maintenant une touche de la souris appuyée.

Les attributs de Event pour la souris sont les suivants :

- pageX (Netscape) ou x (Internet Explorer) : position horizontale de la souris. Sous Netscape, le défilement du document n'est pas pris en compte.
- pageY (Netscape) ou y (Internet Explorer) : position horizontale de la souris. Sous Netscape, le défilement du document n'est pas pris en compte.
- which (Netscape) ou button (Internet Explorer) : bouton de la souris enfoncé, relâché ou maintenu. Avec which, le clique gauche renvoie 1, le milieu renvoie 2 et le droit renvoie 3. Avec button (sous Internet Explorer), le clique gauche renvoie 0, le milieu renvoie 4 et le droit renvoie 2.

Lorsque l'on clique avec la souris un message apparaît avec la position de la souris et le bouton cliqué :

```
document.onmousedown = function(event)
{

```

```
//Netscape
if(navigator.appName != "Microsoft Internet Explorer")
{
    alert(event.pageX+" - "+event.which);
}
//Internet Explorer
else
{
    alert(window.event.x+" - "+window.event.button);
}
}
```

**Objets concernés** : Button, document et Link.

## **onmouseup**

En relâchant une touche de la souris. Pour les attributs de Event voir [onmousedown](#).

**Objets concernés** : Button, document et Link.

## **onmousemove**

En bougeant la souris. Pour les attributs de Event voir [onmousedown](#).

**Objets concernés** : window et document

## **onmouseout**

En quittant l'élément avec la souris.

**Objets concernés** : Area, Layer et Link. Pour les attributs de Event voir [onmousedown](#).

## **onmouseover**

En passant sur l'élément avec la souris.

**Objets concernés** : Area, Layer et Link. Pour les attributs de Event voir [onmousedown](#).

## **onselect**

En sélectionnant du texte.

**Objets concernés** : text et Textarea.

## **onreset**

Lors de l'initialisation du formulaire.

**Objet concerné** : form.

## **onsubmit**

En envoyant le formulaire

**Objets concernés** : form

## **onresize**

Lors du redimensionnement du fichier.

**Objet concerné** : window.

## **onmove**

Lors du déplacement du fichier.

**Objet concerné** : window.

## **dragdrop**

Lors d'un glisser-déposer vers la fenêtre.

**Objet concerné** : window.

## **javascript**

Lorsqu'un lien est activé.





**Objets concernés** : Link

# Objets prédéfinis

JavaScript dispose d'une bibliothèque d'objets étoffée en regard d'un nombre de fonctions classiques réduit au minimum.

Ils possèdent des propriétés et méthodes statiques (directement utilisables sans créer d'instance), et la plupart sont en même temps des classes d'objets. C'est à dire qu'il est possible de créer des instances, lesquelles bénéficient de propriétés et méthodes qui leur sont propres.

- Les valeurs littérales de type chaîne, nombre, booléen, etc instancient implicitement un objet `String`, `Number`, `Boolean`...
- Certains objets comme `Date` se construisent explicitement avec `new`.
- Souvent, le constructeur permet de créer une instance même en oubliant le mot-clé `new`.
- Des objets comme `Math` sont purement statiques et n'ont aucune instance.
- Les objets spécifiques au DOM, `navigator` ou `window` par exemple, sont instanciés par le navigateur automatiquement ou bien indirectement par l'appel de méthodes du DOM, mais jamais directement avec `new`. D'ailleurs aucune classe correspondante n'est disponible pour en créer de nouvelles instances avec l'opérateur `new`.

1. [window](#) 
2. [document](#) 
3. [navigator](#) 
4. [Array \(tableaux\)](#) 
5. [Boolean](#) 
6. [Date](#) 
7. [Error](#) 
8. [Function](#) 
9. [Image](#) 
10. [Math](#) 
11. [Number](#) 
12. [Object](#) 
13. [RegExp](#) 
14. [String](#) 

# Références/Objets/window

## Propriétés

---

- **closed**  
fenêtre fermée
- **defaultStatus**  
affichage normal dans la barre d'état
- **innerHeight**  
hauteur du domaine d'affichage
- **innerWidth**  
largeur du domaine d'affichage
- **locationbar**  
barre d'adresse
- **menubar**  
barre de menus
- **name**  
nom de fenêtre
- **outerHeight**  
hauteur de la fenêtre globale
- **outerWidth**  
largeur de la fenêtre globale
- **pageXOffset**  
position de départ de la fenêtre à partir de la gauche
- **pageYOffset**  
position de départ de la fenêtre à partir du haut
- **personalbar**  
barre pour les adresses favorites
- **scrollbars**  
barres de défilement
- **statusbar**  
barre d'état
- **status**  
Contenu de la barre d'état
- **toolbar**  
barre d'outils

## Méthodes

---

- **alert()**  
boite de dialogue avec infos
- **back()** :  
page précédente
- **blur()**  
quitter la fenêtre
- **captureEvents()**  
surveiller les événements
- **clearInterval()**  
interrompre la suite d'instructions sans fin
- **clearTimeout()**  
interrompre le compte à rebours
- **close()**  
fermer la fenêtre
- **confirm()**  
boite de dialogue pour confirmer
- **disableExternalCapture()**  
empêcher une surveillance extérieure
- **enableExternalCapture()**  
permettre une surveillance extérieure
- **find()**  
chercher du texte
- **focus()**  
activer la fenêtre
- **forward()**  
page suivante
- **handleEvent()**  
renvoyer l'événement
- **home()**  
appeler la page d'accueil
- **moveBy()**  
se mouvoir avec des mentions relatives
- **moveTo()**  
se mouvoir avec des mentions absolues
- **open()**  
ouvrir une nouvelle fenêtre

- **print()**  
imprimer
- **prompt()**  
fenêtre de dialogue pour la saisie de valeur
- **releaseEvents()**  
fermer un événement
- **resizeBy()**  
modifier la taille avec des mentions relatives
- **resizeTo()**  
modifier la taille avec des mentions absolues
- **routeEvent()**  
parcourir la hiérarchie des gestionnaires d'événement
- **scrollBy()**  
défiler un certain nombre de pixels
- **scrollTo()**  
défiler jusqu'à la position
- **setInterval()**  
établir une liste d'instructions planifiées
- **setTimeout()**  
entamer le compte à rebours
- **stop()**  
interrompre

# Références/Objets/document

## Propriétés

---

- **alinkColor**  
couleur des liens lorsqu'ils sont cliqués
- **bgColor**  
couleur d'arrière plan
- **charset**  
jeu de caractères utilisés
- **cookie**  
chaîne de caractères pouvant être sauvegardée chez l'utilisateur
- **defaultCharset**  
jeu de caractères normal
- **fgColor**  
couleur pour le texte
- **lastModified**  
dernière modification du document
- **linkColor**  
couleur pour les liens
- **referrer**  
pages déjà visitées
- **title**  
titre du fichier
- **URL**  
adresse URL du fichier
- **vlinkColor**  
couleur pour les liens à des cibles visitées:

## Méthodes

---

- **captureEvents()** :  
surveiller les événements
- **close()** :  
fermer
- **createAttribute()** :  
créer un nœud d'attributs



- **createElement()** :  
créer un nœud d'éléments
- **createTextNode()** :  
créer un nœud de texte
- **getElementById()** :  
Accès à l'élément HTML par l'attribut Id
- **getElementsByName()** :  
Accès à l'élément HTML par l'attribut name
- **getElementsByTagName()** :  
Accès à l'élément HTML par liste d'éléments
- **getSelection()** :  
texte sélectionné
- **handleEvent()** :  
traiter les événements
- **open()** :  
ouvrir le document
- **releaseEvents()** :  
fermer des événements
- **routeEvent()** :  
parcourir la hiérarchie des gestionnaires d'événement
- **write()** :  
écrire dans la fenêtre du document
- **writeln()** :  
écrire ligne par ligne

# Références/Objets/navigator

## Propriétés

---

- **appCodeName**  
surnom du navigateur
- **appName**  
nom officiel du navigateur
- **appVersion**  
version du navigateur
- **cookieEnabled**  
Cookies permis
- **language**  
langue du navigateur
- **platform**  
plate-forme sur laquelle tourne le navigateur
- **userAgent**  
identification HTTP du navigateur

## Méthodes

---

- **javaEnabled()**  
vérifier la disponibilité Java

# Références/Objets/Array

## Caractéristiques

---

- Les tableaux de type *Array* sont dynamiques. Il n'est pas nécessaire de les dimensionner à l'avance, et leurs taille peut croître ou décroître en fonction du nombre d'éléments contenus. La propriété **length** contient le plus grand indice entier positif + 1.
- Il n'est pas obligatoire que les éléments soient contigus.
- L'objet Array combine les caractéristiques d'un tableau **classique** (adressage des éléments par l'indice 0,1...n), et les caractéristiques d'un tableau **associatif** (adressage des éléments avec un identificateur entre guillemets). L'interpréteur est très souple, il est possible de panacher les deux modes. Mais dans ce cas, la propriété length ne comptabilisera pas les cellules déclarées en mode associatif.
- Souplesse de l'interpréteur, encore, il est possible de mentionner des indices négatifs! Fort heureusement, ils n'influencent pas la valeur de length, qui gère les indices à partir de zéro.
- Un même tableau peut contenir plusieurs types de données, comme des chaînes, nombres, objets, ou ... tableaux, ce qui permet de construire de véritable structures de données.

## Propriétés

---

Une seule propriété : **length**.

Propriété	Lecture/Ecriture	type
length	Lecture seule	entier

## Méthodes

---

Voici une description succincte des méthodes classées par fonctionnalités. En cliquant sur le nom de la méthode, vous accédez à une description plus détaillée avec exemples.

### Modification du contenu

Propriété	Description	Exemple
pop()	Supprime et retourne le dernier élément	mon_tableau.pop()
push()	Ajoute un ou plusieurs éléments à la fin	mon_tableau.push("nouveau","deuxième nouveau")
shift()	Supprime le premier élément	mon_tableau.shift()
unshift()	Ajoute des éléments au début	mon_tableau.unshift("nouveau1", "nouveau2")
splice()	Insère des éléments	mon_tableau.splice(ou,2,"nouveau1","nouveau2")
reverse()	Réorganise le tableau de la fin vers le début	mon_tableau.reverse()
concat()	Concaténer plusieurs tableaux	mon_tableau.concat(T2,t3)

## Obtenir des données

Propriété	Description	Exemple
<code>join()</code>	Extrait les données dans une chaîne avec le délimiteur donné en paramètre	<code>s = mon_tableau.join(" ")</code>
<code>slice()</code>	Renvoie un tableau contenant 'nombre' cellules à partir de 'debut'	<code>n_tab = mon_tableau.slice(debut, nombre)</code>

## Trier

Propriété	Description	Exemple
<code>sort()</code>	Trier le tableau	<code>mon_tableau.sort()</code>

## Méthodes héritées

Propriété	Description	Exemple
<code>toString()</code>	Renvoyer le contenu de l'objet sous forme de chaîne	<code>s = mon_tableau.toString()</code>
<code>toLocaleString()</code>	Renvoyer le contenu de l'objet sous forme de chaîne	<code>s = mon_tableau.toLocaleString()</code>

## Manipulation

---

### Instanciation

L'instanciation consiste à invoquer le constructeur de l'objet à l'aide du mot réservé `new`.

#### Instanciation d'un objet Array

```
mon_tableau = new Array(); // Simple et direct
mon_tableau = new Array(10); // Imposition de dix cases vides, (de 0 à 9) length vaut 10
// Note : rien n'empêche immédiatement après de faire mon_tableau[11] (dynamisme)
mon_tableau = new Array("Élément 1", "Élément 2"); // Remplissage à la déclaration.
```

### Adressage d'un élément

#### Adressage d'un élément

```
t = new Array(null, null, null, "Contenu de trois"); // Création avec 4 éléments
t["nom"] = "contenu de 'nom'";
alert(t[0]); // Affiche 'null'
alert(t[3]); // Affiche "Contenu de trois";
alert(t["nom"]); // Affiche "contenu de 'nom'"
alert(t[4]); // Affiche "undefined"
</pre>
}}
=== Lecture de la propriété length ===
```

```

{{Cadre code|Lecture de la propriété length|
<source lang="javascript">
t = new Array(null,null,null,"Contenu de trois"); // Création avec 4 éléments

var nb_elements = t.length; // Lecture de length dans une variable

alert("Le tableau a " + t.length + " éléments"); // Affiche "le tableau a 4 éléments"

for(i = 0; i < t.length; i++)
    alert("contenu de : " + i + t[i]);
// Affiche "null", puis "null", "null" et "Contenu de trois"

```

## Utilisation d'une méthode

Tri d'un tableau

```

t = new Array("Premier","Deuxieme","Troisieme");
t.sort(); // Invocation de la méthode sort()
alert(t); // Affiche "Deuxieme,Premier,Troisieme", soit le contenu par ordre
alphabétique

```

## Références

---

### Modifier le contenu du tableau

#### concat()

Méthode: concat ( array1 [ , array2 , array3 ] )

Rôle : Le ou les tableaux envoyés en paramètres sont ajoutés à la fin du tableau.

Arguments : Le ou les tableaux à ajouter, séparés par des virgules

Valeur renvoyée : Un tableau contenant les éléments du tableau de départ plus les tableaux ajoutés.

#### pop()

Méthode: pop()

Rôle : Le dernier élément du tableau est supprimé du tableau et renvoyé.

Arguments : aucun

Valeur renvoyée : l'élément supprimé

### **push()**

Méthode: push ( var1 [ , var2 , var3 ] )

Rôle : ajoute un ou plusieurs éléments à la fin.

Arguments : Le ou les éléments à ajouter, séparés par des virgules.

Valeur renvoyée : Jusqu'à JavaScript 1.2, renvoie le dernier élément ajouté. Avec les versions plus récentes, renvoie la taille du tableau.

### **shift()**

Méthode: shift()

Rôle : Le premier élément du tableau est supprimé et renvoyé.

Arguments : aucun

Valeur renvoyée : l'élément supprimé

### **unshift()**

Méthode: unshift( var1 [ , var2 , var3 ] )

Rôle : Un élément ou plusieurs éléments sont insérés au début du tableau.

Arguments : aucun

Valeur renvoyée : La nouvelle longueur

### **splice()**

Méthode: splice (indice , nombre [ , var1 , var2 , var3 ] )

Rôle : Remplace (éventuellement ajoute) un certain nombre d'éléments à l'endroit spécifié.

Arguments :	Indice (base zero) Nombre d'éléments
Valeur renvoyée :	Au choix: rien, un, ou plusieurs éléments aucun ou le tableau des éléments coupés

### reverse()

Méthode: reverse()	
Rôle :	Inverse l'ordre des éléments.
Arguments :	aucun
Valeur renvoyée :	aucun

## Obtenir des données

### join()

Méthode: join( [ sep_string ] )	
Rôle :	Renvoie une chaîne contenant les éléments du tableau séparés par le ou les caractères transmis en argument.
Arguments :	le (ou les) caractères de séparation
Valeur renvoyée :	La chaîne ainsi construite

### slice()

Méthode: slice( indice, nombre )	
Rôle :	Renvoie un tableau contenant un nombre d'éléments extraits à partir de l'indice.
Arguments :	Indice  Nombre
Valeur renvoyée :	Le tableau résultant

## Trier le tableau

### sort()

Prévue en standard pour trier des chaînes alphanumériques, la méthode **sort** est capable de bien plus grâce au mécanisme de fonction de comparaison externe.

En effet, en programmant judicieusement cette fonction, vous pouvez trier des nombres, des dates, ou même des objets abstraits.

Méthode: `sort( [ compareur() ] )`

Rôle : Effectue le tri du tableau. Selon le mode désiré, il peut être nécessaire de définir une fonction de comparaison, et la transmettre en argument.

Arguments : Tri en mode caractère : Pas d'argument

Tri en mode numérique : Une fonction de comparaison

Valeur renvoyée : Aucun

1er exemple : fonction de tri ascendant définie à part, appelée ensuite par son nom dans la fonction `sort`.

tri ascendant

```
function pour_tri_ascendant(n1, n2)
{
    return (n1 - n2); // Retourne positif si n1 > n2, négatif ou nul si n1 < n2
}

t = new Array(1,3,2);
t.sort(pour_tri_ascendant);
alert(t.join(", ")); // Affiche "1,2,3"
```

2eme exemple : fonction de tri descendante.

tri descendant

```
function pour_tri_descendant(n1, n2)
{
    return (n2 - n1); // Retourne positif si n1 < n2, négatif ou nul si n1 > n2
}

t = new Array(1,3,2);
t.sort(pour_tri_descendant);
alert(t.join(", ")); // Affiche "3,2,1"
```

3eme exemple : Intégration d'un fonction anonyme directement définie dans le paramètre.

fonction intégrée

```
t = new Array(1,3,2);
```



```
t.sort( function(n1,n2) { return n1-n2; } );  
alert(t.join(", ")); // Affiche "1,2,3"
```

4eme exemple : tri d'une notion abstraite (petit moyen grand).

tri abstrait

```
function compare_taille(s1,s2)  
{  
  if (s1=="petit") return -1;  
  
  if (s1=="moyen")  
  {  
    if (s2=="petit") return 1;  
    else return -1;  
  }  
  
  if (s1=="grand") return 1;  
  
  return 0; // Egalité  
}  
  
t = new Array("petit", "grand", "moyen", "grand", "petit", "moyen");  
t.sort( compare_taille);  
alert(t.join(", ")); // // Affiche "petit,petit,moyen,moyen,grand,grand"
```

# Références/Objets/Date

## Méthodes

---

- **getDate()**  
renvoie le jour du mois
- **getDay()**  
renvoie le jour de la semaine
- **getFullYear()**  
renvoie l'année complète
- **getHours()**  
renvoie la partie heures de l'heure
- **getMilliseconds()**  
renvoie les millièmes de secondes
- **getMinutes()**  
renvoie la partie minutes de l'heure
- **getMonth()**  
renvoie le mois
- **getSeconds()**  
renvoie la partie secondes de l'heure
- **getTime()**  
renvoie l'heure
- **getTimezoneOffset()**  
renvoie le décalage horaire de l'heure locale
- **getUTCDate()**  
renvoie le jour du mois de l'heure UTC (temps universel coordonné)
- **getUTCDay()**  
renvoie le jour de la semaine de l'heure UTC
- **getUTCFullYear()**  
renvoie l'année complète de l'heure UTC
- **getUTCHours()**  
renvoie la partie heures de l'heure UTC
- **getUTCMilliseconds()**  
renvoie les millièmes de secondes de l'heure UTC
- **getUTCMinutes()**  
renvoie la partie minutes de l'heure UTC
- **getUTCMonth()**

renvoie le mois de l'heure UTC

- **getUTCSeconds()**  
renvoie la partie secondes de l'heure UTC
- **getYear()**  
renvoie l'année
- **parse()**  
renvoie le nombre de millièmes de secondes depuis le 1/1/1970
- **setDate()**  
change le jour du mois de l'objet
- **setFullYear()**  
change l'année complète de l'objet
- **setHours()**  
change la partie heures de l'heure de l'objet
- **setMilliseconds()**  
change la partie millièmes de seconde de l'heure de l'objet
- **setMinutes()**  
change la partie minutes de l'heure de l'objet
- **setMonth()**  
change la partie mois de la date de l'objet
- **setSeconds()**  
change la partie secondes de l'heure de l'objet
- **setTime()**  
change la date et l'heure de l'objet
- **setUTCDate()**  
change le jour du mois de l'heure UTC de l'objet
- **setUTCDay()**  
change le jour de la semaine de l'heure UTC de l'objet
- **setUTCFullYear()**  
change l'année complète de l'heure UTC de l'objet
- **setUTCHours()**  
change la partie heures de l'heure UTC de l'objet
- **setUTCMilliseconds()**  
change la partie millièmes de seconde de l'heure UTC de l'objet
- **setUTCMinutes()**  
change la partie minutes de l'heure UTC de l'objet
- **setUTCMonth()**  
change le mois de l'heure UTC de l'objet

- **setUTCSeconds()**  
change la partie secondes de l'heure UTC de l'objet
- **setYear()**  
change la date et l'heure de l'objet
- **toGMTString()**  
convertir la date et l'heure au format GMT
- **toLocaleString()**  
convertir la date et l'heure au format local
- **UTC()**  
renvoie le nombre de millièmes de secondes entre le 1/1/1970 et un moment donné

## Exemples

---

```
var now = new Date();
var jj  = now.getDate();
var mm  = now.getMonth() + 1;
var aaaa = now.getYear();
var H   = now.getHours();
var M   = now.getMinutes();
```

```
<!DOCTYPE html>
<html>
<body>
<script type="text/JavaScript">
var j,r,a,neuv_a,neuv_j,neuv_m,m,mr,jr;
j=prompt("donner les jours");
j=parseInt(j);
m=prompt("donner le mois");
m=parseInt(m);
a=prompt("donner l'année");
a=parseInt(a);
j_ajou=prompt("donner les jours à ajouter");
j_ajou=parseInt(j_ajou);
mr=j/30;
jr=j%30;
neuv_j=j+mr;
neuv_m=m+mr;
neuv_a=a;
if(neuv_j<=30)
{{ neuv_j=1;
neuv_m=neuv_m+1;}}
else
if(neuv_m<12)
{neuv_m=1;
neuv_a=neuv_a+1;}
}
alert("la nouvelle date est"+neuv_j+'/'+neuv_m+'/'+neuv_a);
```

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

# Références/Objets/Fonction

## Propriétés

---

**arguments**

tableau de noms d'arguments

**arity**

nombre d'arguments

**caller**

nom de la fonction qui appelle

## Exemples

---



Cette section est vide, pas assez détaillée ou incomplète.

# Références/Objets/Math

## Propriétés

---

- **E** :  
constante d'Euler
- **LN2**  
logarithme naturel de 2
- **LN10**  
logarithme naturel de 10
- **LOG2E**  
logarithme constant de 2
- **LOG10E**  
logarithme constant de 10
- **PI**  
constante PI
- **SQRT1\_2**  
constante pour racine carrée de 0,5
- **SQRT2**  
constante pour racine carrée de 2

## Méthodes

---

- **abs()**  
valeur positive
- **acos()**  
arc cosinus
- **asin()**  
arc sinus
- **atan()**  
arc tangente
- **ceil()**  
nombre entier supérieur le plus proche
- **cos()**  
cosinus
- **exp()**  
valeur exponentielle

- **floor()**  
nombre entier inférieur le plus proche
- **log()**  
utilisation du logarithme naturel
- **max()**  
le plus grand de deux chiffres
- **min()**  
le plus petit de deux chiffres
- **pow()**  
nombre puissance exposant
- **random()**  
0 ou 1 aléatoire
- **round()**  
arrondi commercial d'un nombre
- **sin()**  
sinus
- **sqrt()**  
racine carrée
- **tan()**  
tangente



# Références/Objets/Number

## Propriétés

---

- **MAX\_VALUE**  
plus grand nombre pouvant être sauvegardé
- **MIN\_VALUE**  
plus petit nombre pouvant être sauvegardé
- **NaN**  
nombre non valable
- **NEGATIVE\_INFINITY**  
nombre trop petit
- **POSITIVE\_INFINITY**  
nombre trop grand

## Méthodes

---

- **toExponential()**
- **toFixed()**
- **toPrecision()**
- **toString()**

# Références/Objets/String

## Encodage

---

A priori, le type `String` de JavaScript contient du texte dans un encodage donné. En particulier, cet encodage peut-être lié à UTF-16.

Par exemple, Ecma 262 indique en §4.3.16 qu'une *String value* qui est une séquence d'entiers 16 bits non signés. Chaque multipllet de 16-bits est une unité de code de texte UTF-16. ECMAScript ne contraint pas ces valeurs<sup>[1]</sup>.

## Propriétés

---

- **length**  
nombre de *caractères*, ou plus précisément, nombre d'unités de code de 16 bits.

## Méthodes

---

- **anchor()**  
créer une ancre de lien
- **big()**  
créer un grand texte  
???
- **blink()**  
créer un texte clignotant  
???
- **bold()**  
créer un texte en caractères gras  
???
- **charAt()**  
~~rechercher un signe à une position~~  
renvoie le *caractère* (en fait l'unité de code) situé dans la chaîne à l'offset indiqué, ou en son absence, la chaîne vide.
- **charCodeAt()**  
~~valeur de code Latin-1 à une position~~  
renvoie la valeur du *caractère* (en fait l'unité de code) situé dans la chaîne à l'offset indiqué, ou en son absence, le nombre NaN.
- **concat()**  
concaténer des chaînes de caractères
- **fixed()**  
créer un texte style télex

???

■ **fontcolor()**

créer une couleur de police

???

■ **fontsize()**

créer une taille de police

???

■ **fromCharCode()**

~~créer une chaîne de caractères en Latin 1~~

créer une chaîne à partir de la liste des unités de codes numériques indiquée

■ **indexOf()**

rechercher la position d'un caractère

Recherchrche une sous chaîne à parti d'un position, et renvoie l'offset où se trouve la sous-chaîne, et -1 lorsque la sous-chaîne n'est pas trouvée.

■ **italics()**

créer un texte en italique

???

■ **lastIndexOf()**

rechercher la dernière position d'un signe

■ **link()**

créer un lien

???

■ **match()**

appliquer une expression régulière

Renvoie vrai, faux ou nul en fonction de l'éventuelle correspondance.

■ **replace()**

appliquer une expression régulière et remplacer

■ **search()**

chercher avec une expression régulière

■ **slice()**

extraire une partie d'une chaîne de caractères

■ **small()**

créer un petit texte

???

■ **split()**

scinder une chaîne de caractères

■ **strike()**

créer un texte barré

???

- **sub()**  
créer un texte en indice
- **substr()**  
rechercher une sous-chaîne de caractères à partir d'une position
- **substring()**  
rechercher une sous-chaîne de caractères
- **sup()**  
créer un texte en exposant
- **toLowerCase()**  
tout écrire en minuscules  
Renvoie une chaîne ou chaque caractère BMP a été converti en minuscule. Les *surrogates* ne sont pas modifiés.
- **toUpperCase()**  
tout écrire en majuscules  
Renvoie une chaîne ou chaque caractère BMP a été converti en majuscule. Les *surrogates* ne sont pas modifiés.

## Références

---

1. [https://fr.wikibooks.org/wiki/%C3%80\\_la\\_d%C3%A9couverte\\_d'Unicode](https://fr.wikibooks.org/wiki/%C3%80_la_d%C3%A9couverte_d'Unicode)

# Références/Objets/Element

Les éléments HTML sont les nœuds d'un arbre. Ils peuvent donc en avoir des parents et des enfants<sup>[1]</sup>.

## Propriétés

---

Pour récupérer le texte contenu dans une balise HTML :

- `nodeValue`
- `innerHTML`
- `textContent`
- `innerText`

## Méthodes

---



Cette section est vide, pas assez détaillée ou incomplète.

## Références

---

1. [https://www.w3schools.com/jsref/dom\\_obj\\_all.asp](https://www.w3schools.com/jsref/dom_obj_all.asp)

# Références/Objets/RegExp

A partir de la version 6, prévue pour octobre 2015, d'ECMA-script, les regex unicode seront supportées.

## Syntaxe

---

### Expressions rationnelles courantes

Caractère	Type	Explication
.	Point	n'importe quel caractère
[...]	crochets	<u>classe de caractères</u> : tous les caractères énumérés dans la classe
[^...]	crochets et circonflexe	<u>classe complémentée</u> : tous les caractères sauf ceux énumérés
^	circonflexe	marque le début de la chaîne, la ligne...
\$	dollar	marque la fin d'une chaîne, ligne...
	barre verticale	alternative - ou reconnaît l'un ou l'autre
(...)	parenthèses	<u>groupe de capture</u> : utilisée pour limiter la portée d'un masque ou de l'alternative
*	astérisque	0, 1 ou plusieurs occurrences
+	le plus	1 ou plusieurs occurrences
?	interrogation	0 ou 1 occurrence

Classes de caractères POSIX<sup>[1]</sup>

Classe	Signification
<code>[[:alpha:]]</code>	n'importe quelle lettre
<code>[[:digit:]]</code>	n'importe quel chiffre
<code>[[:xdigit:]]</code>	caractères hexadécimaux
<code>[[:alnum:]]</code>	n'importe quelle lettre ou chiffre
<code>[[:space:]]</code>	n'importe quel espace blanc
<code>[[:punct:]]</code>	n'importe quel signe de ponctuation
<code>[[:lower:]]</code>	n'importe quelle lettre en minuscule
<code>[[:upper:]]</code>	n'importe quelle lettre capitale
<code>[[:blank:]]</code>	espace ou tabulation
<code>[[:graph:]]</code>	caractères affichables et imprimables
<code>[[:cntrl:]]</code>	caractères d'échappement
<code>[[:print:]]</code>	caractères imprimables exceptés ceux de contrôle

Expressions rationnelles Unicode<sup>[2]</sup>

Expression	Signification
<code>\A</code>	Début de chaîne
<code>\b</code>	Caractère de début ou fin de mot
<code>\d</code>	Chiffre
<code>\D</code>	Non chiffre
<code>\s</code>	Caractères espace
<code>\S</code>	Non caractères espace
<code>\w</code>	Lettre, chiffre ou underscore
<code>\W</code>	Caractère qui n'est pas lettre, chiffre ou underscore
<code>\X</code>	Caractère Unicode
<code>\z</code>	Fin de chaîne

## Méthodes

- **exec()**  
appliquer une expression régulière
- **test()**  
appliquer une expression régulière pour essayer

- **match()**  
appliquer une expression régulière pour sortir occurrence
- **replace()**  
appliquer une expression régulière pour remplacer

## Recherche

---

La fonction *RegExp* contient deux paramètres : la chaîne à traiter et facultativement, le type de traitement : "g" (global), "i" (ignorer la casse) ou "gi".

```
<SCRIPT language=javascript>
var chaine="Test regex Javascript pour Wikibooks francophone.";
var expression=new RegExp("Wikibooks","g");
if (chaine.match(expression))
    document.write("Le texte parle de Wikibooks");
else
    document.write("Le texte ne parle pas de Wikibooks");
</SCRIPT>
```

## Remplacement

---

```
<SCRIPT language=javascript>
// Remplace tous les espaces par des underscores
var chaine="Test regex Javascript pour Wikibooks francophone.";
var expression=new RegExp(" "), "g");
document.write("Chaîne d'origine : " + chaine + "<BR>");
document.write("Chaîne traitée : " + chaine.replace(expression,"_") + "<BR>");
</SCRIPT>
```

### Attention !

Depuis ES6, le regex est natif en JS.



## Références

---

- <http://www.toutjavascript.com/reference/reference.php?ref=replace&parent=15>



# Fonctions prédéfinies

- **decodeURI()**

décoder une URI codée

- **decodeURIComponent()**

décoder une URI codée - II

- **encodeURI()**

encoder une URI

- **encodeURIComponent()**

encoder une URI - II

- **eval()**

interpréter une expression

- **escape()**

transformer des signes ASCII en nombres

- **isFinite()**

vérifier le domaine numérique de valeurs

- **isNaN()**

vérifier si la valeur n'est pas un nombre

- **parseFloat()**

transformer en nombre avec décimales

- **parseInt()**

transformer en nombre entier

- **Number()**

transformer un objet en nombre

- **String()**

transformer le contenu d'un objet en une chaîne de caractères

- **unescape()**

transforme des nombres en caractères ASCII

[<< Objets prédéfinis](#)

# Opérateurs de bits

## Opérations binaires

---

Les opérations binaires correspondent à une manipulation bit à bit d'une valeur contenue dans un variable. Il est important de comprendre le procédé des opérations binaires avant de travailler avec ces dernières.

### Représentation binaire

Une valeur contenue dans une variable est représenté coté machine par une succession de "0" et de "1". Chaque bit est placé à une position qui définit son poids et permet de cumuler les valeurs.

Le principe est en fait plus simple qu'il n'y paraît. Prenons une donnée sur 4 bits:

0011

Ici, nous avons 4 colonnes qui représentent un bit chacun. Il faut lire les données de droite à gauche. La première colonne représente  $2^0$ , le second  $2^1$ , le troisième  $2^2$  et enfin la quatrième  $2^3$

Il suffit alors de cumuler les valeurs. Ici

```
1* 2^0 = 1
+ 1* 2^1 = 2
+ 0* 2^2 = 0
+ 0* 2^3 = 0
```

Soit: 3

La combinaison de 0 et 1 permet donc de représenter n'importe quel chiffre selon la taille en octet alloué à la variable. Sur 4 bit, on peut alors représenter au maximum le chiffre 1111 qui donne:

```
1* 2^0 = 1
+ 1* 2^1 = 2
+ 1* 2^2 = 4
+ 1* 2^3 = 8
```

soit: 15. En conséquence, en partant de 0, 4 bits représentent 16 valeurs possible allant de 0 à 15

### Opérations

Il est possible de manipuler des valeurs binaire en utilisant des opérateurs spécifiques. Ainsi, il est possible de faire:

- une opération "ou"
- une opération "et"
- une opération "ou exclusif"
- décalage de bits

Les 3 premières opérations se font "bit à bit" à place respectives.

- Le "et" vérifie si les deux bits valent "1". Dans ce cas, l'opération résulte d'un "1", sinon "0".
- Le "ou" vérifie si un des bit est égal à "1" ou les deux. Dans ce cas, l'opération résulte d'un "1", sinon "0".
- Le "ou" vérifie si un des bit est égal à "1" mais pas les deux. Dans ce cas, l'opération résulte d'un "1", sinon "0".

Ainsi, prenons un exemple: 9 et 12 (ne fait pas 9+12): 9 = 1001 12 = 1100

on regarde chaque colonne, et on opère avec un "et".

- colonne 1 => 1 et 0 = 0
- colonne 2 => 0 et 0 => 0
- colonne 3 => 0 et 1 => 0
- colonne 4 => 1 et 1 => 1

Résultat 1000 => 8

Le "ou" donne:

- colonne 1 => 1 ou 0 = 1
- colonne 2 => 0 ou 0 => 0
- colonne 3 => 0 ou 1 => 1
- colonne 4 => 1 ou 1 => 1

Résultat 1101 => 13

et enfin le ou exclusif:

- colonne 1 => 1 xou 0 = 1
- colonne 2 => 0 xou 0 => 0
- colonne 3 => 0 xou 1 => 1
- colonne 4 => 1 xou 1 => 0

Résultat 0101 => 5

## Utilité

On se sert d'opération binaire dans beaucoup de cas, notamment pour le cryptage, le chiffage, ou encore des opérations optimisées en mathématiques.

Il est à noter que répéter deux fois un "ou exclusif" revient à revenir à la valeur d'origine. Exemple:

Valeur 5 => 0101

Appliquons deux fois le ou exclusif avec 4 = 0100

```

0101 => valeur initiale 5
xou 0100 => ou exclusif avec 4
0001 => 1
xou 0100 => encore avec 4
0101 = valeur 5, retour à l'état initial

```

## Les opérateurs

---

Il existe plusieurs opérateurs qui permettent ces opérations.

- & - "ET" (and)
- | - "OU" (or)
- ^ - "OU Exclusif" (xor)

Ainsi en javascript:

```
var a = 5
var b = 4
alert(a&b) /*Affiche 4*/
alert(a|b) /*Affiche 5*/
alert(a^b) /*Affiche 1*/
```

## Manipulation sur chaîne de caractères

---

L'opération binaire ne se faisant que sur des entiers, il est nécessaire de connaître le code ASCII des caractères sur lesquels nous voulons opérer. Il suffit d'utiliser la méthode "charCodeAt()" qui retourne le code ASCII à la position donnée.

Ainsi

```
var a="a"
//on récupère le code ASCII à l'index "0" => premier caractère
alert(a.charCodeAt(0)) //affiche 97
```

Cela permet donc de faire une opération binaire sur une chaîne:

```
//hello et world ont la même taille, c'est important pour cet exemple.
var a="hello"
var b="world"
var buffer = []
//cela va faire h^w, e^o, l^r, l^l; o^d
for (var i=0; i<a.length; i++) {
    buffer.push ( a.charCodeAt(i)^b.charCodeAt(i) )
}
alert ( buffer.join('.') ) //31:10:30:0:11

var result = ""
//opération inverse, revient à "hello"
for (var i=0; i<a.length; i++) {
    result += String.fromCharCode(buffer[i] ^ b.charCodeAt(i))
}
alert(result) //hello
```

## Décalage de bits

---

# Programmation objet

## Programmation objet

---

Dans la programmation orientée objet, le programme est conçu pour que ses composants soient aussi modulaires que possible. En particulier, quand un nouveau type d'objet est créé, il devrait fonctionner sans problèmes lorsqu'il est placé dans un environnement différent ou nouveau projet de programmation, c'est à dire qu'il est indépendant. Le résultat est une réduction du temps passé dans la réécriture de code.

JavaScript utilise des objets pour représenter des types de données complexes. Ces objets sont de petites structures de données avec leurs propres champs et des fonctions d'accès ou de modification de ces données. Ces objets bénéficient d'une approche où les variables sont protégées de toute ingérence extérieure. Si les variables peuvent être modifiées directement par le code d'un programme *en dehors* de la fonction ou de l'objet, alors il ne peut plus être tributaire de donner un résultat précis.

Contrairement à d'autres langages de programmation, JavaScript n'a pas de niveaux de protection sur les membres d'un objet. JavaScript utilise un prototype de forme des objets, qui peuvent encore hériter de classes des parents, mais n'est pas une forme pure de langage orienté objet ; il s'agit un langage de programmation objet par prototype. Cependant, la plupart des modèles de conception peut encore s'appliquer au langage tant que l'on ne cherche pas à accéder directement à l'état interne de l'objet (par exemple, en utilisant les méthodes d'un objet).

Comme avec d'autres langages de programmation, en JavaScript, les références aux champs et fonctions d'un objet utilisent le point (.) entre l'objet et le nom du champ.

## Mot clé new

---

Le mot clé `new` crée un nouvel objet.

```
item = new Object();
```

## Object methods and fields

---

Dans JavaScript, les objets n'ont pas de forme fixe - ils peuvent se modifier en cours d'exécution pour créer un nouvel objet, ou pour créer des champs ou fonctions.

```
money = new Object();  
money.quarters = 10;
```

Comme on le voit plus bas, on peut créer des méthodes pour l'objet de la même manière.

## Function et prototype

---

En JavaScript, qui est un langage de programmation orientée prototype, chaque objet hérite des attributs et méthodes de son prototype.

Exemple :

```

function Animal(race, sex)
{
    this.race = race;
    this.sex = sex

    manger = function()
    {
        return "je mange";
    }
}

Animal.prototype.attaquer = function(){
    alert("J'attaque");
};

var anim = new Animal("labrador","male");

anim.age = 10;
anim.courir = function(vitesse){
    alert("Je cours à "+vitesse+" km/h");
};

// ->
anim.courir(50);
anim.attaquer();

```

## mot clé this

Le mot clé `this` est utilisé dans une méthode, et se réfère à l'objet qui est attaché à cette méthode.

```

money.addQuarters = function(amount) {
    this.quarters += amount;
}
money.addQuarters(10);

```

## paradigme de programmation classe/objet

```

/**
 * Modélisation d'une voiture
 *
 * @class Voiture
 */
var Voiture = (function () {
    //'use strict';

    Voiture.couleur = "verte";
    // methode de classe
    Voiture.construire = function (marque) {
        return new Voiture(marque);
    };
}());

```

```
// constructeur
function Voiture(marque) {
    this.marque = marque;
}

// méthodes d'instance
Voiture.prototype = {

    rouler: function (direction) {
        document.write("la voiture roule");
    },
    getMarque: function () {
        return this.marque;
    }
};

return Voiture;
})();

// ->
var saab = new Voiture('Saab');
var chrysler = Voiture.construire('Chrysler');
chrysler.rouler();
alert(saab.getMarque());
```

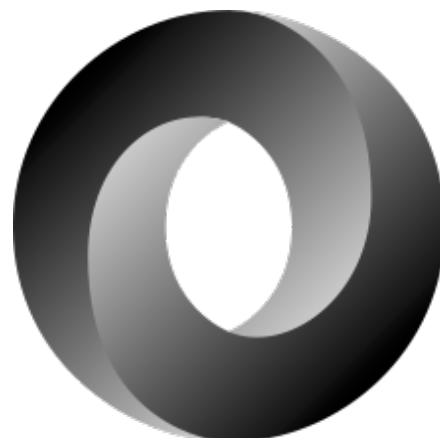


# Notation JSON

La notation objet du langage Javascript JSON (*JavaScript Object Notation* en anglais) permet de déclarer des objets en donnant la valeur des membres. Cette notation concerne également les tableaux qui sont considérés comme des objets.

Elle abrège la déclaration de données en permettant de donner toutes les valeurs d'un objet ou d'un tableau en une seule expression. Cette expression peut, par exemple, être passée en paramètre d'une fonction.

Contrairement sa principale alternative, le XML, cette notation ne comprend aucun en-tête et est toujours encodée en UTF8.



Logo JSON

## Tableau

Un tableau est en réalité une forme spéciale d'objet. Pour preuve, la fonction `typeof(...)` retourne le type "object" quand on lui passe un tableau.

La syntaxe abrégée pour déclarer le contenu d'un tableau utilise une série de valeurs entre crochets :

```
[ valeur, valeur... ]
```

Où *valeur* est une expression dont le type peut être quelconque (entier, chaîne de caractères, tableau, objet, ...).

Exemple :

```
var fruits = [ "pomme", "orange", "pamplemousse" ];
```

Équivalent sans la notation :

```
var fruits = new Array();  
fruits[0] = "pomme";  
fruits[1] = "orange";  
fruits[2] = "pamplemousse";
```

Variante :

```
var fruits = new Array();  
fruits.push( "pomme" );  
fruits.push( "orange" );  
fruits.push( "pamplemousse" );
```

Il est possible d'imbriquer les tableaux, c'est à dire déclarer un tableau de tableaux :

```
var parfums_glaces = [
```

```
[ "chocolat", "vanille" ],  
[ "fraise" ],  
[ "framboise", "cassis" ]  
];
```

Tous les éléments d'un tableau ne sont pas obligatoirement du même type :

```
var article = [ "Livre", 21.99 ]; // nom et prix
```

## Objet

Un objet Javascript est en réalité une table associant un nom (attribut ou clé) à une valeur.

Exemple :

```
var article = new Object();  
article.nom = "Livre";  
article.prix_unitaire = 21.99;  
article.quantite = 2;
```

Variante :

```
function Article(nom, prix_unitaire, quantite)  
{  
    this.nom = nom;  
    this.prix_unitaire = prix_unitaire;  
    this.quantite = quantite;  
    this.resume = function(){  
        return this.nom+" x "+this.quantite+" à "+this.prix_unitaire+  
            " l'unité coûte(nt) "+( this.quantite*this.prix_unitaire );  
    }  
    return this;  
}  
var article = new Article( "Livre", 21.99, 2);
```

Pour accéder au nom de l'article :

```
alert( "L'article est : " + article.nom ); // syntaxe objet  
alert( "L'article est : " + article["nom"] ); // syntaxe table associative
```

La notation abrégée utilise une série d'association nom-valeur entre accolades :

```
{ nom : valeur , nom : valeur , ... }
```

Exemple :

```
var article = {  
    "nom" : "Livre",  
    "prix_unitaire" : 21.99,  
};
```

```
"quantite" : 2,
"resume" : function(){
    return this.nom+ " x "+this.quantite+ " à "+this.prix_unitaire+
        " l'unité coûte(nt) "+( this.quantite*this.prix_unitaire );
}
};
```

Le nom peut se passer des guillemets s'il suit la syntaxe des identificateurs :

```
var article = {
    nom : "Livre",
    prix_unitaire : 21.99,
    quantite : 2,
    resume : function(){
        return this.nom+ " x "+this.quantite+ " à "+this.prix_unitaire+
            " l'unité coûte(nt) "+( this.quantite*this.prix_unitaire );
    }
};
```

## Imbrications

---

Il est possible de déclarer des tableaux d'objets, des objets contenant des tableaux ou d'autres objets, ...

Exemple complexe :

```
var commandes = [
    { "client": "Jean",
      "articles": [
        { "nom": "Livre", "quantite": 2, "prix_unitaire": 21.99 } ,
        { "nom": "Stylo", "quantite": 4, "prix_unitaire": 0.79 }
      ],
      "mode_paiement": "chèque"
    },
    { "client": "Pierre",
      "articles": [
        { "nom": "Livre", "quantite": 1, "prix_unitaire": 21.99 } ,
        { "nom": "Trombones", "quantite": 50, "prix_unitaire": 0.05 }
      ],
      "mode_paiement": "espèces"
    }
];
```

## Voir aussi

---

Convertisseurs JSON :

- <http://www.passiondataviz.fr/json-csv/#>
- <http://www.utilities-online.info/xmltojson/#.VKhxeXuj-So>

Bases de données JSON :

- [MongoDB](#)
- [CouchDB](#)



# Ajax

## Ajax : comment créer un sommaire

---

### Intérêt de l'utilisation d'Ajax

Lorsqu'on écrit un sommaire en PHP classique, à chaque fois qu'on clique sur un lien la totalité de la page est affichée. Sur un sommaire, cela crée un effet de clignotement indésirable et d'autant plus important que la page est lourde. De plus, comme il faut régénérer toute la page, la tendance est à surcharger le serveur avec des requêtes inutiles.

Avec la technologie AJAX, seule la partie qui est modifiée dans la page est rechargée. On diminue ainsi à la fois la charge du serveur, celle du réseau et l'effet de clignotement.

- Sans la technologie AJAX, on observe un effet de clignotement<sup>[3]</sup>.
- Avec Ajax, plus de clignotement et un site plus rapide<sup>[4]</sup>.

#### Remarque :

- ici on voit très peu la différence (quasiment pas d'ailleurs) car le site est super simple. Si le site était plus complexe notamment avec des accès à une base de donnée, la différence serait beaucoup plus nette !
- Regardez aussi la différence au niveau de l'utilisation des retours en arrière.

#### Un autre exemple :

- Sans la technologie AJAX, on observe un effet de clignotement<sup>[5]</sup>.
- Avec Ajax, plus de clignotement et un site plus rapide<sup>[6]</sup>.

## Les fichiers

### Le fichier index.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
  <head>
    <title>Sommaire en PHP !</title>
    <style type="text/css">
#sommaire
{
  position: absolute;
  background-color: cyan;
  left: 10px;
  width: 100px;
}

#page
{
  position: absolute;
```

```
background-color:#AAAAAA;
left : 200px;
width:500px;
height:500px;
}
</style>

<script type='text/JavaScript'>
var xhr = null;
function getXhr()
{
    if(window.XMLHttpRequest)xhr = new XMLHttpRequest();
else if(window.ActiveXObject)
    {
        try{
            xhr = new ActiveXObject("Msxml2.XMLHTTP");
        } catch (e)
        {
            xhr = new ActiveXObject("Microsoft.XMLHTTP");
        }
    }
else
    {
        alert("Votre navigateur ne supporte pas les objets XMLHttpRequest...");
        xhr = false;
    }
}

function ShowPage(page)
{
    getXhr();
    xhr.onreadystatechange = function()
        {
            if(xhr.readyState == 4 && xhr.status == 200)
                {
                    document.getElementById('page').innerHTML=xhr.responseText;
                }
        }
    xhr.open("GET", "ajax.php?page="+page, true);
    xhr.send(null);
}

</script>

</head>

<body onLoad="ShowPage(1)">

    <div id="sommaire">
        <h3>Sommaire</h3>
        <a href="#" onClick="ShowPage(1)">Page 1</a><br/>
        <a href="#" onClick="ShowPage(2)">Page 2</a><br/>
        <a href="#" onClick="ShowPage(3)">Page 3</a><br/>
        <a href="#" onClick="ShowPage(4)">Page 4</a><br/>
    </div>

    <div id="page">
    </div>

```

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

### Le fichier ajax.php

```
<?php
$page=$_GET['page'];
    if($page==1)require 'page1.html';
else if($page==2)require 'page2.html';
else if($page==3)require 'page3.html';
else require 'page4.html';
?>
```

### Le fichier page1.html

```
<h1>Page 1</h1>
bla bibib blan
```

### Le fichier page2.html

```
<h1>Page 2</h1>
bonjour
```

### Le fichier page3.html

```
<h1>Page 3</h1>
bli bli bli
```

### Le fichier page4.html

```
<form method="get" action="http://www.google.com/search"><fieldset style="border: 1px
solid black;"><legend style="font-family:verdana;font-weight:bold;font-size:1em;
color:orange;">Recherche Google</legend><TABLE><tr><td align="center"><div style="text-
align: center;">
<A HREF="http://www.google.fr">
<IMG SRC="http://www.google.com/logos/Logo_40wht.gif" border="0"
ALT="Google" align="middle"></A></div></td></tr>
<tr><td align="center"><div style="text-align: center;"><INPUT TYPE=text name=q size=20
value="">
<INPUT TYPE=hidden name=hl value=fr></div></td></tr>
<tr><td align="center" colspan="2"><div style="text-align: center;"><INPUT
style="border: 2px outset purple;color:white;background-color:purple;font-
weight:bold;font-family:verdana;" type=submit name=btnG VALUE="Recherche"></div>
</td></tr></TABLE>
```

```
|</FORM></fieldset>
```

## Références

---

1. <https://www.regular-expressions.info/posixbrackets.html>
2. <http://www.regular-expressions.info/unicode.html>
3. <http://xavier.merrheim.free.fr/PHP/sommaire1>
4. [http://xavier.merrheim.free.fr/PHP/sommaire\\_ajax1](http://xavier.merrheim.free.fr/PHP/sommaire_ajax1)
5. <http://xavier.merrheim.free.fr/PHP/sommaire2>
6. [http://xavier.merrheim.free.fr/PHP/sommaire\\_ajax2](http://xavier.merrheim.free.fr/PHP/sommaire_ajax2)



# Dojo

Le **cadriciel Dojo** est un framework open source en JavaScript.

Le présent manuel se fonde sur la documentation officielle de Dojo disponible en Wiki (<http://redesign.dojotoolkit.org/>).

Son but est le développement rapide d'applications en JavaScript exécutées côté client et communiquant avec le serveur avec une granularité inférieure à la page grâce à Ajax.

- Hello World
- Widgets
- Ressources

# Dojo/Hello World

Le présent livre a pour finalité de fournir un ensemble de connaissance de départ pour les nouveaux utilisateurs de Dojo. Il se base sur <http://dojo.jot.com/WikiHome/Tutorials/HelloWorld>.

## Notes

---

Il est important que la version 0.4.0 ou supérieure soit utilisée. Elle peut être trouvée à cette adresse :

<http://download.dojotoolkit.org/release-0.4.0/>

## Initialiser Dojo

---

Créer l'arborescence suivante :

```
- HelloWorldTutorial/
  |
  |-- js/
  |   |-- dojo/
```

télécharger la [dernière version de dojo](http://dojotoolkit.org/downloads/) (<http://dojotoolkit.org/downloads/>) et la décompresser dans le répertoire HelloWorld/js/dojo/

Vous devriez alors avoir la structure suivante : ('..' indique qu'il y a plus de répertoire)

```
- HelloWorldTutorial/
  |
  |-- js/
  |   |-- dojo/
  |       |-- build.txt
  |       |-- CHANGELOG
  |       |-- demos
  |           |
  |           |-- ..
  |       |-- dojo.js
  |       |-- dojo.js.uncompressed.js
  |       |-- iframe_history.html
  |       |-- LICENSE
  |       |-- README
  |       |-- src/
  |           |
  |           |-- ..
```

## Pour commencer

---

Nous allons maintenant créer une page html de base qui servira à appeler toute les fonctionnalités de Dojo que nous utiliserons par la suite.

```
<html>
<head>
  <title>Dojo: Hello World!</title>
  <!-- SECTION 1 -->
  <script type="text/javascript" src="js/dojo/dojo.js"></script>
</head>
<body>
</body>
</html>
```

## Créer un bouton gadget

---

Nous allons créer un bouton gadget (*widget* en anglais, c'est ce terme qui est utilisé par Dojo) avec le texte *Hello world*. Dans ce cas ci, trois option sont disponible (mouseOut, mouseOver, et mouseDown), qui enrichissent considérablement l'expérience de l'utilisateur !

La première chose à faire est de demander à Dojo de charger le module correspondant.

Dans l'en-tête (<head> jusqu'à </head>), placer la section correspondante :

```
<!-- SECTION 2 -->
<script type="text/javascript">
  // Charge le code de Dojo relatif aux fonctions de chargement des widgets
  dojo.require("dojo.widget.*");
  // Charge le code de Dojo relatif au bouton gadget
  dojo.require("dojo.widget.Button");
</script>
```

Le premier **dojo.require** instruit dojo d'inclure les fonctions **widget** (gadgets) (Attention cela ne charge pas tout les widgets!) ; c'est les lignes d'instructions du second dojo.require qui charge le bouton. Si vous oubliez la deuxième ligne, vous aurez un bouton en HTML.

Après avoir fait ces changements, insérer ce code suivant à l'intérieur des balises <body> et </body>

```
<button dojoType="Button" widgetId="helloButton">Salut tout le monde!</button>
```

L'élément clé à percevoir ici est le **dojoType**. Le type est ici un bouton, mais nous pourrions mettre un **input** à la place.

Le **widgetId** identifie le bouton. Il peut être remplacé par simplement **id** .

Pour mettre plusieurs bouton avec la même phrase, il faut copier cette ligne et changer l'id.

# Dojo/Widgets

## HTML valide W3C

---

Si vous voulez que votre HTML soit valide W3C (<http://validator.w3.org/>), il faut changer l'attribut **dojoType** qui n'est pas reconnu. Exemple :

```
<input dojoType="ComboBox">
```

devient

```
<input class="dojo-ComboBox">
```

où ComboBox est votre Widget. Vous pouvez donc ainsi faire de l'AJAX compatible aux normes du W3C.

# Dojo/Ressources

## Liens externes

---

- [Téléchargement \(http://dojotoolkit.org/docs/\)](http://dojotoolkit.org/docs/)
- [Blog \(http://blog.dojotoolkit.org/\)](http://blog.dojotoolkit.org/)
- [Documentation \(http://dojotoolkit.org/docs/\)](http://dojotoolkit.org/docs/)
  - [API \(http://dojotoolkit.org/docs/apis/\)](http://dojotoolkit.org/docs/apis/)
  - [Manuel \(http://manual.dojotoolkit.org/index.html\)](http://manual.dojotoolkit.org/index.html)
  - [Wiki \(http://dojo.jot.com/WikiHome\)](http://dojo.jot.com/WikiHome)
  - [FAQ \(http://dojo.jot.com/FAQ\)](http://dojo.jot.com/FAQ)
  - [Mailing List \(http://news.gmane.org/gmane.comp.web.dojo.user/\)](http://news.gmane.org/gmane.comp.web.dojo.user/)
- [Communauté \(http://dojotoolkit.org/community/\)](http://dojotoolkit.org/community/)

# React

## Syntaxe de base

---

React privilégie la programmation déclarative à la programmation impérative<sup>[1]</sup> pour programmer des applications monopage.

- `React.createClass` contient les fonctions.
- `ReactDOM` les invoque.

Prérequis : npm, Webpack, JSX, Babel.

## Routes

---

Pour gérer les URL, on utilise la classe *Route*<sup>[2]</sup>.

## Exemple

---

Affiche d'une base de données en tableau avec Griddle.



Cette section est vide, pas assez détaillée ou incomplète.

## Références

---

1. <https://facebook.github.io/react/>
2. <https://reacttraining.com/react-router/>

# Débogage

## Afficher des objets

---

Pour afficher des objets non scalaires (équivalent du `var_dump()` de PHP) :

```
alert(JSON.stringify(monObjet1));
```

## Débogage au sein d'un navigateur

---

### Firefox

- [Firefox](#) fournit une console d'erreur JavaScript (Menu "Outils" / "Console d'erreur")
- Des extensions spécifiques permettent de faciliter le débogage du JavaScript :
  - [Venkman JavaScript Debugger](#)<sup>[1]</sup>.
  - [Firebug](#) pour Firefox, qui était indispensable avant que le navigateur intègre DevTools.

Un mode d'exécution pas à pas est ainsi possible dans l'onglet Débugueur (CTRL + Maj + S). En rafraichissant la page, il permet d'afficher les contenus des variables, sans avoir à placer des `console.log()` dans le code.

### Chrome

- [Firebug](#) pour [Chrome](#).

### Internet Explorer

Sous [IE](#), dans les options avancées (Menu "Outils" / "Options Internet" / Onglet "Avancées"), décocher 2 options "désactiver le débogage de script". Si une erreur JavaScript est présente dans la page, une popup vous demandera si vous souhaitez ouvrir le débogueur de Microsoft (*Microsoft Script Editor*).

Ce navigateur vous dira ainsi, qu'il ne reconnaît pas la fonction `getElementsByClassName` disponible dans d'autres navigateurs.

F12 lance le débogage de la page.

## Exemples d'erreur

---

### **\$ is not defined**

jQuery n'est pas importé, ou alors pas dans le bloc "head".

### **addOnloadHook is not defined**

La fonction doit être chargée sans `addOnloadHook()`.

## missing ) after argument list

Peut provenir d'un littéral qui n'est pas entre guillemets.

## ReferenceError: invalid assignment left-hand side

Un attribut est redéfini avec le mauvais opérateur, par exemple :

```
// Remplacer "$('#champ1').val() = '1';" par :  
$('#champ1').attr('value', '1');  
  
// ou "$('#URL1').attr('href') = 'https://fr.wikibooks.org';" par :  
$('#URL1').attr('href', 'https://fr.wikibooks.org');
```

## uncaught exception: out of memory <inconnu>

Le quota a été atteint.



Cette section est vide, pas assez détaillée ou incomplète.

## Références

---

1. <http://www.mozilla.org/projects/venkman/>

## Voir aussi

---

- <http://www.misfu.com/static/Javascript/detect.html>
- [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Erreurs/JSON\\_bad\\_parse](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Erreurs/JSON_bad_parse)



# Références

## Liens externes

---

### Spécifications

- Standard ECMA 262 ECMAScript Language Specification  
<https://www.ecma-international.org/ecma-262/6.0/>

### Documentation

- <https://wiki.selfhtml.org/wiki/JavaScript>
- <https://github.com/getify/You-Dont-Know-JS>

### Outils

- [http://wiki.moxiecode.com/index.php/Main\\_Page](http://wiki.moxiecode.com/index.php/Main_Page) : TinyMCE WYSIWYG.
- <https://jsfiddle.net/>



Vous avez la permission de copier, distribuer et/ou modifier ce document selon les termes de la **licence de documentation libre GNU**, version 1.2 ou plus récente publiée par la Free Software Foundation ; sans sections inaltérables, sans texte de première page de couverture et sans texte de dernière page de couverture.

---

Récupérée de « [https://fr.wikibooks.org/w/index.php?title=Programmation\\_JavaScript/Version\\_imprimable&oldid=483813](https://fr.wikibooks.org/w/index.php?title=Programmation_JavaScript/Version_imprimable&oldid=483813) »

**La dernière modification de cette page a été faite le 14 juillet 2015 à 01:42.**

Les textes sont disponibles sous licence Creative Commons attribution partage à l'identique ; d'autres termes peuvent s'appliquer.

Voyez les termes d'utilisation pour plus de détails.