



Langages Web

JavaScript

Gestion de versions

Version	Date	Rédacteur	Description
1.0	10/05/2020	Lesly	Description détaillée du langage JavaScript
1.1	07/11/2021	Soupramanien	Application de la charte Baobab et de la traçabilité Corrections et mise à jour: ajout d'un slide sur la portée du bloc et l'ajout des urls de référence pour les fonctionnalités abordées



Bases et introduction à JavaScript

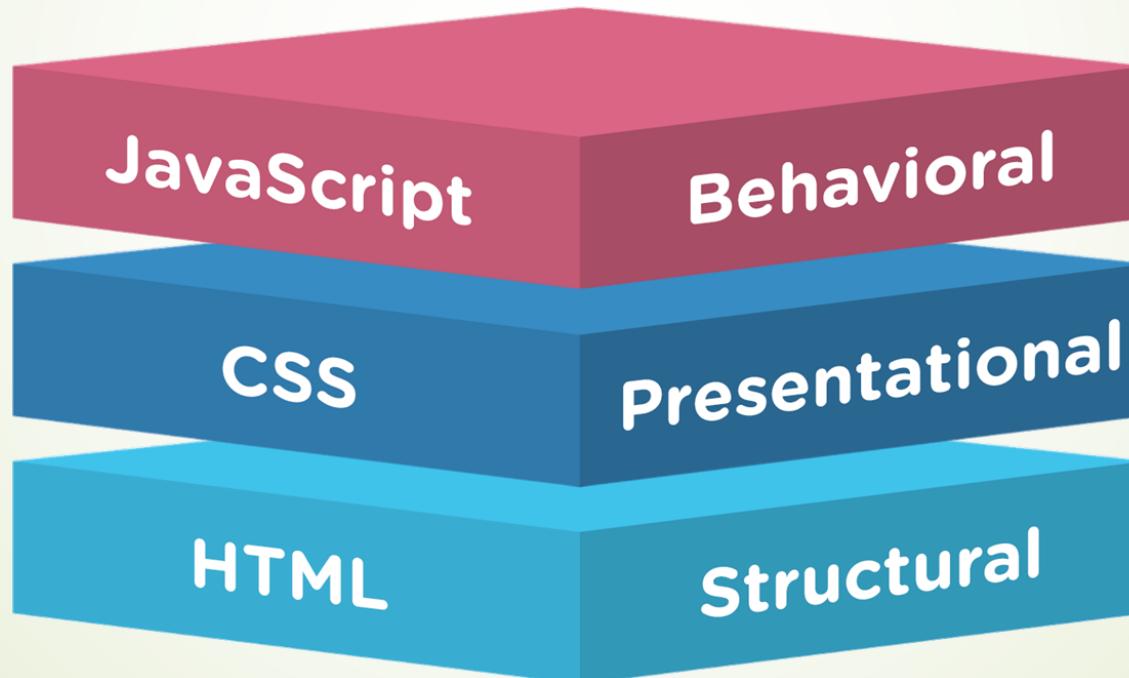
Développer des sites Web dynamiques avec JavaScript

Historique

- ▶ JavaScript a été créé en mai 1995 par **Brendan Eich** en seulement dix jours pour le compte de la société Netscape
- ▶ Basé sur de nombreux langages (notamment Java) mais avec une syntaxe simplifiée pour les débutants
- ▶ Langage **client-side** qui s'exécute du côté client contrairement aux langages server-side qui eux s'exécutent côté serveur (ASPX, PHP ou JSP)
- ▶ Code géré et **exécuté directement par le navigateur** sans appel aux ressources du serveur qui héberge la page
- ▶ Premier langage de script à avoir été développé pour le Web et qui demeure encore aujourd'hui le **plus répandu**

Historique

- ▶ Permet d'étendre les possibilités du HTML car il ajoute de l'**interactivité** aux pages Web (HTML)



Quelques exemples d'utilisation

- ▶ Affichage de messages dans une page Web ou dans une boîte de dialogue
- ▶ Validation du contenu d'un formulaire
- ▶ Détection du navigateur employé et affichage de contenus adaptés aux différents navigateurs
- ▶ Avec les nouvelles **API HTML5**, :
 - ▶ Géolocalisation
 - ▶ Canvas
 - ▶ Drag & Drop
 - ▶ Web Storage
 - ▶ Offline Web Application
 - ▶ Web Workers et Web Messaging

Limites et avantages de JavaScript

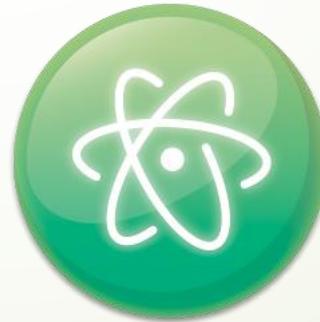
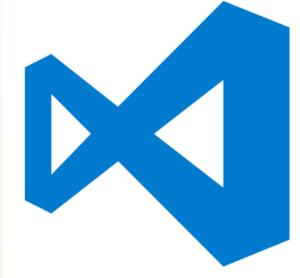
- ▶ Langage **interprété** (\neq compilé)
- ▶ Langage standardisé par **ECMAScript** en juin 1997
- ▶ Code intégré au HTML
- ▶ Code **faiblement typé**
- ▶ Liaison dynamique : les références des objets doivent être vérifiées au chargement
- ▶ Accessibilité du code (\neq confidentialité) → non crypté
- ▶ Code sûr : ne peut pas écrire sur le disque dur (sauf cookies)
- ▶ **Sensible à la casse** : mavariable \neq maVariable

Versions de JavaScript

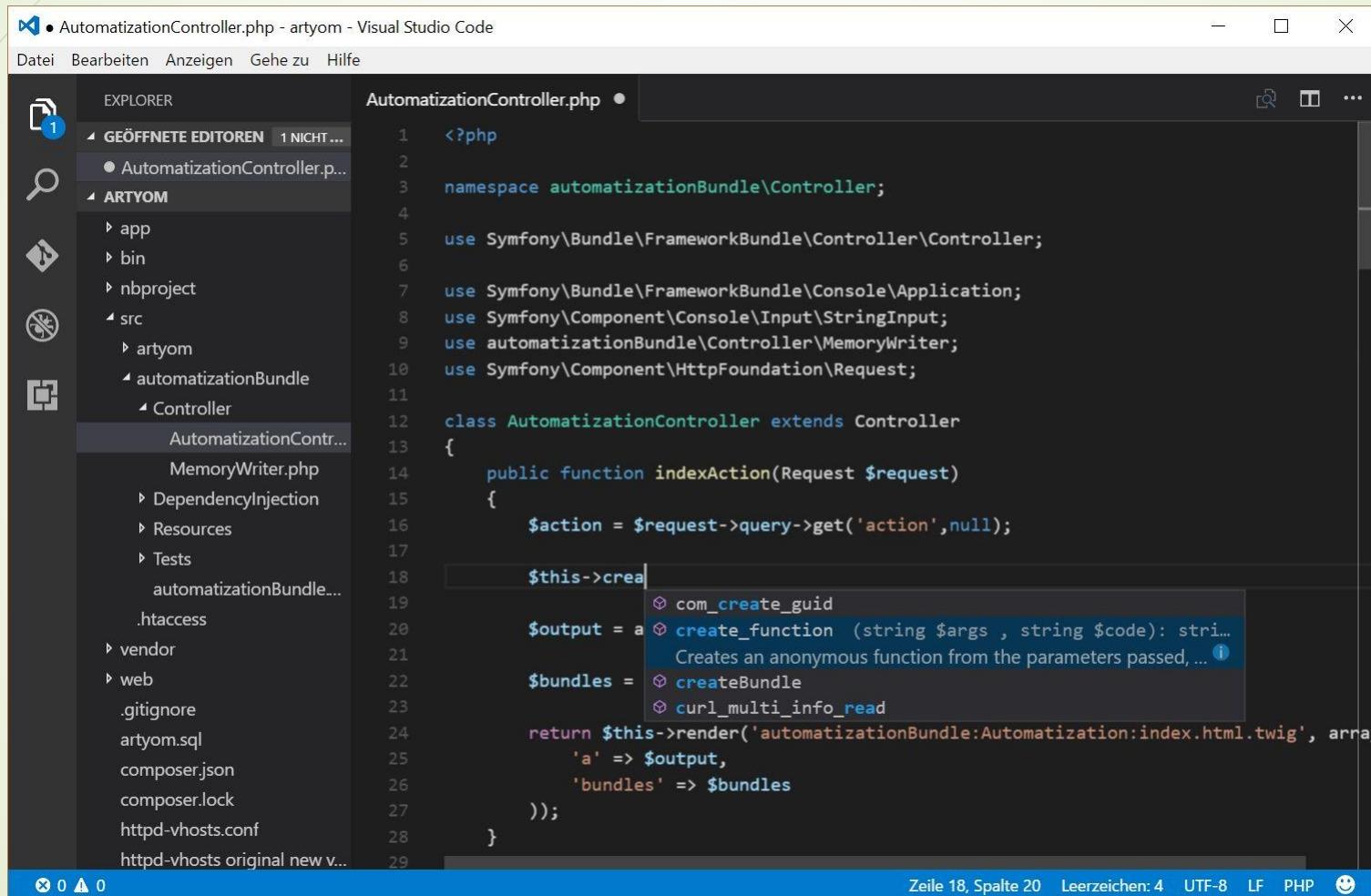
Ver	Nom Officiel	Description
ES1	ECMAScript 1 (1997)	Première édition
ES2	ECMAScript 2 (1998)	Modifications rédactionnelles
ES3	ECMAScript 3 (1999)	Ajout d'expressions régulières, de try/catch, de switch, de do-while, etc...
ES4	ECMAScript 4	Jamais sorti
ES5	ECMAScript 5 (2009)	Ajout du "mode strict", du support JSON, de String.trim(), de Array.isArray(), etc...
ES6	ECMAScript 2015	Ajout de let et const, de valeurs de paramètres par défaut, etc...
	ECMAScript 2016	Ajout de l'opérateur exponentiel (**), d'Array.includes(), etc...
	ECMAScript 2017	Ajout d'Object.entries(), d'Object.values(), de fonctions asynchrones
	ECMAScript 2018	Ajout de propriétés de rest, de spread, etc...

Outils de conception

- ▶ Les outils permettant de coder en JavaScript sont nombreux :
 - ▶ Un simple éditeur de texte comme Notepad de Windows
 - ▶ Un éditeur de code HTML tel que DreamWeaver d'Adobe
 - ▶ Un environnement de développement intégré (IDE) comme **SublimeText**, **VSCode** de Microsoft, **Atom** de Github ou encore **Brackets** d'Adobe



Outils de conception : VSCode



The screenshot shows the Visual Studio Code interface with a dark theme. The title bar reads "AutomatizationController.php - artyom - Visual Studio Code". The left sidebar is the Explorer view, showing a file tree with a folder named "ARTYOM" containing "app", "bin", "nbproject", "src", and "artyom". Inside "artyom", there is a "automatizationBundle" folder with a "Controller" folder containing "AutomatizationController.php", "MemoryWriter.php", "DependencyInjection", "Resources", "Tests", and "automatizationBundle". The main editor area displays the following PHP code:

```
<?php  
namespace automatizationBundle\Controller;  
  
use Symfony\Bundle\FrameworkBundle\Controller\Controller;  
  
use Symfony\Bundle\FrameworkBundle\Console\Application;  
use Symfony\Component\Console\Input\StringInput;  
use automatizationBundle\Controller\MemoryWriter;  
use Symfony\Component\HttpFoundation\Request;  
  
class AutomatizationController extends Controller  
{  
    public function indexAction(Request $request)  
    {  
        $action = $request->query->get('action',null);  
  
        $this->crea|  
        $output = a create_function (string $args , string $code): string  
        Creates an anonymous function from the parameters passed, ... ⓘ  
        $bundles = createBundle  
        curl_multi_info_read  
        return $this->render('automatizationBundle:Automatization:index.html.twig', array  
            'a' => $output,  
            'bundles' => $bundles  
        );  
    }  
}
```

A tooltip is displayed over the word "crea" in line 18, showing the definition of the `create_function` function from the `create_function` class in the `create` namespace.

The status bar at the bottom shows "Zeile 18, Spalte 20" and "Leerzeichen: 4".

Outils de conception : Atom

```
* Recommended: use a compiled version, especially in production!
*/
'use strict';

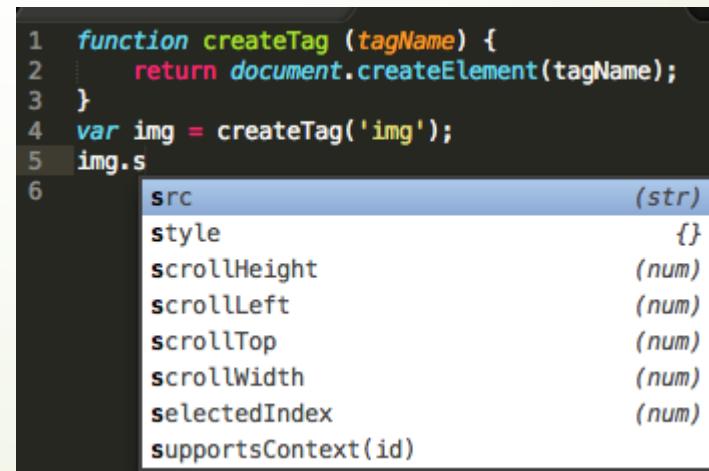
module.exports = (function(){

    var angu;
    angular
try {
    angular = require('angular');
} catch(err){ }

if(!angular || !angular.version){
    /*global window: true*/
    angular = window.angular;
    /*global window: false*/
}
}
```

Outils de conception

- ▶ L'usage de ces **IDE** permet de disposer d'un certains nombre d'outils facilitant l'écriture du code
- ▶ Il devient alors facile de :
 - ▶ Vérifier une syntaxe par la coloration automatique du code (**coloration syntaxique**)
 - ▶ Disposer de **l'autocomplétion** (proposition des méthodes ou propriétés disponibles de l'objet)
 - ▶ Connaître la valeur d'une variable lors de l'exécution d'un script



A screenshot of an IDE interface showing a code editor and a completion dropdown menu. The code in the editor is:

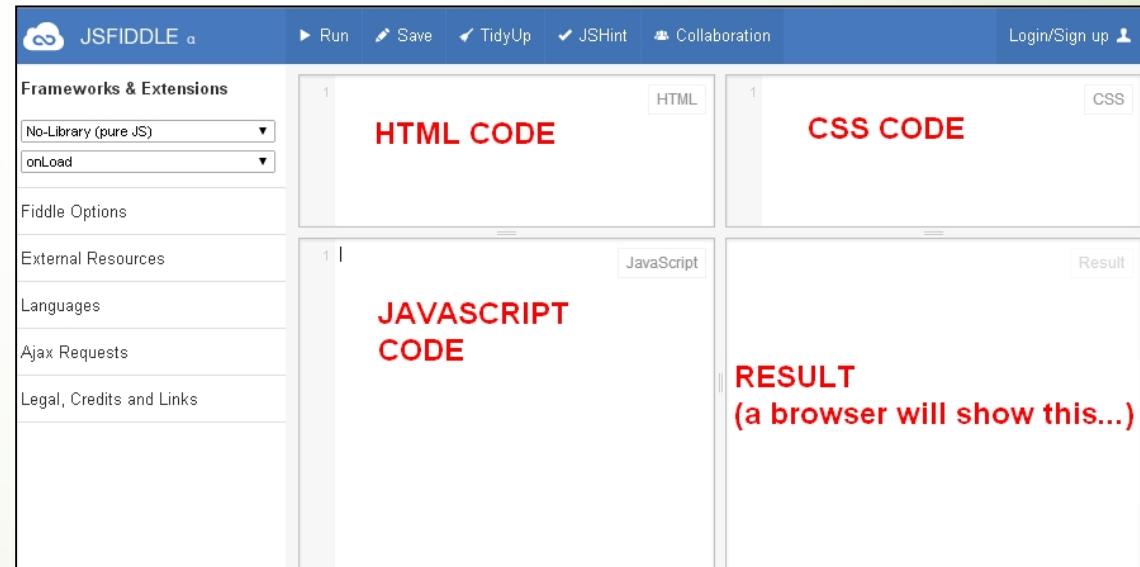
```
1 function createTag (tagName) {  
2     return document.createElement(tagName);  
3 }  
4 var img = createTag('img');  
5 img.s
```

The completion dropdown shows several properties starting with 'src':

- src (str)
- style {}
- scrollHeight (num)
- scrollLeft (num)
- scrollTop (num)
- scrollWidth (num)
- selectedIndex (num)
- supportsContext(id)

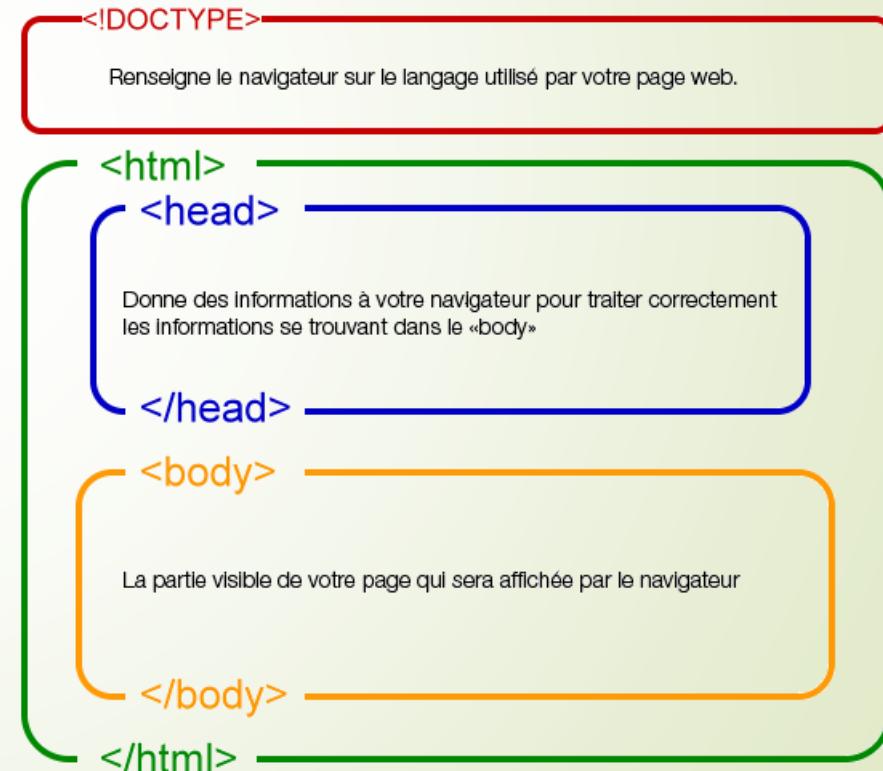
Outils de conception

- ▶ Il est également possible de trouver sur le web de nombreux sites web permettant d'écrire du code en JavaScript et d'avoir immédiatement un aperçu du résultat :
 - ▶ <https://jsfiddle.net/>
 - ▶ <http://jsbin.com/>
- ▶ Ces sites sont très pratiques lorsque l'on souhaite tester rapidement un bout de code par exemple ou pour effectuer des démonstrations



Environnement optimal de test

- ▶ Développer en JavaScript nécessite un minimum de connaissances en **HTML** et notamment sur la notion de balise permettant de se situer dans la page
- ▶ Rappelons qu'une page HTML se divise en deux grandes parties :
 - ▶ la partie **head** (tête en français) dans laquelle se situent les informations correspondant à la description du contenu
 - ▶ la partie **body** (corps en français) où figure le code permettant la construction des objets dans la page (champs de formulaire, zone de texte, image, etc.)

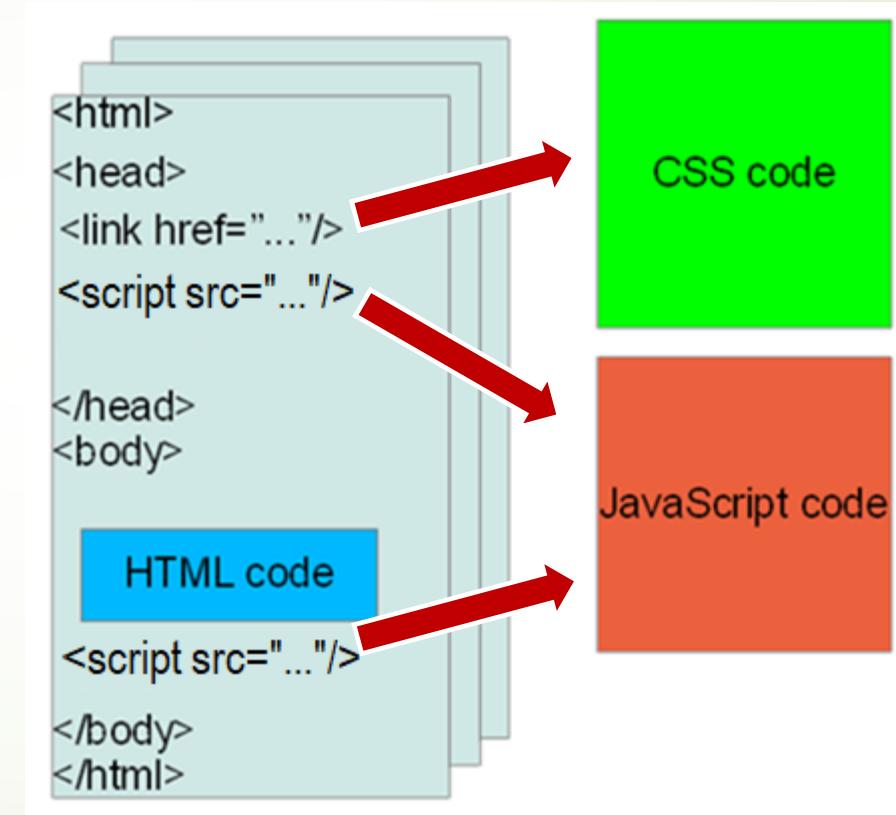


Environnement optimal de test

- ▶ Un script JavaScript peut se trouver au choix dans l'une ou l'autre de ces deux parties : **head** ou **body**
- ▶ Par convention, les scripts se sont souvent retrouvés dans la partie **head** d'une page HTML
- ▶ Leur exécution peut alors être :
 - ▶ **Immédiate** (au chargement de la page)
 - ▶ ou **Différée** (clic sur un bouton, par exemple)
- ▶ Pour des questions de performance, on recommande plutôt de placer les scripts en fin de page juste avant **</body>**

Environnement optimal de test

- ▶ Enfin pour obtenir une meilleure clarté dans un document HTML :
 - ▶ Les scripts définissant des fonctions vont entre les balises `<head></head>`
 - ▶ Tous les autres scripts et les scripts faisant références à ces fonctions vont entre les balises `<body></body>`



Environnement optimal de test

- ▶ L'élément **noscript** est conçu pour fournir une information à l'utilisateur dans le cas où son navigateur n'est pas capable d'exécuter les scripts
- ▶ Dans ce cas, le contenu entre les marqueurs de **noscript** sera vu comme du texte normal

```
<noscript>
  <!-- Un lien vers un site externe -->
  <a href="http://www.mozilla.com/">Un autre site</a>
</noscript>
<p>Elle est où la poulette ?</p>
```





Insertion du code JavaScript

Développer des sites Web dynamiques avec JavaScript

Les deux types d'exécution du code JavaScript

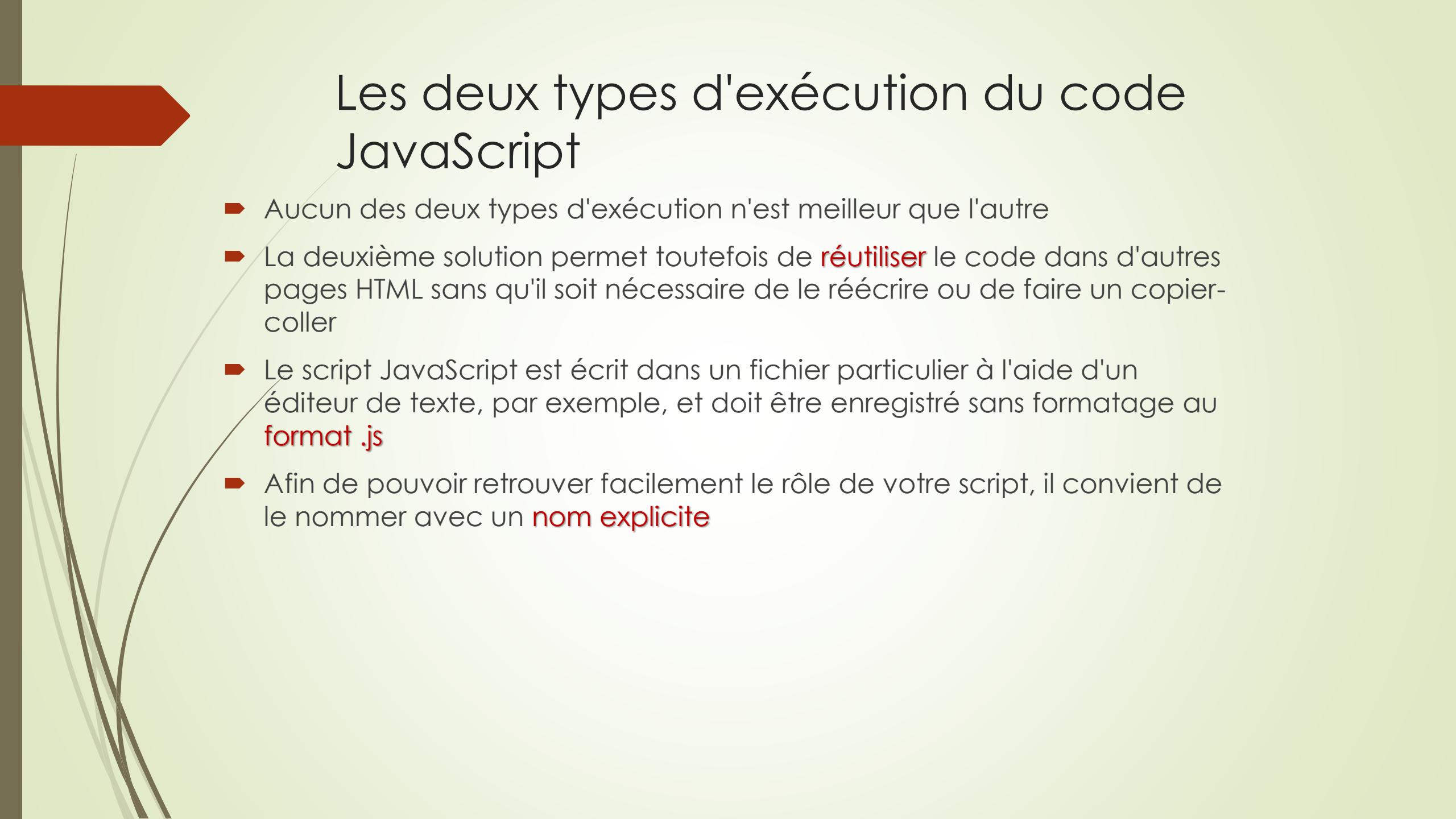
- ▶ Les blocs de script JavaScript peuvent être directement présents :
 - ▶ dans le code HTML de la page entre deux balises (une de début et une de fin)

```
<script>  
// instructions  
</script>
```

- ▶ ou écrits dans un fichier JavaScript au format .js, totalement dissocié du code HTML de la page

```
<script src="fichier_externe.js">  
</script>
```

- ▶ Le premier cas est désigné sous le terme de JavaScript interne par opposition au second, appelé JavaScript externe



Les deux types d'exécution du code JavaScript

- ▶ Aucun des deux types d'exécution n'est meilleur que l'autre
- ▶ La deuxième solution permet toutefois de **réutiliser** le code dans d'autres pages HTML sans qu'il soit nécessaire de le réécrire ou de faire un copier-coller
- ▶ Le script JavaScript est écrit dans un fichier particulier à l'aide d'un éditeur de texte, par exemple, et doit être enregistré sans formatage au **format .js**
- ▶ Afin de pouvoir retrouver facilement le rôle de votre script, il convient de le nommer avec un **nom explicite**

Règles de syntaxe du code

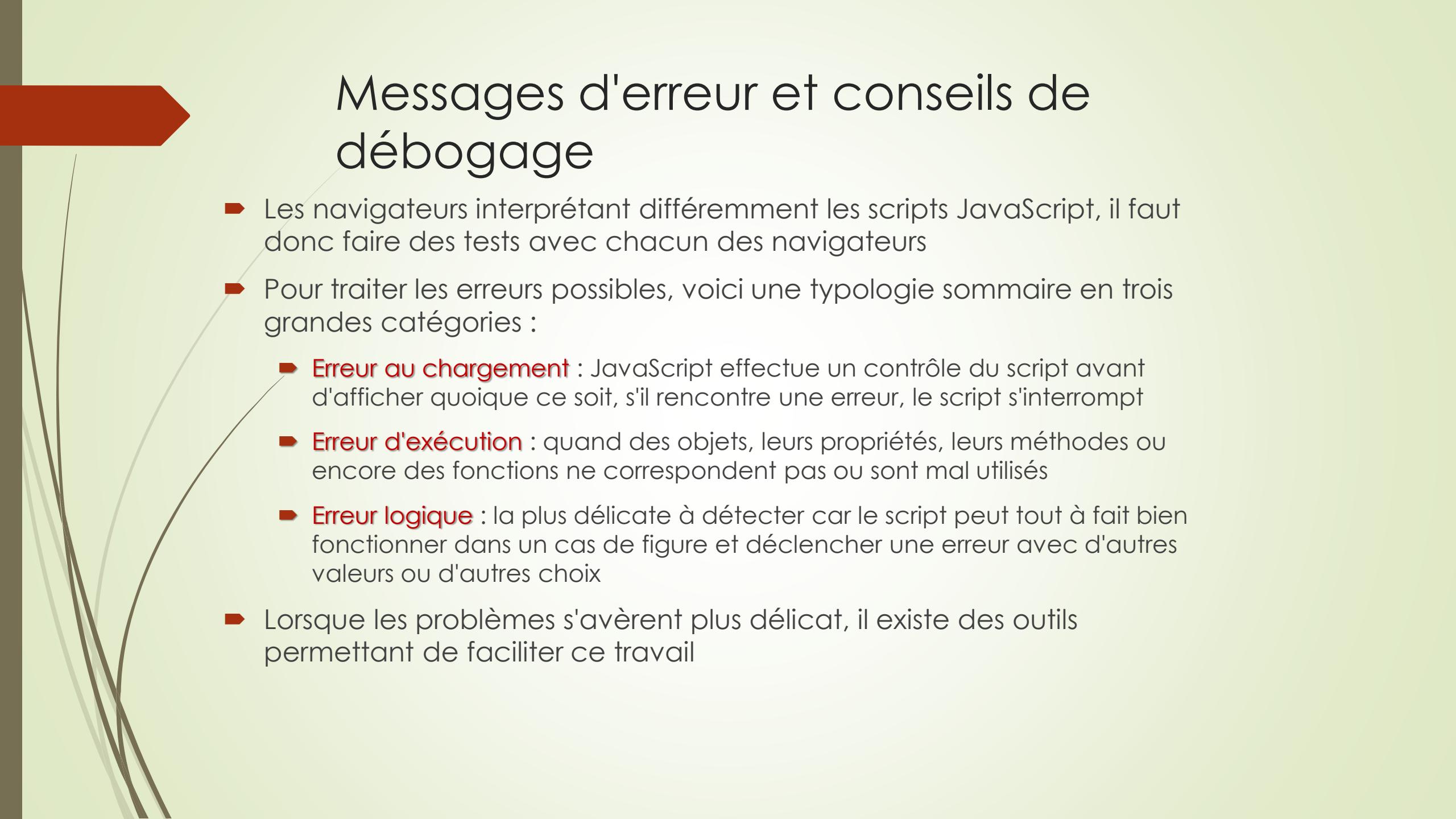
- ▶ Langage **sensible à la casse**, c'est-à-dire qu'il différencie les majuscules des minuscules : mavariable ≠ maVariable
- ▶ Ajouter des **commentaires** dans un script JavaScript peut s'avérer fort utile pour faciliter la lisibilité et la maintenance du code
 - ▶ Commentaires contenus sur une seule ligne précédés du double slash `//`
 - ▶ Commentaires contenus sur une plusieurs lignes précédés de `/*` et terminés par `*/`

```
<script>
//Ceci est un commentaire monoligne
</script>
<script language="javascript">
/* Ceci est un commentaire
sur plusieurs lignes */
</script>
```

Règles de syntaxe du code

- ▶ Chaque ligne de code JavaScript doit se terminer par un **point-virgule** sauf exception de syntaxe que nous détaillerons plus tard
- ▶ Lorsque les lignes de codes commencent à être nombreuses il peut être utile d'utiliser **l'indentation** : règle de présentation des scripts consistant à décaler vers la droite des instructions se correspondant

```
<script>
var i,j=0;
for (i=0;i<2;i++) {
    alert("voici la valeur de mon premier compteur i: "+i);
    for (j=0;j<2;j++) {
        alert("voici la valeur de mon deuxième compteur j: "+j);
    }
}
</script>
```

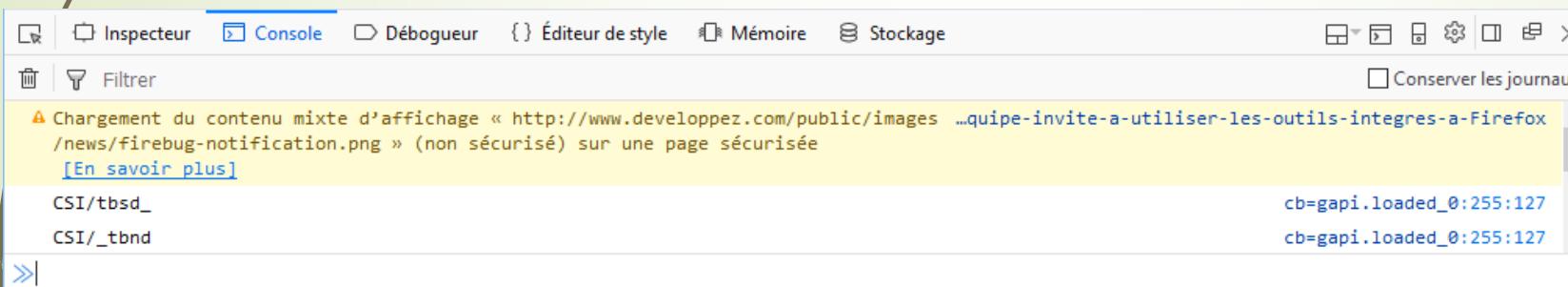


Messages d'erreur et conseils de débogage

- ▶ Les navigateurs interprétant différemment les scripts JavaScript, il faut donc faire des tests avec chacun des navigateurs
- ▶ Pour traiter les erreurs possibles, voici une typologie sommaire en trois grandes catégories :
 - ▶ **Erreur au chargement** : JavaScript effectue un contrôle du script avant d'afficher quoique ce soit, s'il rencontre une erreur, le script s'interrompt
 - ▶ **Erreur d'exécution** : quand des objets, leurs propriétés, leurs méthodes ou encore des fonctions ne correspondent pas ou sont mal utilisés
 - ▶ **Erreur logique** : la plus délicate à détecter car le script peut tout à fait bien fonctionner dans un cas de figure et déclencher une erreur avec d'autres valeurs ou d'autres choix
- ▶ Lorsque les problèmes s'avèrent plus délicat, il existe des outils permettant de faciliter ce travail

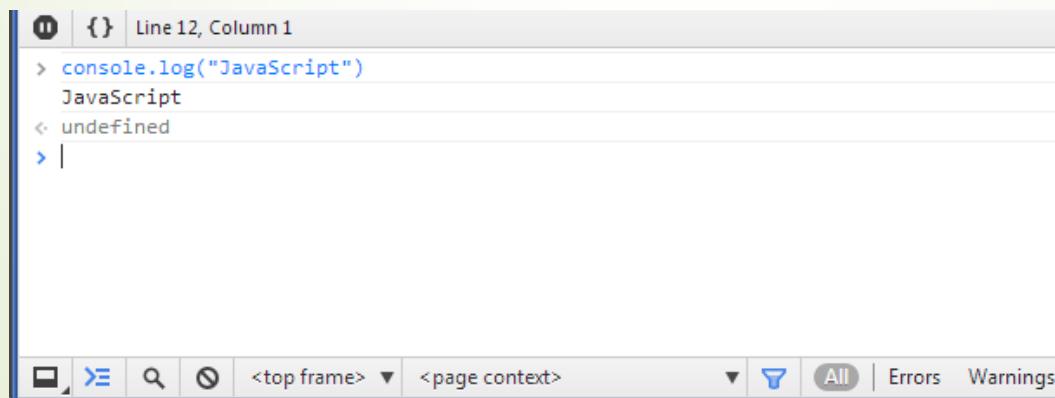
Les outils de débogage JavaScript

- ▶ Les environnements de développement qui fournissent un moteur ont généralement un **débogueur intégré** qui permet d'analyser le code dès qu'il est exécuté
- ▶ Sans environnement de développement intégré, il est aussi possible d'utiliser le navigateur web qui fournira des **outils de développement web** activés la touche **F12** pour :
 - ▶ Inspecter le code HTML et modifier style et mise en page en temps réel
 - ▶ Utiliser le débogueur JavaScript disponible indépendamment du navigateur
 - ▶ Analyser avec précision l'utilisation et la performance du réseau



Les outils de débogage JavaScript

- ▶ Chrome, Firefox, Internet Explorer et Safari ont aujourd'hui des **outils de développement web** performants
- ▶ Ils offrent en outre un environnement pour l'exécution directe du code JavaScript : la **console**
 - ▶ Dans cette fenêtre spéciale on peut écrire du code JavaScript pour voir son exécution immédiate
 - ▶ Elle peut être utilisée pour vérifier l'affichage d'un script, grâce à l'objet du même nom **console**



A screenshot of a browser's developer tools console. The title bar says "Line 12, Column 1". The main area shows the following code:

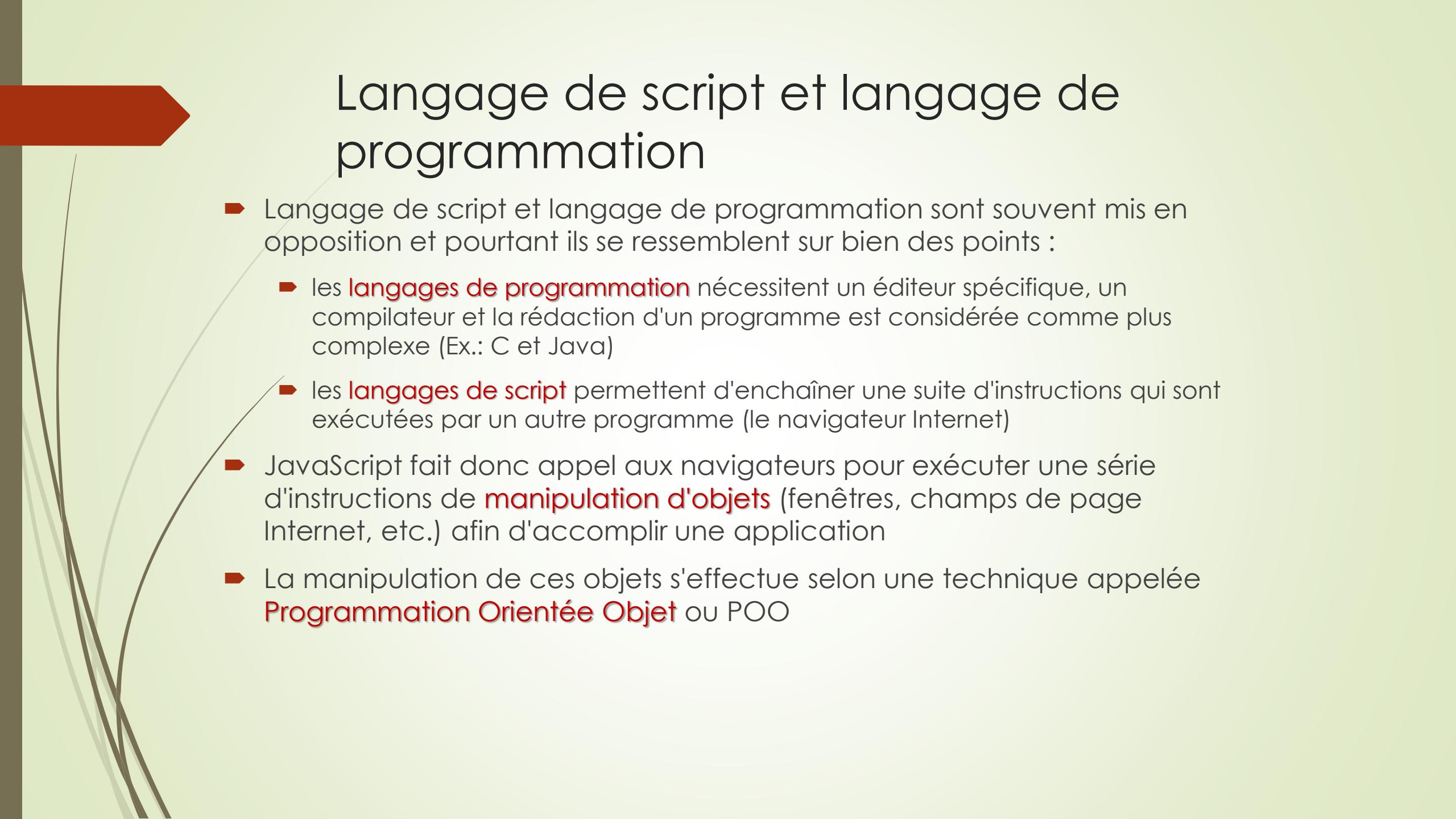
```
> console.log("JavaScript")
JavaScript
< undefined
>
```

The bottom status bar shows various icons and text: "top frame", "page context", "All", "Errors", and "Warnings".



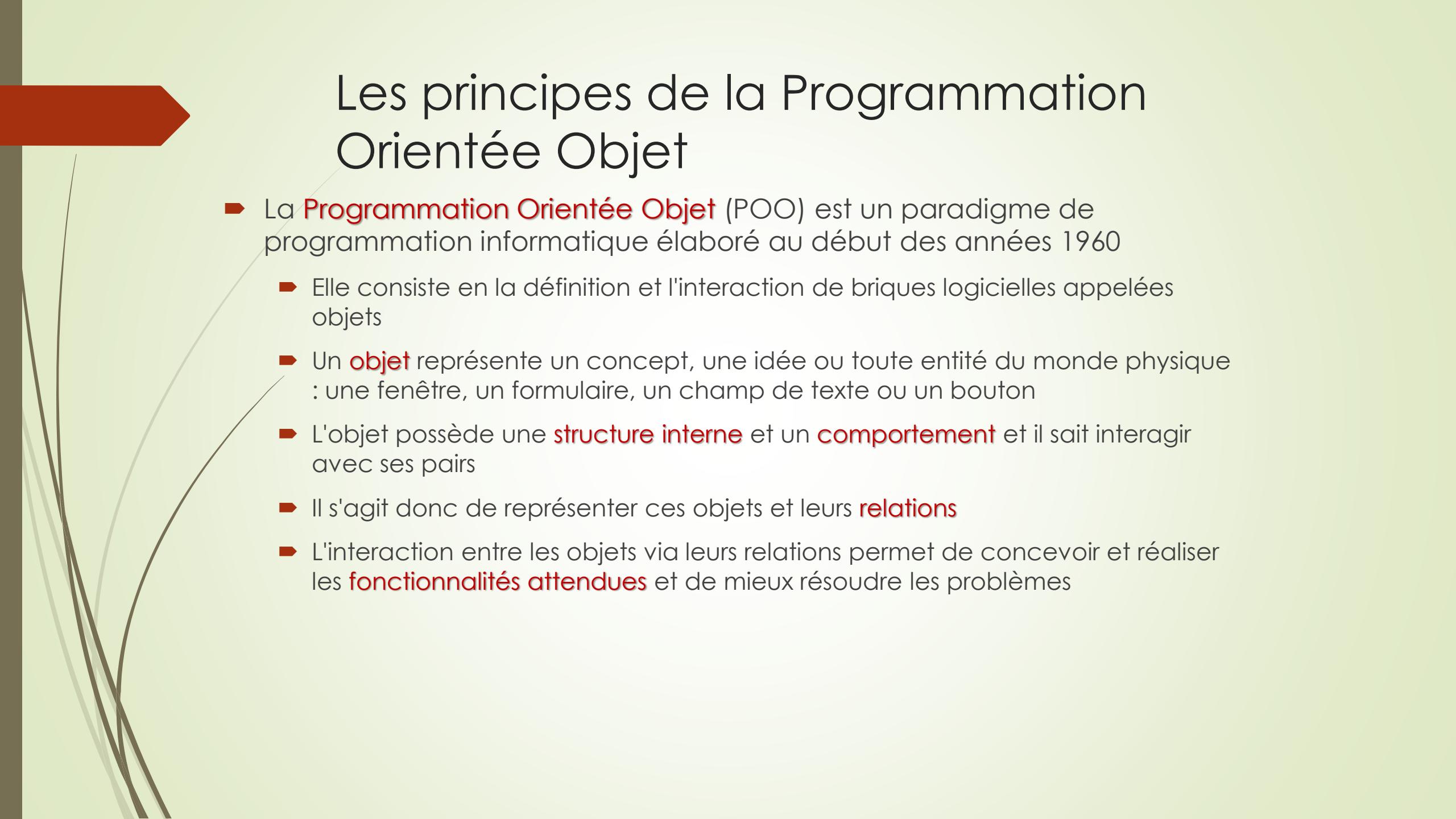
Initiation à la Programmation Orientée Objet et JavaScript

Développer des sites Web dynamiques avec JavaScript



Langage de script et langage de programmation

- ▶ Langage de script et langage de programmation sont souvent mis en opposition et pourtant ils se ressemblent sur bien des points :
 - ▶ les **langages de programmation** nécessitent un éditeur spécifique, un compilateur et la rédaction d'un programme est considérée comme plus complexe (Ex.: C et Java)
 - ▶ les **langages de script** permettent d'enchaîner une suite d'instructions qui sont exécutées par un autre programme (le navigateur Internet)
- ▶ JavaScript fait donc appel aux navigateurs pour exécuter une série d'instructions de **manipulation d'objets** (fenêtres, champs de page Internet, etc.) afin d'accomplir une application
- ▶ La manipulation de ces objets s'effectue selon une technique appelée **Programmation Orientée Objet** ou POO

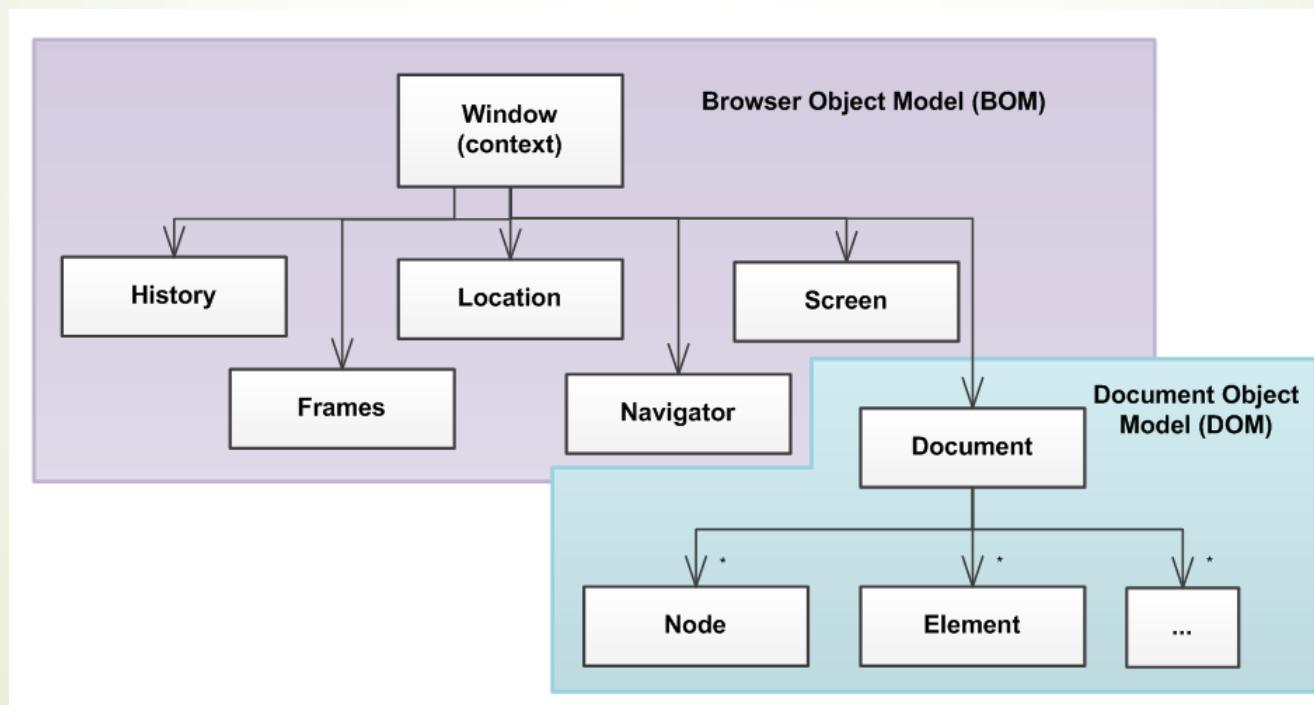


Les principes de la Programmation Orientée Objet

- ▶ La **Programmation Orientée Objet** (POO) est un paradigme de programmation informatique élaboré au début des années 1960
 - ▶ Elle consiste en la définition et l'interaction de briques logicielles appelées objets
 - ▶ Un **objet** représente un concept, une idée ou toute entité du monde physique : une fenêtre, un formulaire, un champ de texte ou un bouton
 - ▶ L'objet possède une **structure interne** et un **comportement** et il sait interagir avec ses pairs
 - ▶ Il s'agit donc de représenter ces objets et leurs **relations**
 - ▶ L'interaction entre les objets via leurs relations permet de concevoir et réaliser les **fonctionnalités attendues** et de mieux résoudre les problèmes

Les principes de la Programmation Orientée Objet

- ▶ Tous les objets n'ont pas la même importance car certains objets dépendent d'autres
- ▶ Les termes **notion de hiérarchie** sont utilisés pour caractériser cette dépendance



Les principes de la Programmation Orientée Objet

- ▶ Pour atteindre un objet il faut **décrire un chemin** en partant de l'objet le plus haut hiérarchiquement pour arriver à celui le plus bas
- ▶ À chaque changement d'objet, un point est ajouté : cette forme de syntaxe se nomme **syntaxe pointée**
- ▶ Si dans une page comportant un formulaire nommé formu contenant un champ pseudo on désire accéder à ce champ on utilisera la syntaxe suivante : **document.formu.pseudo**

Manipulation des formulaires

file:///Users/baptiste/Dropbox

Rechercher

Formulaire d'inscription

Pseudo :

Mot de passe :

Courriel :

M'envoyer un courriel de confirmation

M'abonner à la newsletter et aux promotions

M'abonner uniquement à la newsletter

Ne pas m'abonner

Nationalité :

Envoyer Annuler

Objets, Méthodes et Propriétés

- La syntaxe pointée désigne également l'accès à des **méthodes** ou des **propriétés** qui permettent de manipuler/décrire ces objets :
 - Propriétés** : constituent un ensemble d'attributs qui permettent de modifier l'apparence d'un objet
 - Méthodes** : représentent des actions réalisables par cet objet
- D'un point de vue syntaxique, le point est utilisé pour séparer l'objet de sa propriété ou de sa méthode :
myCar.Couleur ou **myCar.Avancer**

Propriétés
Couleur
Forme



Méthodes
Avancer
Tourner
Reculer

Objets, Méthodes et Propriétés

- ▶ Un des principaux objets prédéfinis de JavaScript est l'objet **window** qui correspond à la fenêtre du navigateur
- ▶ Pour imprimer la page active on utilisera la méthode **print()** (imprimer) de l'objet **window** (fenêtre)

```
<script>
window.print();
</script>
```
- ▶ L'objet **window** étant le plus élevé dans la hiérarchie, il n'est pas nécessaire de le nommer à chaque manipulation

```
<script>
print();
</script>
```

Objets, Méthodes et Propriétés

- ▶ En JavaScript, il existe deux types d'objets :
 - ▶ les **objets prédéfinis** qui se retrouvent dans la hiérarchie d'objets (abordée précédemment) enrichie à chaque nouvelle version de JavaScript
 - ▶ les **objets créés par le développeur** à l'aide d'une fonction (abordé plus loin)
- ▶ Un navigateur pas à jour ne pourra pas disposer des nouveaux objets, propriétés ou méthodes car à chaque propriété ou méthode, correspond une version de chaque navigateur
- ▶ C'est donc l'objet qui reste l'élément de base du langage JavaScript
- ▶ La vraie difficulté réside plus dans le type du navigateur car certaines méthodes ou propriétés peuvent ne pas être supportées par certains navigateurs et acceptées par d'autres
- ▶ Voir le tableau de compatibilité : <https://kangax.github.io/compat-table/es6/>

Méthodes JavaScript : alert()

- ▶ La méthode **alert()** de l'objet **window** permet l'affichage d'une boîte de dialogue comprenant un message d'avertissement
- ▶ Lors de l'affichage de cette boîte, JavaScript stoppe son déroulement et attend que l'utilisateur clique sur le bouton **OK**

```
<script>
alert("Ceci est une boîte de dialogue affichée par
JavaScript");
</script>
```



Méthodes JavaScript : confirm()

- ▶ La méthode **confirm()** de l'objet **window** affiche une boîte de dialogue avec un message suivi des deux boutons **OK** et **Annuler**
- ▶ En fonction du bouton cliqué on peut renvoyer la valeur vers une variable (**true** pour **OK** et **false** pour **Annuler**)

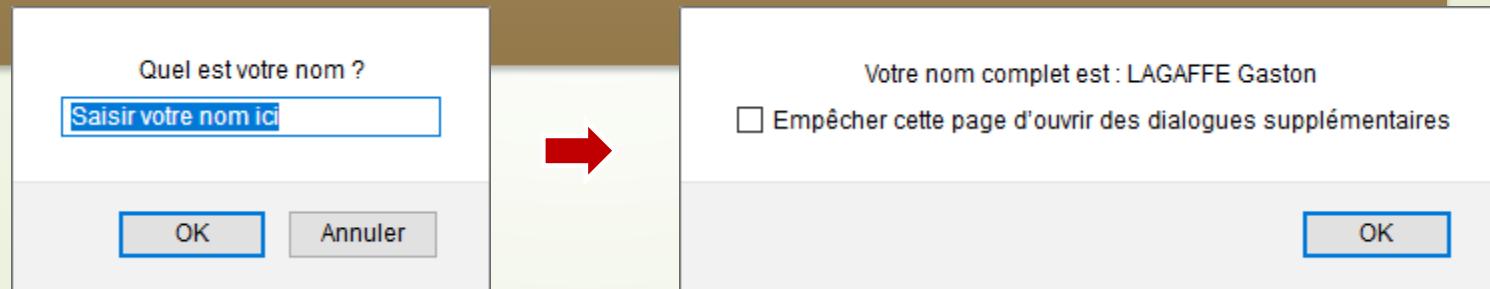
```
<script>
var reponse=window.confirm("ok = true, Annuler = false");
alert("la valeur de la variable réponse est : "+reponse);
</script>
```

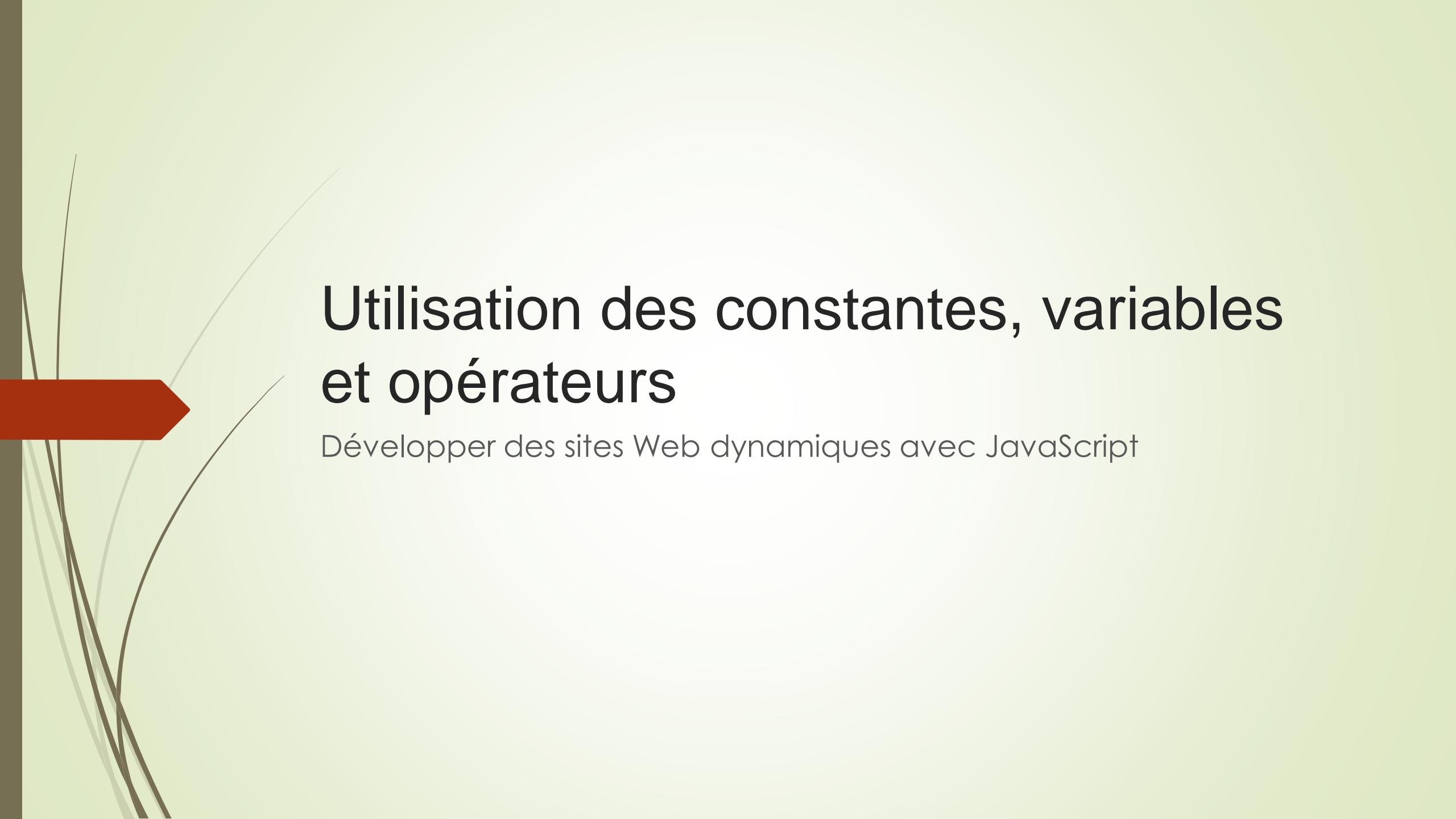


Méthodes JavaScript : prompt()

- ▶ La méthode **prompt()** permet d'afficher une boîte de message proposant de saisir une valeur dans un champ appelé **invite**
- ▶ Cette boîte contient deux boutons : **OK** qui affecte la valeur saisie à une variable et **Annuler** qui affecte la valeur **Null** à la variable

```
<script>
nom=prompt("Quel est votre nom ?", "Saisir votre nom ici");
prenom=prompt("Quel est votre prenom ?", "Saisir votre
prenom ici");
alert("Votre nom complet est : \r"+nom+"\r "+prenom);
</script>
```





Utilisation des constantes, variables et opérateurs

Développer des sites Web dynamiques avec JavaScript

Typologie et utilisation des constantes

- ▶ Une **constante** est un élément d'information dont la valeur est indiquée explicitement dans le code JavaScript
- ▶ La déclaration **const** permet de créer une constante nommée accessible uniquement en lecture

```
const Ma_constante = valeur de la constante ;
```
- ▶ La valeur d'une constante ne peut donc plus être modifiée par des réaffectations ultérieures
- ▶ Les constantes JavaScript peuvent être classées en plusieurs catégories

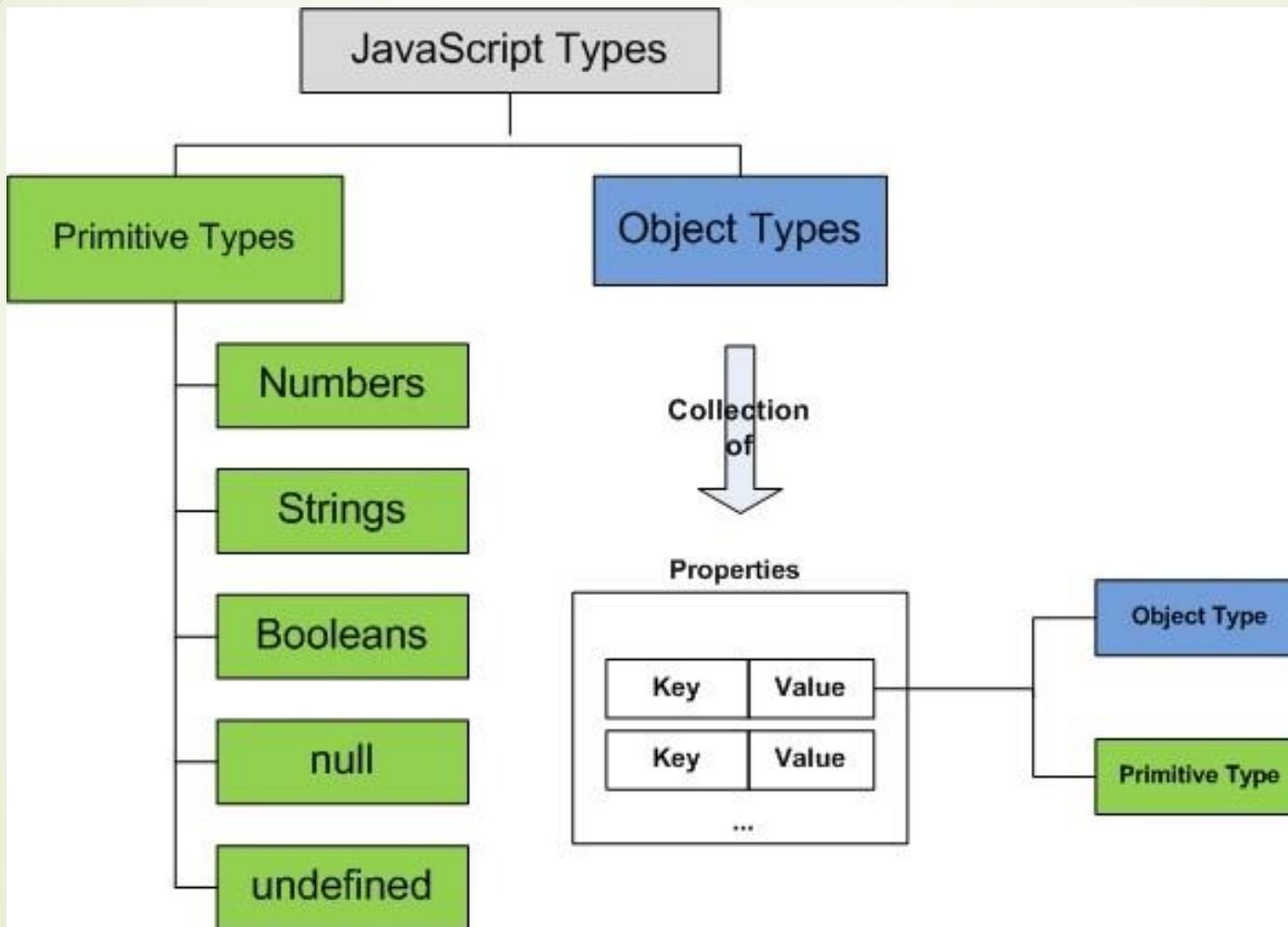
Typologie et utilisation des constantes

- ▶ **Constantes arithmétiques** correspondant à un nombre constitué d'une suite de chiffres, on distingue :
 - ▶ **Constante décimale** : suite de 17 chiffres maximum allant de 0 à 9 et pouvant être négative (Ex.: 64 ou -64 ou 64.14 ou -3.14)
 - ▶ **Constante octale** : suite de chiffres allant de 0 à 7 débutant par 0 et obligatoirement entière et positive (Ex.: 02542571)
 - ▶ **Constante hexadécimale** : suite de 16 caractères comprenant des chiffres décimaux de 0 à 9 auxquels on ajoute des lettres de A à F et commençant par 0 suivi d'un x (Ex.: 0xF45B)
- ▶ **Constantes chaînes de caractères** correspondant à une suite de caractères encadrée par des guillemets ou des apostrophes qui peuvent être des lettres, des chiffres ou une association des deux
- ▶ **Constantes booléennes** ne peuvent correspondre qu'aux valeurs **true** ou **false**

Typologie des variables

- ▶ Contrairement aux constantes, les **variables** peuvent changer de valeur tout au long du déroulement du script
- ▶ JavaScript retiendra la dernière définition de la variable, c'est là que réside l'intérêt de leur utilisation
- ▶ JavaScript est **faiblement typé**, les variables n'ont donc pas besoin de correspondre à un type pour fonctionner :
 - ▶ **Avantage** : en se passant de la détermination du type on économise du code
 - ▶ **Inconvénient** : on peut parfois aboutir à des incohérences comme additionner du texte et des nombres sans que cela ne gêne JavaScript
- ▶ Les types de variables correspondent à ceux des constantes (texte, numérique ou booléen = **types primitifs**)

Typologie des variables



Portée et déclaration des variables

- ▶ Portée des variables :
 - ▶ Lorsqu'une variable est définie à l'intérieur d'une **fonction** à l'aide de mot-clé **var**, sa portée est dite **locale** : sa valeur ne peut pas être récupérée dans une autre fonction
 - ▶ Lorsqu'une variable est définie directement dans le script, sa portée est dite **globale** : elle est alors disponible à tout moment
- ▶ La déclaration d'une variable s'effectue à l'aide du mot clé **var** ou **let** : on parle alors de **déclaration explicite**

```
var nom_de_la_variable ;
```

- ▶ On peut déclarer plusieurs variables sur une seule ligne : elles seront alors séparées par une virgule

```
var mavariable1, mavariable2, mavariable3 ;
```

Portée de bloc

- ▶ Les mots-clés **let** et **const** définissent des variables et constantes qui sont locales au **bloc** où elles apparaissent.
- ▶ Une variable définie dans un bloc sera accessible dans tout autre bloc interne (sauf si elle est masquée).

```
{  
  let x =3;  
  console.log(x);  
}  
console.log(x); // error
```

```
let x = 3; // variable de portée globale  
let y = 5;  
if (true) {  
  console.log(x); // 3  
  console.log(y); // 5  
  const x = 8; // x global est masqué  
  console.log(x); // 8  
  if (true) {  
    console.log(x); // 8  
    console.log(y); // 5  
  }  
  console.log(x); // 8  
}  
console.log(x); // 3
```

Affectation de variables

- ▶ L'affectation peut s'effectuer directement dans le code par l'usage du signe égal

```
var mavariable = 10 ;
```

- ▶ Il est possible de le faire indirectement par l'intermédiaire d'un calcul impliquant constantes ou variables

```
var somme = mavariable1 + mavariable2 ;
```

- ▶ Enfin, il est possible de demander à l'utilisateur une valeur qui sera affectée à la variable par la méthode prompt()

```
var reponse = prompt("Valeur de la variable ? ") ;
```

- ▶ Les variables n'ayant pas reçu de valeurs sont dites **undefined**

Affectation de variables : Exemples

```
var iNum1 = 55;
var iNum2 = 070; // octal : vaut 56 en décimal
var iNum3 = 0x1f; // hexadécimal : vaut 31 en décimal
var iNum4 = 0xAB; // hexadécimal : vaut 171 en décimal
var iNum5 = 5.0; // virgule flottante
var iNum6 = 3.125e7; // notation avec exposant

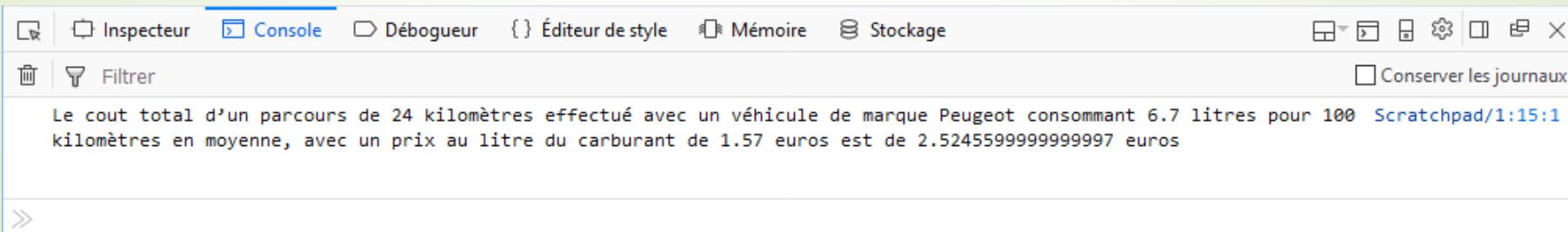
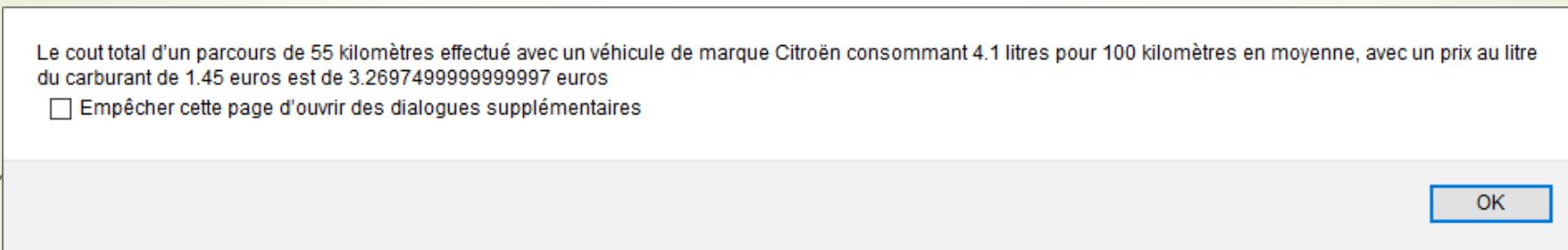
alert( isNaN ( "56" ) ); // renvoie false
alert( isNaN ( 56 ) ); // renvoie false
alert( isNaN ( "blue" ) ); // renvoie true

var sColor1 = "blue";
var sColor2 = 'green';

var bFound = true;
var bLost = false;
```

Affichage des variables

- ▶ L'affichage des variables s'effectue généralement :
 - ▶ par l'intermédiaire des méthodes **alert()** ou **confirm()** de l'objet window



Transfert de valeurs entre variables et conversion de type

- ▶ Pour donner la valeur d'une variable à une autre, il suffit d'utiliser le signe égal pour **transférer la valeur de la variable** à droite du signe égal dans celle située à gauche du signe égal.

```
var mavariable2 = mavariable1 ;
```

- ▶ Il est possible de convertir une variable d'un type à un autre (généralement de texte vers du numérique) grâce aux méthodes :
 - ▶ **parseInt()** : conversion de texte vers un nombre entier
 - ▶ et **parseFloat()** : conversion de texte vers un nombre réel
- ▶ À l'inverse, il est possible de convertir une variable numérique en texte par l'intermédiaire de la méthode **toString()**

Transfert de valeurs entre variables et conversion de type : Exemples

► Numérique vers texte :

```
var iNum1 = parseInt("1234blue"); // renvoie 1234
var iNum2 = parseInt("0xA"); // renvoie 10
var iNum3 = parseInt("22.5"); // renvoie 22
var iNum4 = parseInt("blue"); // renvoie NaN
var fNum1 = parseFloat("1234blue"); // renvoie 1234.0
var fNum2 = parseFloat("0xA"); // renvoie 0
var fNum3 = parseFloat("22.5"); // renvoie 22.5
var fNum4 = parseFloat("22.34.5"); // renvoie 22.34
var fNum5 = parseFloat("0908"); // renvoie 908
var fNum6 = parseFloat("blue"); // renvoie NaN
```

Transfert de valeurs entre variables et conversion de type : Exemples

- ▶ Texte vers du numérique :

```
var bFound = false;  
alert(bFound.toString()); // renvoie "false"  
  
var iNum1 = 10;  
var fNum2 = 10.0;  
alert(iNum1.toString()); // renvoie "10"  
alert(fNum2.toString()); // renvoie "10"  
  
var iNum = 10;  
alert(iNum1.toString(2)); // renvoie "1010" : binaire  
alert(iNum1.toString(8)); // renvoie "12" : octal  
alert(iNum1.toString(16)); // renvoie "A" : hexadecimal
```

Règles de nommage et mots réservés

► Conseils pour le **nommage**

- Une variable ne peut contenir d'espaces : remplacer par l'underscore " _ "
- Le premier caractère ne peut pas être un point ":"
- Peut contenir des majuscules et des minuscules mais il est important de bien respecter ce changement de casse lors des appels à cette variable

► Mots **réservés**

- Tous les mots correspondant généralement à des objets, propriétés ou méthodes déjà utilisés par

JavaScript						
abstract	continue	export	if	native	short	true
boolean	debugger	extends	implements	new	static	try
break	default	false	import	null	super	typeof
byte	delete	finally	in	package	switch	var
case	do	float	instanceof	private	synchronized	void
char	double	for	int	protected	this	volatile
class	else	function	interface	public	throw	while
const	enum	goto	long	return	transient	with

Opérateurs associatifs

- ▶ Il s'agit de l'opérateur **d'affectation** égal qui permet de donner une valeur à une variable : ce qui est à droite du signe égal est affecté à ce qui est à gauche

```
mavariable = 1 ;
```

- ▶ Cet opérateur peut également être combiné avec les opérateurs de calcul pour réaliser des calculs au moment de l'affectation :

```
mavariable += 10 ;
```

- ▶ Cette instruction remplace astucieusement l'instruction :

```
mavariable = mavariable + 10 ;
```

Opérateurs arithmétiques

Opérateur	Nom	Rôle	Exemple	Résultat
+	Plus	Addition	15+10	25
-	Moins	Soustraction	15-10	5
*	Produit ou Multiplié	Multiplication	15*10	150
/	Divisé	Division	15/10	1.5
%	Modulo	Reste de division	5%3	1
++	Incrémentation	Incrémantation par 1	variable = 1 variable++	2
--	Décrémentation	Décrémentation par 1	variable = 2 variable	1
-	Négation	Opposé	variable = 1 variable = -variable	-1

Opérateurs de comparaison

Opérateur	Nom	Rôle	Exemple	Résultat
<	Inférieur		1 < 2	true
<=	Inférieur ou égal		2 <= 2	true
==	Egal		2 == 2	true
>	Supérieur		2 > 1	true
>=	Supérieur ou égal		2 >= 2	true
!=	Différent		1 != 2	true
==>	Strictement égal		"1" ==> 1	false
!==>	Strictement différent		"1" !==> 1	true

Ne pas confondre l'opérateur de comparaison == avec l'opérateur d'affectation =, ce dernier sert à affecter une valeur dans la variable située à gauche du symbole

Opérateurs logiques (ou booléens)

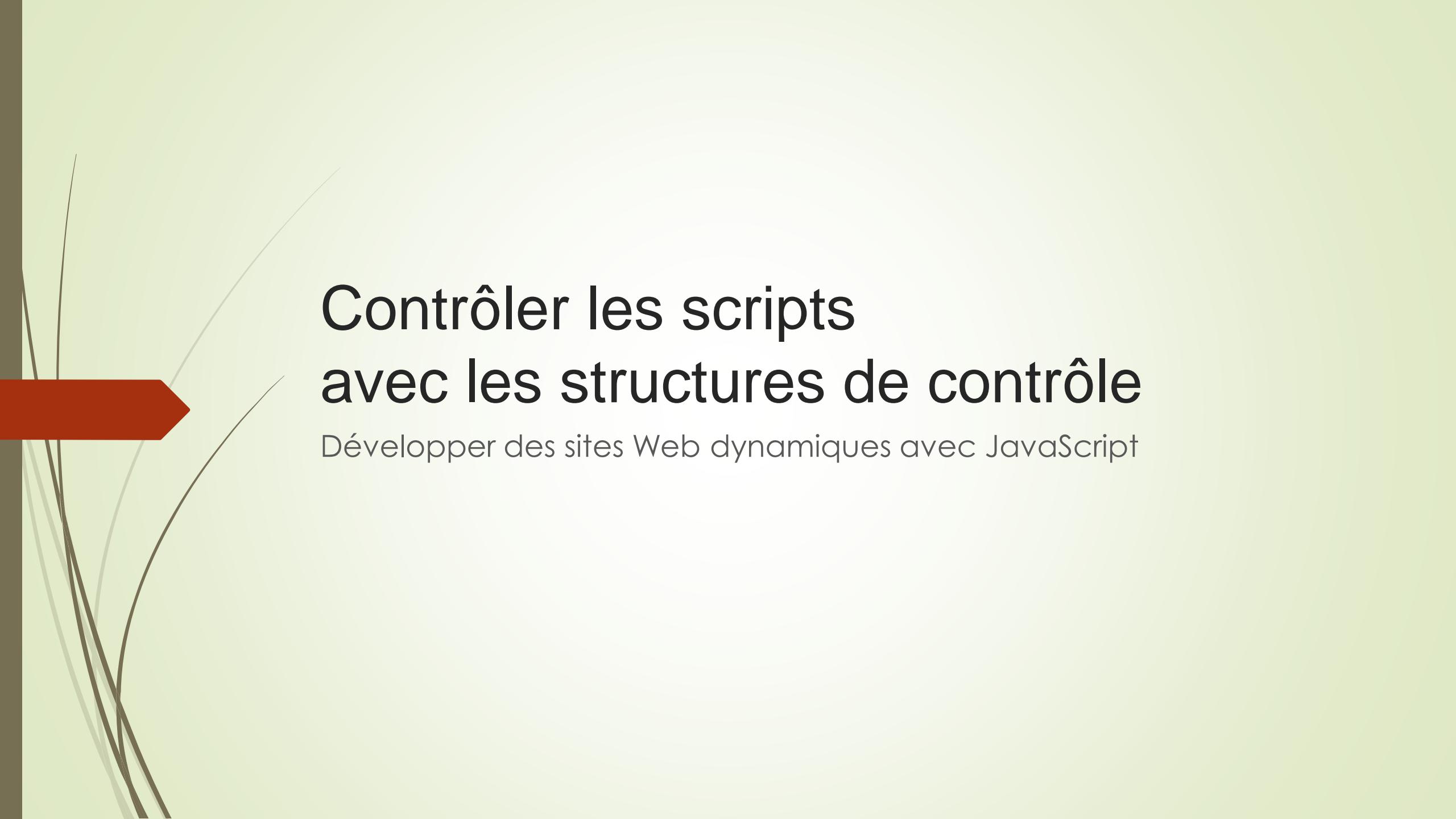
Opérateur	Nom	Rôle	Exemple	Résultat
&&	Et	Permet de concaténer plusieurs conditions en vue d'un test. Renvoie true si les deux conditions sont réunies.	5 est supérieur à 4 et inférieur à 6	true
	Ou	Vérifie que l'une ou l'autre des conditions est remplie. Renvoie true si c'est le cas.	5 est supérieur à 4 ou inférieur à 3	true
!	Non	Vérifie que la condition du test n'est pas respectée. Renvoie true si c'est le cas.	5 est supérieur à 6	true

Opérateurs de concaténation

Opérateur	Nom	Rôle	Exemple	Résultat
+	Plus	Concatène les chaînes de caractères.	"Bon" + "jour"	"Bonjour"

Il existe un seul opérateur de concaténation, c'est le symbole plus "+" qui permet de souder des chaînes de caractères entre elles.

```
// Concaténation de chaînes  
  
"coucou" + " monde" // "coucou monde"  
  
// En additionnant une chaîne à un nombre tout est  
// d'abord converti en une chaîne. Le résultat est  
// parfois surprenant  
  
"3" + 4 + 5; // "345"  
3 + 4 + "5"; // "75"
```



Contrôler les scripts avec les structures de contrôle

Développer des sites Web dynamiques avec JavaScript

if

- ▶ **if** permet de distinguer deux cas de figure
 - ▶ Le test doit être placé entre **parenthèses** et comprend des opérateurs de comparaison et des variables ou constantes
 - ▶ Le test est suivi **d'accolades** entre lesquelles se trouvent autant d'instructions nécessaires pour traiter le test s'il est vrai
 - ▶ Le mot clé **else** peut être ajouté à la suite du premier bloc
 - ▶ Il sera également suivi **d'accolades** entre lesquelles seront placées les instructions pour traiter le test s'il est faux

```
if (test) {  
    Ligne 1 d'instruction ;  
    Ligne 2 d'instruction ;  
} else {  
    Ligne 1 d'instruction ;  
    Ligne 2 d'instruction ;  
}
```

if : exemple

```
<script>
avis=confirm("Aimez-vous le JavaScript ? ");
suite=confirm("Voulez-vous poursuivre ? ");
if (suite==true && avis==true) {
    alert("J'en suis heureux ");
} else {
    alert("C'est dommage");
}
</script>
```



L'opérateur conditionnel ternaire

- ▶ Pour effectuer des tests, il est également possible d'utiliser **l'opérateur conditionnel ternaire**
`(test) ? instruction si vrai : instruction si faux ;`
- ▶ Même si la syntaxe est **plus concise**, la relecture n'est pas facilitée par cette structure de contrôle
- ▶ C'est pour cette raison que son utilisation est **peu fréquente** et qu'elle est rarement utilisée dans les scripts

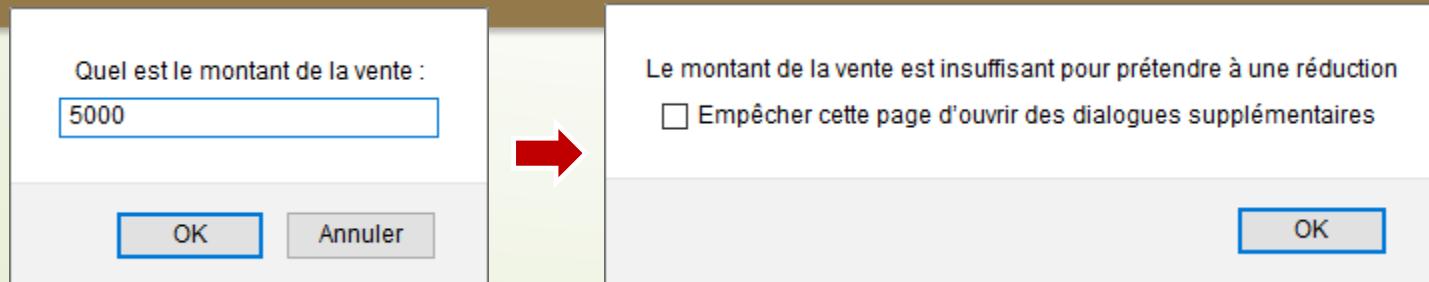
else if

- ▶ L'instruction **else if** est utilisée en complément de **if** dans le cas où le nombre de situations est supérieur à deux
- ▶ On parle alors de concaténation de tests

```
if (test) {  
    Ligne 1 d'instruction ;  
    Ligne 2 d'instruction ;  
} else if (test) {  
    Ligne 1 d'instruction ;  
    Ligne 2 d'instruction ; }  
} else {  
    Ligne 1 d'instruction ;  
    Ligne 2 d'instruction ;  
}
```

else if : exemple

```
<script language="javascript">
commande = prompt("Quel est le montant de la vente :",
"Saisissez ici le montant de la commande");
if (commande<10000) {
    alert("Le montant de la vente est insuffisant pour
    prétendre à une réduction");
}
else if(commande<25000) {
    reduction=commande*0.05;
} else {
    reduction=commande*0.1;
}
alert("Le montant de la réduction pour une commande de : " +
commande + " euros est de : " + reduction + " euros");
</script>
```



switch

- ▶ l'instruction **switch** permet de gérer plusieurs cas de figure
 - ▶ Son utilisation s'avère plus facile qu'une concaténation de **if**
 - ▶ Les opérateurs de comparaison inférieur et supérieur sont interdits
 - ▶ Débute avec **switch** suivi de la variable à tester entre parenthèses
 - ▶ Entre accolades et pour chaque valeur possible de la variable, on indique le mot clé **case** suivi de la valeur de la variable et de deux points
 - ▶ Pour terminer le traitement de chaque cas on utilise le mot clé **break**

```
switch (variable de test) {  
    case " premier cas " :  
        instructions  
        break ;  
    case " deuxième cas " :  
        instructions  
        break ;  
}
```

switch : exemple

```
<script>
reponse=prompt("Javascript est un langage : A.non typé
B.Faiblement typé C.Typé", "Saisissez ici la lettre
correspondant à votre réponse");
switch (reponse) {
    case "A":
        alert("Pas tout-à-fait, ré-essayez");
        break;
    case "B":
        alert("Bravo ! C'est exact");
        break;
    case "C":
        alert("Pas tout-à-fait, ré-essayez");
        break;
    default:
        alert("Votre réponse ne correspond pas aux propositions");
        break;
}
</script>
```

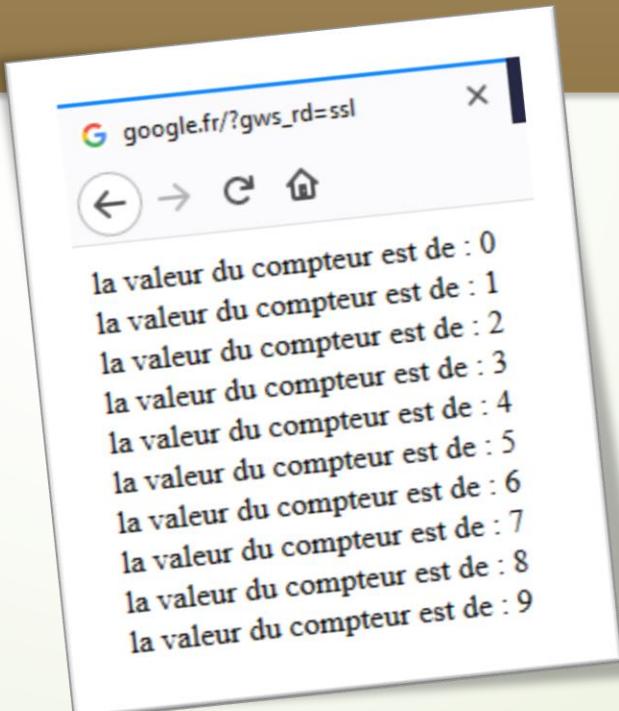
for

- ▶ En Javascript, il en existe deux types de **boucles** permettant de réaliser des blocs d'instructions à plusieurs reprises ou **itératives** :
 - ▶ les boucles avec **for**
 - ▶ et les boucles avec **while**
- ▶ La boucle **for** nécessite l'utilisation d'une valeur initiale de compteur, d'un test et d'un facteur de progression
- ▶ La valeur à atteindre du compteur doit être connue

```
for (valeur initiale du compteur ; test de répétition ;  
facteur de progression) {  
    instructions à répéter  
}
```

for : exemple

```
<script>
for (compteur=0; compteur<10;compteur++) {
    document.write("la valeur du compteur est de : " +
    compteur+"<BR>") ;
</script>
```



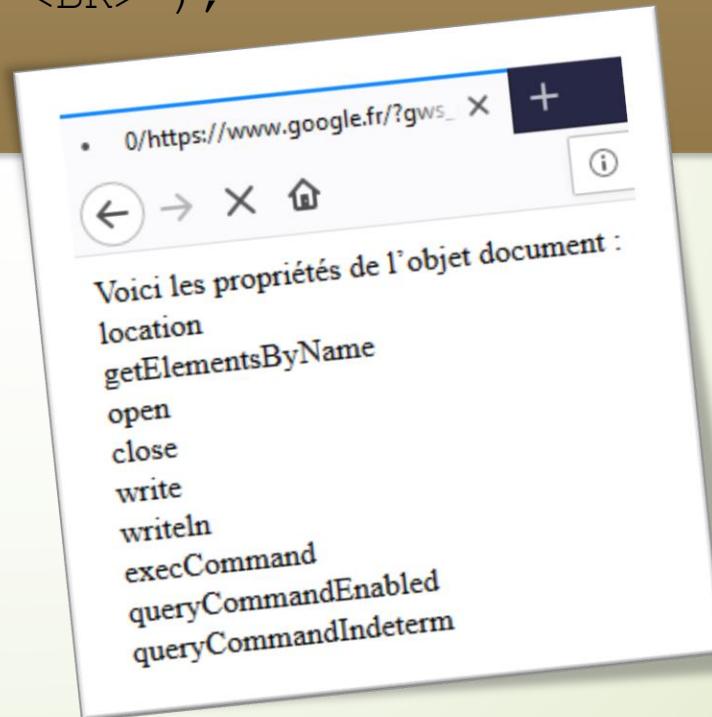
for...in

- ▶ Ce type de boucle est uniquement destiné à manipuler un objet en **itérant** sur ses propriétés énumérables
- ▶ Une propriété est **énumérable** si elle est créée grâce à une affectation simple ou grâce à un initialisateur de propriété
- ▶ Pour chaque propriété obtenue, on exécute une instruction (ou plusieurs grâce à un bloc d'instructions)

```
for (var nompropriété in nomdelobjet) {  
    instructions ;  
}
```

for...in : exemple

```
<script>  
document.write("Voici les propriétés de l'objet  
document : "+<BR>");  
for (var proprietee in document) {  
    document.write(proprietee+<BR>");  
}  
</script>
```



while / do while

- ▶ Les boucles avec **while** permettent d'exécuter un bloc d'instructions tant qu'une condition est vraie
- ▶ Avec **while**, les instructions figurant entre accolades seront exécutées tant que le test est vérifié :

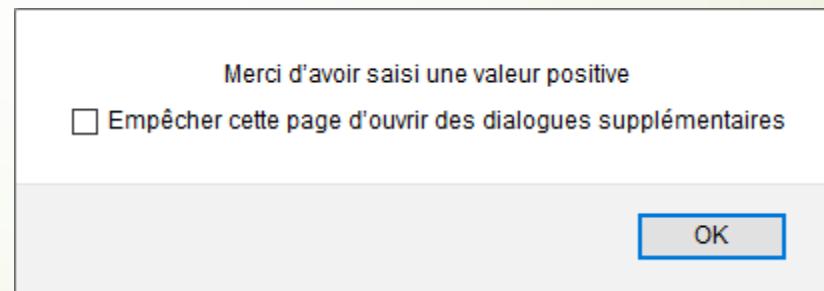
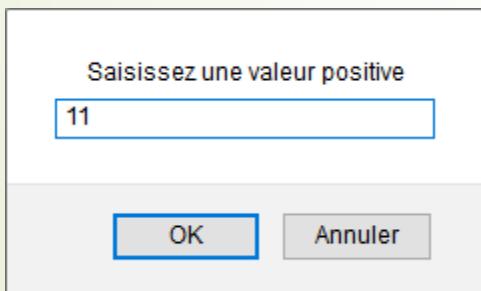
```
while (test) {  
    instructions à répéter  
}
```

- ▶ **do while** permet également la répétition d'instructions à ceci près que le test suit le bloc d'instructions au lieu de le précéder :

```
do {  
    instructions à répéter  
} while (test);
```

while / do while : exemple

```
<script language="javascript">
reponse=-1;
while (reponse<0) {
    reponse=prompt("Saisissez une valeur positive",
    "Saisissez ici votre valeur");
}
alert(" Merci d'avoir saisi une valeur positive " );
</script>
```



break

- ▶ L'instruction **break** permet également de renvoyer le déroulement du script vers une étiquette (identifiant suivi de deux points)

```
boucle:  
for (i=0;i<10;i++) {  
    alert("La valeur de i est égale à : "+i);  
    for (j=0;j<10;j++) {  
        alert("La valeur de j est égale à : "+j);  
        if (j==3) {  
            break boucle; // renvoie à l'étiquette boucle  
                        // donc en dehors de la boucle  
                        // correspondant à j et à i  
        }  
    }  
}
```

- ▶ Enfin **break** ne peut être utilisée qu'à l'intérieur de boucles imbriquées avec **for** ou **switch**

continue

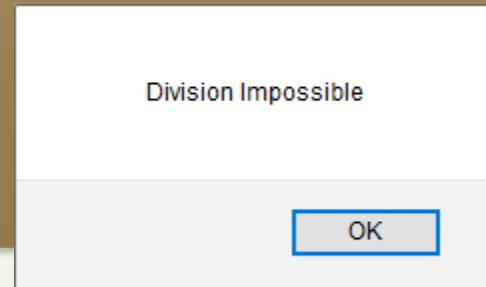
- ▶ L'instruction **continue** permet de ne pas prendre en compte certaines valeurs qui seraient fausses dans le test
- ▶ Elle assure ainsi la poursuite de la boucle

```
<script language="javascript">
for (j=0;j<5;j++) {
    if (j==3) {
        continue; // saute l'instruction suivante
    }
    alert("La valeur de j est égale à : "+j);
}
</script>
```

La structure try...catch

- **Structure de contrôle** permettant de gérer les erreurs renvoyées par le navigateur lors de l'exécution de script
 - ▶ Cette structure permet d'exécuter les instructions placées entre les accolades et correspondant au mot clé **try** : c'est le bloc d'essai
 - ▶ Dans le cas où une erreur est détectée par le navigateur, les instructions situées entre les accolades correspondant au mot clé **catch** sont exécutées

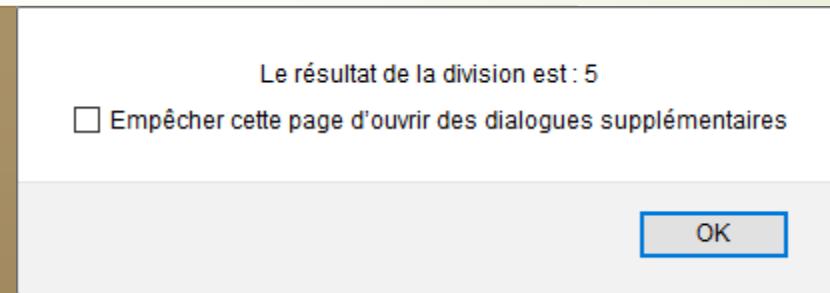
```
<script>
dividende=10;
try {
    resultat=dividende/diviseur;
}
catch(exception) {
    alert("Division Impossible");
}
</script>
```



La structure try...finally

- ▶ Dans le cas où aucune exception n'est détectée, on peut ajouter un bloc final dans lequel on inclut les instructions à effectuer dans un bloc délimité par le mot clé **finally**

```
dividende=10;  
try {  
    resultat=dividende/diviseur;  
}  
catch(diviseur){  
    alert("Division Impossible"); // erreur est détectée  
}  
finally{  
    diviseur=2;  
    resultat=dividende/diviseur;  
    alert("Le résultat de la division est : "+resultat);  
}
```



- ▶ Pour plus d'info :

<https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Statements/try...catch>

Les fonctions

Développer des sites Web dynamiques avec JavaScript

Rôle des fonctions

- ▶ Les **fonctions** font partie des éléments fondamentaux en JavaScript, avec les méthodes :
 - ▶ les **méthodes** s'appliquent à un objet particulier
 - ▶ les **fonctions** sont totalement détachées de cette notion
- ▶ On distingue deux types de fonction :
 - ▶ fonctions **prédéfinies**, déjà intégrées à JavaScript, qui s'emploient sans déclaration préalable
 - ▶ fonctions **définies par l'utilisateur** qu'il faut impérativement déclarer avant de pouvoir les appeler
- ▶ Le concept de fonction est la base de la programmation objet en JavaScript qui présente deux avantages :
 - ▶ regroupement d'instructions dans des **blocs nommés**
 - ▶ exécution des blocs via la **programmation évènementielle**

Rôle des fonctions

Fonctions prédéfinies	Rôle
encodeURI()	Encode un Uniform Resource Identifier (URI) en remplaçant chaque exemplaire de certains caractères par une, deux, trois ou quatre séquences d'échappement représentant le caractère encodé en UTF-8
eval()	Exécute le code JavaScript placé entre les parenthèses
isFinite()	Teste la valeur placée entre les parenthèses pour savoir si elle correspond ou non à un nombre fini
isNaN()	Teste la valeur placée entre les parenthèses pour savoir si elle n'est pas numérique
Number()	Renvoie le résultat d'une conversion en chiffres de la variable placée entre les parenthèses
String()	Renvoie le résultat d'une conversion en texte de la variable placée entre les parenthèses

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects

Rôle des fonctions : exemple

```
document.write(encodeURI('https://mozilla.org/?x=шеллы'));  
//https://mozilla.org/?x=%D1%88%D0%B5%D0%BB%D0%BB%D1%8B  
document.write(eval("Math.pow(3+2, 2)"));  
// 25  
document.write(isFinite(Math.log(0)));  
// false  
document.write isNaN("abcd"));  
// true  
document.write("12.34"+2);  
// 12.342  
document.write(parseFloat("12.34") + 2);  
// 14.34
```

La déclaration des fonctions

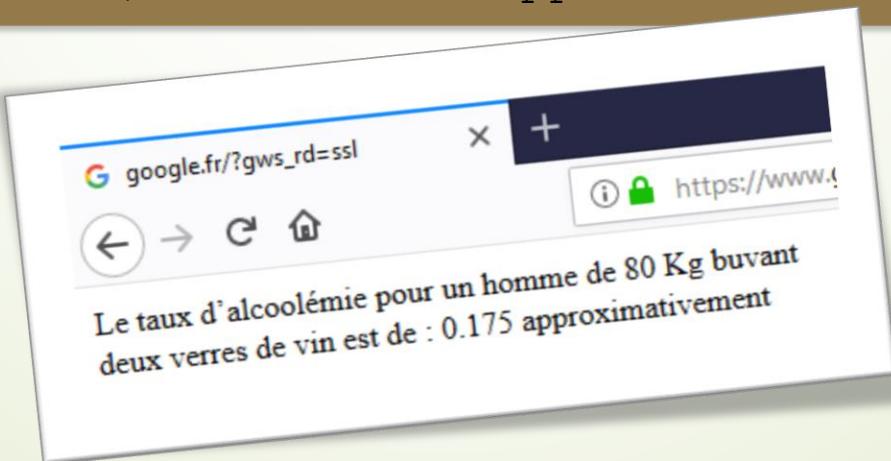
- ▶ Pour déclarer une fonction on utilise le mot clé **function** suivi de :
 - ▶ son **nom** respectant les mêmes règles de nommage que celles des variables
 - ▶ une paire de **parenthèses** pour passer des arguments non typés et séparés par une virgule (255 max.)

```
function maFonction(argument1, argument2) {  
    instruction1;  
    instruction2;  
}
```

- ▶ Une fois la fonction déclarée et créée, on peut l'utiliser en l'appelant et en lui fournissant les arguments nécessaires
- ▶ L'instruction **return** permet de retourner le résultat d'une fonction, en son absence la fonction renvoie **undefined**

La déclaration des fonctions : exemple

```
function tauxAlcool(poids,alcool) {  
    coeff=0.7;  
    resultat=alcool/poids*coeff;  
    return resultat;  
}  
poids=80;  
alcool=2*10;  
document.write("Le taux d'alcoolémie pour un homme de  
80 Kg buvant deux verres de vin est de : " +  
tauxAlcool(poids,alcool) + " approximativement");
```



L'objet arguments

- ▶ L'objet **arguments** correspond aux arguments passés à une fonction
 - ▶ Il contient une entrée pour chaque argument passé à la fonction
 - ▶ L'indice de la première entrée commence à 0
- ▶ L'objet **arguments** est similaire à un tableau, mais il n'en a pas les propriétés, exceptée la propriété **length**

```
function maFonction(a, b, c) {  
    console.log(arguments.length);  
    console.log(arguments[0]);  
    console.log(arguments[1]);  
    console.log(arguments[2]);  
}  
  
maFonction(5, 7, 11); // renvoie 3, 5, 7 et 11
```

L'objet arguments

- ▶ Pour définir une valeur par défaut à un paramètre, il faut tester s'il vaut **undefined** et lui affecter une valeur choisie le cas échéant

```
function multiplier(a, b) {  
    var b = (typeof b !== 'undefined') ? b : 1;  
    return a * b;  
}
```

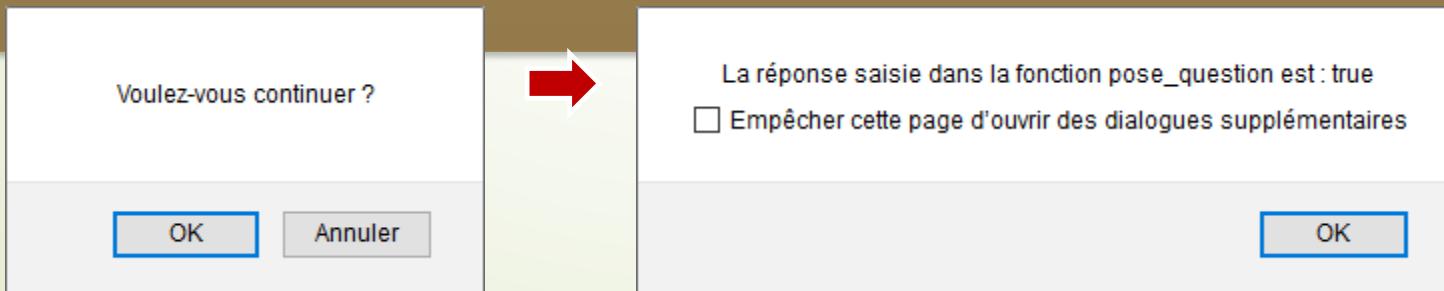
- ▶ Grâce aux paramètres par défaut qui existent depuis ECMAScript 2015 (ES6), on peut se passer de cette vérification et alléger le code de la fonction

```
function multiplier(a, b = 1) {  
    return a * b;  
}
```

Utiliser plusieurs fonctions et transférer des données entre elles

- On peut transférer des valeurs entre deux fonctions d'une manière relativement simple : appeler la 2^{nde} fonction à partir de la 1^{ère}

```
function pose_question() {  
    var reponse = confirm("Voulez-vous continuer ?");  
    test_reponse(reponse)  
}  
  
function test_reponse(reponse) {  
    alert("Nous sommes dans la fonction test_reponse");  
    alert("La réponse saisie dans la fonction  
pose_question est : " + reponse);  
}
```



L'instruction return

- Une fonction peut également être **affectée à une variable** : Il suffit de déclarer la variable en lui affectant la fonction par le signe égal

```
<script>
var total = function(chiffre1, chiffre2) {
    addition = chiffre1 + chiffre2;
    return addition;
}
result = total(10, 20);
alert(" Le résultat de l'addition de la fonction total
est : " + result);
</script>
```

Le résultat de l'addition de la fonction total est : 30

OK

IIFE

- ▶ **IIFE** (Immediately Invoked Function Expression pour Expression de Fonction Invoquée Immédiatement) est une fonction JavaScript qui est exécutée dès qu'elle est définie
- ▶ Ce modèle de conception, également appelé "**Fonction anonyme auto-exécutable**" contient deux parties principales :
 - ▶ La première est la fonction anonyme
 - ▶ La deuxième partie crée la fonction immédiatement exécutable ()

```
var resultat = (function () {  
    var name = "Lesly";  
    return name;  
})();  
// Crée immédiatement la sortie:  
resultat; // "Lesly"
```

Les closures

- ▶ JavaScript supporte les **closures** (ou **fermetures** en français)
 - ▶ Cela signifie qu'une fonction A peut comporter une autre fonction B qui, elle-même, fait référence à la fonction A

```
function externe(parametre) {  
    var variable1 = parametre;  
    function interne() {  
        var variable2 = 10;  
        resultat = variable1 + variable2;  
        alert("L'addition des deux variables provenant de  
        deux fonctions différentes est : " + resultat);  
    }  
    return interne;  
}  
  
var affiche = externe(1);  
affiche();
```

L'addition des deux variables provenant de deux fonctions différentes est : 11

OK

Utiliser une fonction pour créer un objet

- ▶ Deux façons de **construire un objet** en JavaScript :
 - ▶ directement en initialisant un objet de façon littérale
 - ▶ ou par l'intermédiaire d'un **prototype** (plus courante et plus simple)
- ▶ On crée une fonction qui sert de **constructeur** au nouvel objet et de déclaration des différentes propriétés

```
function MonObjet(propriété1, propriété2, propriété3) {  
    this.propriété1 = valeurpropriété1 ;  
    this.propriété1 = valeurpropriété2 ;  
    this.propriété1 = valeurpropriété3 ;  
}
```

- ▶ On réutilise ensuite l'objet à l'aide des mots clés var et new :

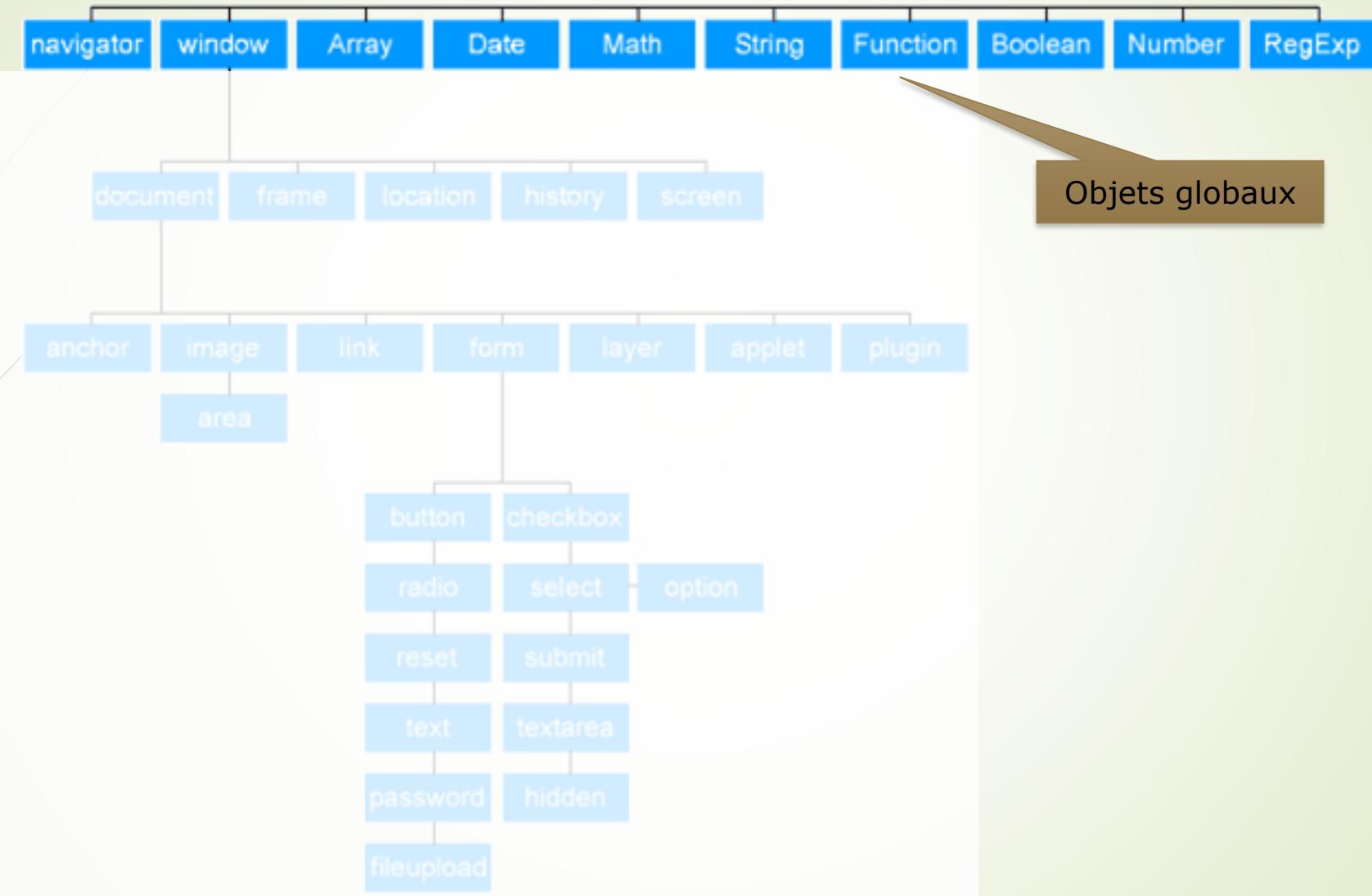
```
var obj=new MonObjet("valeur1","valeur2","valeur3") ;
```



Les objets natifs standards JavaScript

Développer des sites Web dynamiques avec JavaScript

Les objets JavaScript



L'objet Array

- L'objet **Array** représente une variable dans laquelle sont stockées des informations identifiées par un n° d'indice débutant par 0
 - ▶ Le tableau permet de stocker l'information et de la mettre à disposition pour la manipuler (ajouter, supprimer, trier)
 - ▶ Il ne peut pas recevoir plus de 256 valeurs qui peuvent être de type différent (texte, date, numérique, tableau, objet)
 - ▶ Pour l'utiliser, on doit d'abord le déclarer :

```
var tab1 = new Array()  
var tab2 = new Array(5);  
var tab3 = new Array("Ryan", 15, 1.35);  
var tab4 = ["Eve", 72, 1.55];
```

- L'accès aux valeurs stockées se fait par leur numéro d'indice :

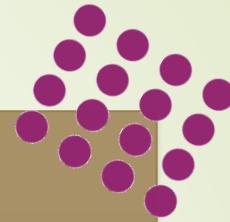
```
document.write(tab4[1]); // renvoie 72
```

L'objet Array

- ▶ Propriétés
 - ▶ **length** : nombre d'éléments du tableau
- ▶ Méthodes
 - ▶ **concat()** : concatène plusieurs tableaux en un
 - ▶ **join()** : crée une chaîne de caractères à partir des éléments du tableau
 - ▶ **pop()** : supprime le dernier élément d'un tableau (file)
 - ▶ **push()** : ajoute des éléments en fin d'un tableau (file)
 - ▶ **reverse()** : inverse l'ordre des éléments d'un tableau
 - ▶ **shift()** : supprime le premier élément d'un tableau (pile)
 - ▶ **unshift()** : ajoute un élément au début d'un tableau (pile)
 - ▶ **slice()** : retourne ou ajoute une partie d'un tableau
 - ▶ **splice()** : ajoute, supprime ou remplace les éléments d'un tableau

L'objet Array : exemple

```
var semainier=["lundi", "mardi", ... , "dimanche"];
tableau=semainier.join(" et ");
document.write(tableau);
// lundi et mardi ... et dimanche
semainier.pop();
document.write(semainier);
// lundi,mardi,mercredi,jeudi,vendredi,samedi
semainier.shift();
document.write(semainier);
// mardi,mercredi,jeudi,vendredi,samedi
var envers=semainier.reverse();
document.write(envers);
// samedi,vendredi,jeudi,mercredi,mardi
travail=semainier.slice(0,3);
document.write(travail);
// samedi,vendredi,jeudi
croissant=semainier.sort();
document.write(croissant);
// jeudi,mardi,mercredi,samedi,vendredi
```



Les tableaux multidimensionnels

- ▶ Quand les données sont nombreuses, on peut utiliser les tableaux à plusieurs entrées, aussi appelés **multidimensionnels**
- ▶ La construction de ce type de tableau nécessite la déclaration des tableaux imbriqués
- ▶ La syntaxe suivante crée un tableau de 2 lignes et 2 colonnes :

```
var tableau = new Array(2);
var tableau[0] = new Array(2);
var tableau[1] = new Array(2);
tableau[0][0] = "valeur1";
tableau[0][1] = "valeur2";
tableau[1][0] = "valeur1";
tableau[1][1] = "valeur2";
```

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Array

L'objet Date

- ▶ L'objet **Date** permet de gérer le calcul du temps jusqu'à une précision de la milliseconde
- ▶ Les dates sont représentées par un nombre compris entre -100 000 000 et +100 000 000 avec comme référence le 1^{er} janvier 1970
- ▶ L'objet **Date** n'a pas de propriété, par contre il possède trois types de méthode pour lire, écrire ou convertir
- ▶ Pour manipuler l'objet **Date**, il faut en créer une instance dans une variable
- ▶ Il existe en plus un système de minuterie permettant de gérer les intervalles de temps

L'objet Date : exemple



```
var today = new Date()
document.write(today);
// Sat Feb 03 2018 17:24:56 GMT+0100 (Paris, Madrid)
var birthday = new Date("July 12, 1998 01:02:03")
document.write(birthday);
// Sun Jul 12 1998 01:02:03 GMT+0200
birthday = new Date(98,6,12)
document.write(birthday);
// Sun Jul 12 1998 00:00:00 GMT+0200
birthday = new Date(98,6,12,1,2,3)
document.write(birthday);
// Sun Jul 12 1998 01:02:03 GMT+0200
document.write(birthday.getDay());
// 0
document.write(birthday.getDate());
// 12
birthday.setDate(25);
document.write(birthday);
// Sat Jul 25 1998 01:02:03 GMT+0200
```

L'objet Date

► Méthodes

- ▶ **getDay()** : n° du jour de la semaine (0=dimanche à 6=samedi)
- ▶ **getDate()**, **setDate()** : n° du jour du mois entre 1 et 31
- ▶ **getMonth()**, **setMonth()** : n° du mois courant (0=janvier à 11=décembre)
- ▶ **getHours()**, **setHours()** : heure courante (0=minuit)
- ▶ **getMinutes()**, **setMinutes()** : nombre de minutes de l'heure courante
- ▶ **getTime()**, **setTime()** : nombre de ms écoulées depuis le 1^{er} janvier 1970
- ▶ **getFullYear()**, **setFullYear()** : année en cours sur 4 chiffres
- ▶ **getYear()**, **setYear()** : année en cours sur 2 chiffres
- ▶ **parse()** : convertit une date en nombre de ms depuis le 1er janvier 1970
- ▶ **toString()**, **toLocaleString()** : convertit l'objet Date en une chaîne

L'objet Date : exemple

```
birthday = new Date(98, 6, 12, 1, 2, 3)
document.write(birthday.getMonth());
// 6
birthday.setMonth(10);
document.write(birthday);
// Thu Nov 12 1998 01:02:03 GMT+0100 (Paris, Madrid)
document.write(birthday.getFullYear());
// 98
birthday.setYear(96);
document.write(birthday);
// Tue Nov 12 1996 01:02:03 GMT+0100 (Paris, Madrid)
document.write(birthday.getFullYear());
// 1996
birthday.setFullYear(2018);
document.write(birthday);
// Mon Nov 12 2018 01:02:03 GMT+0100 (Paris, Madrid)
document.write(Date.parse("Jul 12,1998"))
// 900194400000
```



Les minuteries

- Il existe quatre méthodes particulières qui permettent de gérer une minuterie :
 - **setTimeout()** : exécute une fonction après un délai déterminé (en ms)
 - **clearTimeout()** : stoppe la minuterie lancée par setTimeout()

```
// Compatibilité accrue grâce aux fonctions anonymes
window.setTimeout(function() {
    generateOutput(true);
}, 1000);
```

- **setInterval()** : appelle une fonction de manière répétée, avec un certain délai fixé entre chaque appel (en ms)
- **clearInterval()** : stoppe la minuterie lancée par setInterval()

<https://developer.mozilla.org/fr/docs/Web/API/setTimeout>

<https://developer.mozilla.org/en-US/docs/Web/API/clearTimeout>

<https://developer.mozilla.org/en-US/docs/Web/API/setInterval>

<https://developer.mozilla.org/fr/docs/Web/API/clearInterval>

Les minuteries : exemple



```
var intervalID;

function changeCouleur() {
    intervalID = setInterval(flashText, 1000);
}

function flashText() {
    var elem = document.getElementById("ma_boite");
    if (elem.style.color == 'red') {
        elem.style.color = 'blue';
    } else {
        elem.style.color = 'red';
    }
}

function stopTextColor() {
    clearInterval(intervalID);
}
```

L'objet Math

- ▶ L'objet **Math** (sans constructeur) propose des méthodes et propriétés statiques permettant l'utilisation de constantes et fonctions mathématiques
- ▶ Propriétés
 - ▶ **E** : Nombre d'Euler, la base des logarithmes naturels = 2,718
 - ▶ **LN2** : Logarithme naturel de 2 = 0,693
 - ▶ **LN10** : Logarithme naturel de 10 = 2,302
 - ▶ **LOG2E** : Logarithme de base 2 de E = 1,442
 - ▶ **LOG10E** : Logarithme de base 10 de E = 0,434
 - ▶ **PI** : Quotient de la circonférence d'un cercle par son diamètre = 3,14159
 - ▶ **SQRT1_2** : Racine carrée de 1/2 = 0,707
 - ▶ **SQRT2** : Racine carrée de 2 = 1,414

L'objet Math

► Méthodes

- ▶ **abs()** : valeur positive absolue du nombre passé en argument
- ▶ **acos(), asin(), atan()** : angle dont l'argument représente le cosinus/sinus/tangente
- ▶ **cos(), sin(), tan()** : cosinus/sinus/tangente d'un nombre passé en argument
- ▶ **ceil(), floor()** : entier supérieur/inférieur d'une valeur passée en argument
- ▶ **log()** : logarithme népérien d'un nombre passé en argument
- ▶ **max(), min()** : nombre le plus grand/petit dans la série en argument
- ▶ **pow()** : nombre élevé à une puissance passée en argument
- ▶ **random()** : renvoie un nombre aléatoire compris entre 0 et 1
- ▶ **sqrt()** : racine carrée d'un nombre passé en argument

L'objet Math : exemple

$\Sigma \sqrt{\Pi}$

```
document.write(115.04+15) ;
// 130.04000000000002 - Surprenant !
document.write(Math.PI) ;
// 3.141592653589793
document.write(Math.abs(-12.34)) ;
// 12.34
document.write(Math.floor(12.54)) ;
// 12
document.write(Math.round(12.54)) ;
// 13
document.write(Math.ceil(12.54)) ;
// 13
document.write(Math.random()) ;
// 0.394555831655689
```

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Math

L'objet String

- ▶ L'objet **String**, à ne pas confondre avec le type du même nom, permet de manipuler les chaînes de caractères
- ▶ Propriétés
 - ▶ **length** : longueur d'une chaîne de caractères
- ▶ Méthodes
 - ▶ **charAt()** : pour accéder à un seul caractère dans une chaîne
 - ▶ **charCodeAt()** : valeur en décimal du caractère indiqué en argument
 - ▶ **concat()** : concatène plusieurs chaînes de caractères
 - ▶ **fromCharCode()** : chaîne à partir d'une suite de caractères Unicode
 - ▶ **indexOf()** : position à partir du début du caractère passé en argument
 - ▶ **lastIndexOf()** : position à partir de la fin du caractère passé en argument
 - ▶ **match()** : recherche une expression régulière dans une chaîne

L'objet String : exemple

ABC

```
var s = "Bon anniversaire Lesly";
document.write(s.charAt(2));
// n
document.write(s.charCodeAt(2));
// 110
document.write(s.concat(" du service IT"));
// Bon anniversaire Lesly du service IT
document.write(String.fromCharCode(49, 50));
// 12
document.write(s.indexOf("Lesly"));
// 17
document.write(s.lastIndexOf("a"));
// 12
document.write(s.match(/Lesly$/));
// Lesly (null si non trouvé)
```

L'objet String

► Méthodes (suite)

- ▶ **replace()** : remplace une expression régulière dans une chaîne
- ▶ **search()** : effectue une recherche dans une chaîne de caractères
- ▶ **slice()** : extrait une sous-chaîne via un caractère de début et de fin
- ▶ **split()** : découpe une chaîne en fonction d'un séparateur passé en argument
- ▶ **substr()** : renvoie une sous-chaîne via une position et une longueur
- ▶ **substring()** : renvoie une sous-chaîne via une position
- ▶ **toLowerCase()** : convertit en minuscules
- ▶ **toUpperCase()** : convertit en majuscules
- ▶ **toString()** : convertit un objet en chaîne de caractères

L'objet String : exemple

ABC

```
var s = "Bon anniversaire Lesly" ;
document.write(s.replace(/i/g, 'I')) ;
// Bon annIversaIre Lesly
document.write(s.search(/n{2}/i)) ;
// 5
document.write(s.slice(17)) ;
// Lesly
document.write(s.split(" ")) ;
// Bon,anniversaire,Lesly
document.write(s.substr(4, 12)) ;
// anniversaire
document.write(s.substring(4, 16)) ;
// anniversaire
document.write(s.toUpperCase()+s.toLowerCase()) ;
// BON ANNIVERSAIRE LESLYbon anniversaire lesly
```

https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/String

L'objet RegExp

- ▶ Cet objet permet de manipuler les **expressions régulières**, présentes dans la plupart des langages de programmation
 - ▶ Il permet de chercher, remplacer, découper des chaînes de caractères
 - ▶ On rédige un masque (pattern en anglais) qu'il est possible d'appliquer à une chaîne de caractères pour la filtrer ou la gérer
 - ▶ Pour créer une variable on peut utiliser l'une des 2 syntaxes suivantes :

```
var monexpression = new RegExp(motif, option);  
var monexpression = /motif?option;
```
- ▶ Où **motif** représente le masque de recherche et **option** correspond à un commutateur
 - ▶ **g** : Force une recherche globale
 - ▶ **i** : Ne tient pas compte de la casse des caractères
 - ▶ **gi** : Associe les options i et g

L'objet RegExp

- ▶ Les caractères d'ensemble
 - ▶ **[xyz]** : 1 élément de la liste
 - ▶ **[a-z]** : 1 élément de la série min.
 - ▶ **[A-Z]** : 1 élément de la série maj.
 - ▶ **[^xyz]** : 1 élément pas dans la liste
 - ▶ **[^a-z]** : 1 élément pas dans la série
 - ▶ **[0-9]** : 1 élément de la série num.
 - ▶ **\d = [0-9]** : chiffre
 - ▶ **\D = [^0-9]** : non chiffre
- ▶ Les caractères de groupement
 - ▶ **()** : groupe des caractères en sous-motif
- ▶ Les caractères de répétition
 - ▶ ***** : 0 à n fois
 - ▶ **+** : 1 à n fois
 - ▶ **?** : 0 à 1 fois
 - ▶ **{n}** : n fois
 - ▶ **{n,}** : au moins n fois
 - ▶ **{n,m}** : de n à m fois

L'objet RegExp

- ▶ Les caractères de positionnement
 - ▶ **`\^`** : début d'expression
 - ▶ **`\$`** : fin d'expression
 - ▶ **`\b`** : début du mot
 - ▶ **`\B`** : fin du mot
 - ▶ **`(x)`** : trouve x et retient sa position
- ▶ Le caractère de choix
 - ▶ **`x | z`** : caractère peut être x ou z
- ▶ Les caractères spéciaux
 - ▶ **`.`** : tout caractère sauf retour chariot
 - ▶ **`\f`** : saut de page
 - ▶ **`\n`** : retour à la ligne
 - ▶ **`\r`** : retour chariot
 - ▶ **`\s`** : séparateur de mot
 - ▶ **`\S`** : non séparateur de mot
 - ▶ **`\t`** : tabulation
 - ▶ **`\w = [A-Za-z0-9_]`** : caractère alphanumérique

L'objet RegExp

- ▶ Propriétés
 - ▶ **source** : texte du masque
 - ▶ **global** : la recherche doit s'arrêter à la première occurrence trouvée (g)
 - ▶ **ignoreCase** : la casse doit être ignorée (i)
- ▶ Méthodes
 - ▶ **test()** : renvoie une valeur booléenne si le test est vérifié ou non
 - ▶ **exec()** : renvoie la première occurrence trouvée dans la chaîne
 - ▶ **match()** : trouve les occurrences d'une sous-chaîne de caractères

[A-Z] {2} - \d{3} - [A-Z] {2}



L'objet RegExp : exemple

RegEx

```
document.write(/l/.test('Hello')); // true
document.write(/^l/.test('Hello')); // false
document.write(/^h/.test('Hello')); //false
document.write(/^h/i.test('Hello')); // true
document.write(/^Hel.o/.test('Hello')); // true
document.write(/^Hel+o/.test('Hello')); // true
document.write(/^He+llo/.test('Hello')); // true
document.write(/^Hea*ll{o$/.test('Hello')); // true
document.write(/^He(l|o)*$/.test('Hello')); // true
document.write(/^H[leos]+/.test('Hello')); // true
document.write(/^H[^leo]+/.test('Hello')); // false
document.write(/^H[^kyz]+/.test('Hello')); // true
document.write(/^H[a-z]*/.test('Hello')); // true
document.write(/^H[a-z]*$/.test('Hello')); // true
```

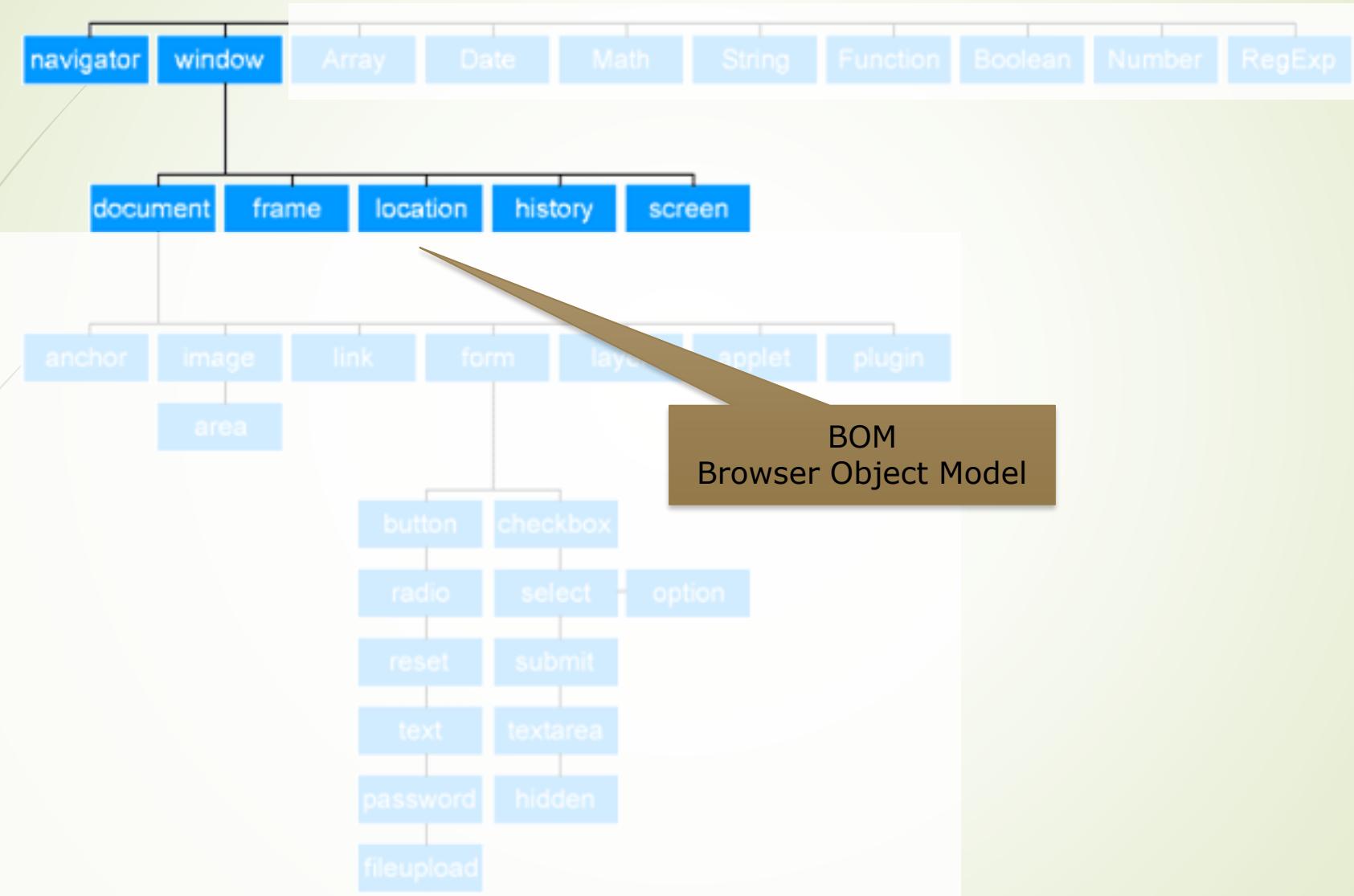
https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/RegExp



Les principaux objets JavaScript en détail

Développer des sites Web dynamiques avec JavaScript

Les objets JavaScript



L'objet Navigator

- ▶ L'objet **Navigator** correspond au navigateur utilisé par l'internaute
 - ▶ Son interprétation est différente selon les navigateurs et leur version
- ▶ Principales propriétés :
 - ▶ **language** : retourne la langue définie dans le navigateur
 - ▶ **geolocation** : retourne un objet utilisé pour localiser l'utilisateur
 - ▶ **product** : retourne le nom du moteur utilisé par le navigateur
 - ▶ **cookieEnabled** : détermine si les cookies sont autorisés ou non
 - ▶ **appName** : retourne le nom du navigateur
 - ▶ **appCodeName** : retourne le nom de code du navigateur
 - ▶ **appVersion** : retourne la version du navigateur utilisée
 - ▶ **online** : détermine si le navigateur est en ligne ou pas
 - ▶ **userAgent** : retourne l'en-tête du fichier user-agent envoyé par le navigateur au serveur

L'objet Navigator : exemple

```
navigateur = navigator.appName;
version = navigator.appVersion;
plateforme = navigator.platform;
cookie = navigator.cookieEnabled;
if (cookie == true) {
    alert("Vous utilisez actuellement "+navigateur+
        "+version+\r comme navigateur Internet, sur une plate-
        forme de type : "+plateforme+" avec les cookies activés");
} else {
    alert("Vous utilisez actuellement "+navigateur+
        "+version+\r comme navigateur Internet, sur une plate-
        forme de type : "+plateforme+"Attention ! les cookies ne
        sont pas activés");
}
```

Vous utilisez actuellement Netscape 5.0 (Windows) comme navigateur Internet, sur une plate-forme de type : Win64 avec les cookies activés

OK

L'objet Window

- ▶ L'objet **Window** est le plus élevé dans la hiérarchie des objets
 - ▶ C'est le parent de tous les objets placés à l'intérieur du modèle objet
 - ▶ Il n'est pas utile de le nommer pour accéder aux objets placés en dessous
- ▶ Principales propriétés :
 - ▶ **status, defaultStatus** : message de la barre d'état
 - ▶ **innerHeight, innerWidth** : hauteur/largeur utilisable d'une fenêtre
 - ▶ **outerHeight, outerWidth** : hauteur/largeur extérieure d'une fenêtre
 - ▶ **pageXOffset, pageYoffset** : position horizontale/verticale de la fenêtre
 - ▶ **name** : nom donné à la fenêtre
 - ▶ **parent** : fenêtre parente
 - ▶ **self, window** : fenêtre courante

L'objet Window

- ▶ Principales méthodes :

- ▶ **alert()**, **confirm()**, **prompt()** : affiche une boîte de dialogue
- ▶ **back()**, **forward()** : une page en arrière/avant dans l'historique
- ▶ **find()** : recherche de texte dans la page active
- ▶ **focus()** : active une fenêtre
- ▶ **moveTo()** : déplace la fenêtre active
- ▶ **open()**, **close()** : ouvre/ferme une fenêtre
- ▶ **print()** : imprime la page active
- ▶ **resizeBy()**, **resizeTo()** : modifie la taille de la fenêtre
- ▶ **setInterval()** : effectue un traitement à intervalle régulier
- ▶ **setTimeout()** : déclenche une minuterie

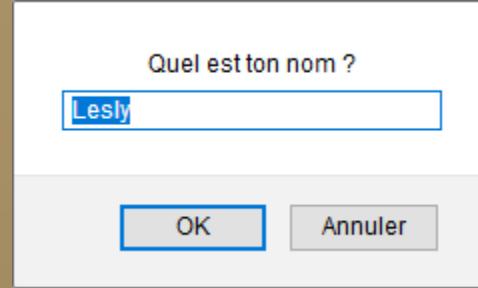
L'objet Window : exemple

```
// Boites de dialogue
alert("Hello world!");
confirm("Etes-vous sûr ?");
prompt("Quel est ton nom ?", "Lesly");

// Manipulation des fenêtres
window.moveBy(10, 20);
window.resizeTo(150, 300);
window.resizeBy(150, 0);
window.moveTo(0, 0);

// Barre d'état
window.defaultStatus = "Ceci est le texte par défaut";
window.status = "Information sur le sujet"

// Intervalles et Chronos
var iTimeoutId = setTimeout("alert('Hello world!')", 1000);
clearTimeout(iTimeoutId);
setInterval("alert('Hello world!')", 1000);
```



Méthode open()

- ▶ Souvent employée pour créer ce que l'on appelle des **popup**, sa syntaxe est la suivante :

```
f = window.open("page", "nom", "param1,param2,param3");
```

- ▶ **f** : nom de l'objet représenté par la nouvelle fenêtre
- ▶ **page** : adresse de la page à afficher
- ▶ **nom** : nom de la nouvelle fenêtre
- ▶ **param1/2/3** : paramètres optionnels (position, taille, etc.)
les paramètres de position/taille sont exprimés numériquement,
les autres utilisent les valeurs (true ou false) ou (yes ou no)

```
f = window.open("https://www.ib-formation.fr/",  
"maFenetre", "height=150, width=300, top=10, left=10,  
resizable=yes, menubar=no, status=no, scrollbars=no");
```

Méthode close()

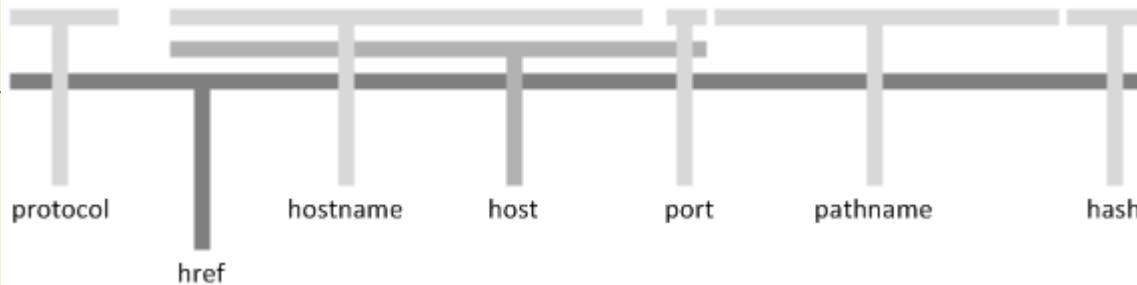
- ▶ Elle permet de fermer les fenêtres ouvertes par la méthode open(), en utilisant le nom de l'objet employé lors de sa création
- ▶ Sa syntaxe est la suivante :

```
function popup() {  
    f=window.open("https://www.ib-formation.fr/","IB",  
    "width=200,height=200,location=no");  
}  
  
function fermerf() {  
    f.close();   
}  
  
function fermeractive() {  
    self.close();   
}
```

L'objet Location

- ▶ Cet objet correspond à l'URL courante que l'on peut interroger ou surtout modifier
- ▶ Propriétés

`http://www.ellipsetours.com:80/Menu/Breakfast#top`



- ▶ Méthodes

- ▶ **reload()** : recharge la page actuelle
- ▶ **replace()** : remplace l'URL de la page actuelle par une autre

L'objet Location : exemple

```
document.write("Voici le nom d'hôte du serveur de la page  
actuelle : " + window.location.hostname+<br>);  
document.write("Voici le chemin de la page actuelle : " +  
    window.location.pathname+<br>);  
document.write("Voici le protocole utilisé pour la page  
actuelle : " + window.location.protocol+<br>);  
document.write("Voici l'URL de la page actuelle : " +  
    window.location.href+<br>);  
window.location.href="https://developer.mozilla.org/fr/";
```

Voici le nom d'hôte du serveur de la page actuelle : developer.mozilla.org
Voici le chemin de la page actuelle : /fr/docs/Web/API/window/location
Voici le protocole utilisé pour la page actuelle : https:
Voici l'URL de la page actuelle : https://developer.mozilla.org/fr/docs/Web/API/window/location

L'objet History

- ▶ L'objet **History** est un sous-objet de **Window** qui correspond à l'historique conservé dans le navigateur
- ▶ Propriété
 - ▶ **length** : nombre de pages visitées pour la fenêtre active
- ▶ Méthodes
 - ▶ **back()** : charge la dernière page visitée et présente dans l'historique
 - ▶ **forward()** : charge la page suivante visitée et présente dans l'historique
 - ▶ **go()** : charge la page liée à un index dans l'historique

<https://developer.mozilla.org/fr/docs/Web/API/Window/history>

L'objet Screen

- ▶ L'objet **Screen** donne accès aux informations concernant l'écran, comme sa taille ou sa résolution par exemple
- ▶ Connaître ces informations permet d'adapter les pages web pour un affichage optimisé pour chaque visiteur
- ▶ Il ne possède pas de méthode et a six propriétés :
 - ▶ **availHeight** , **availWidth** : hauteur/largeur utile de l'écran
 - ▶ **height**, **width** : hauteur/largeur totale d'affichage de l'écran
 - ▶ **colorDepth** : nombre de couleurs que l'écran peut restituer
 - ▶ **pixelDepth** : retourne la résolution de l'écran en bits par pixel

L'objet Screen : exemple

```
var alargeur = screen.availWidth;
var ahauteur = screen.availHeight;
var largeur = screen.width;
var hauteur = screen.height;
var couleurs = screen.colorDepth;

alert("La résolution de votre écran est de : " + largeur + " pixels de largeur et de " + hauteur + " pixels en hauteur avec une palette de couleurs de " + couleurs + " bits");

alert("La surface utile de votre affichage n'est que de " + alargeur + " pixels de largeur et de " + ahauteur + " pixels en hauteur");
```



La résolution de votre écran est de : 1600 pixels de largeur et de 900 pixels en hauteur avec une palette de couleurs de 24 bits

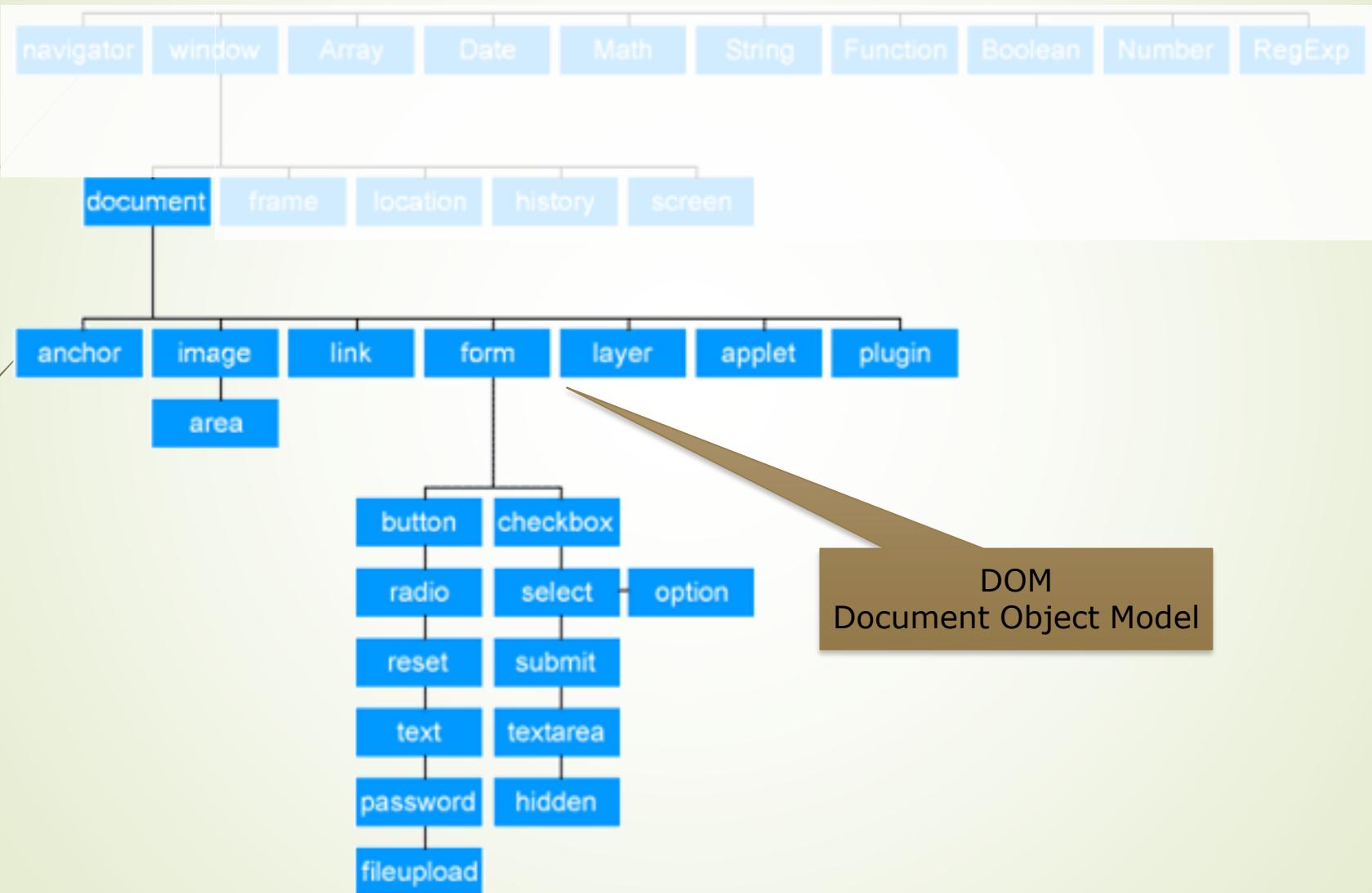
OK



Les objets du document

Développer des sites Web dynamiques avec JavaScript

Les objets JavaScript



L'objet Document

- ▶ L'objet **Document** correspond à la page HTML elle-même
 - ▶ Souvent manipulé en JS, notamment lorsqu'il est fait usage du HTML
 - ▶ Permet d'accéder aux éléments de la page pour en modifier les propriétés ou le contenu
 - ▶ Permet également d'accéder aux cookies et autres stockages locaux
- ▶ Principales propriétés :
 - ▶ **anchor** : tableau contenant toutes les ancrées du document
 - ▶ **cookie** : sous la forme d'une chaîne de caractères
 - ▶ **forms** : tableau contenant tous les formulaires du document
 - ▶ **images** : tableau contenant toutes les images du document
 - ▶ **lastModified** : date de dernière modification du document
 - ▶ **referrer** : URL du document ayant chargé la page active
 - ▶ **title** : titre de la page HTML

L'objet Document

► Principales méthodes :

- **write()** : écriture dans le document actif
- **writeln()** : identique à write() mais ajoute un retour à la ligne

```
function dernieremodif() {  
    datemaj=document.lastModified;  
    var datemodif = new Date(datemaj);  
    var mois=datemodif.getMonth ()+1;  
    var jour=datemodif.getDay();  
    var annee=datemodif.getFullYear();  
    var heure=datemodif.getHours();  
    var minute=datemodif.getMinutes();  
    var secondes=datemodif.getSeconds();  
    document.write("Page modifiée le :  
"+jour+"/"+mois+"/"+annee+" à "+heure+" h "+minute+" mins  
"+secondes+" secondes");  
}
```

Page modifiée le : 3/1/2018 à 15 h 31 mins 10 secondes

L'objet Image

- ▶ L'objet **Image** se trouve sous l'objet document et correspond à la balise HTML
- ▶ Propriétés
 - ▶ **alt** : texte affiché si le navigateur ne prend pas l'image en charge
 - ▶ **complete** : Indique si l'image est définitivement chargée
 - ▶ **fileSize** : taille de l'image en octets
 - ▶ **height/width** : hauteur/largeur de l'image en pixels
 - ▶ **length** : nombre d'images situées dans la page
 - ▶ **src** : adresse source (URL) du fichier de l'image
 - ▶ **title** : texte affiché dans une bulle quand la souris est positionnée dessus

L'objet Image : exemple

```
<html>
<head>
<script>
var limite=40;
var compteur=38;
function banniere() {
compteur++;
if (compteur>=limite) {compteur=38;}
document.images["velo"].src=compteur+".jpg";
return compteur;
}
function permute() {var id=setInterval("banniere()",3000);}
</script>
</head>
<body onload="permute()">

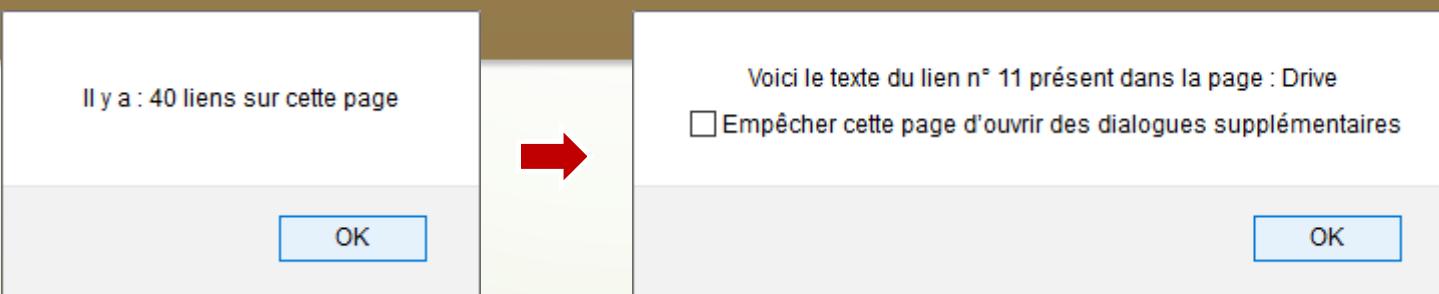
</body>
</html>
```

L'objet Link

- ▶ Cet objet correspond aux liens définis dans un document HTML qu'il est possible de compter, lire et modifier
- ▶ Propriétés
 - ▶ **name** : nom donné à un lien dans la page
 - ▶ **length** : nombre de liens présents dans la page
 - ▶ **target** : fenêtre cible d'un lien (_self, _blank, etc.)
 - ▶ **text** : texte d'un lien présent dans la page
 - ▶ **x/y** : position horizontale/verticale d'un lien dans la page

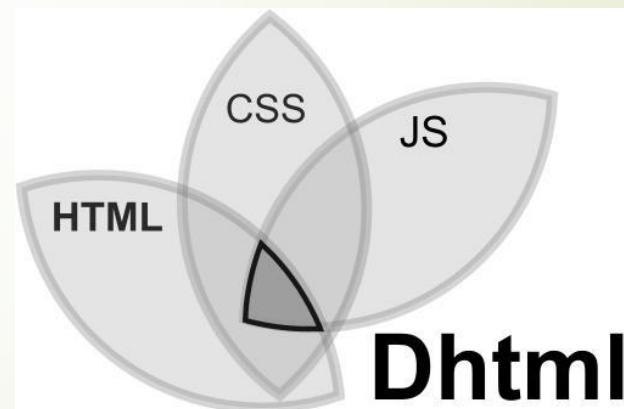
L'objet Link : exemple

```
function exemplelinks() {  
    alert("Il y a : "+document.links.length+" liens sur cette page");  
    for (i=0;i<document.links.length;i++) {  
        alert("Voici le texte du lien n° "+(i+1)+" présent dans la page : "+document.links[i].text);  
        alert("Voici la cible du lien n° "+(i+1)+" présent dans la page : "+document.links[i].target);  
    }  
}
```



Le DHTML (Dynamic HTML)

- ▶ Le **DHTML** est une alternative à Flash
 - ▶ Il combine HTML, CSS, le modèle d'objet DOM et JavaScript pour donner aux pages web un aspect graphique amélioré et plus d'interactivité
 - ▶ Le DHTML n'est donc pas un langage de programmation à part entière mais une combinaison de ces 3 techniques où JavaScript tient une place centrale
 - ▶ Malheureusement le DOM n'a pas été implémenté de la même manière par tous les éditeurs des navigateurs
 - ▶ Il est donc souvent nécessaire de rédiger un script de détection de navigateurs puis d'écrire autant de scripts qu'il y a de versions de navigateurs



Le DOM (Document Object Model)

- ▶ Le **DOM** est une description hiérarchique des éléments composants une page web
 - ▶ Grâce à cette structure hiérarchisée on peut accéder aux objets présents dans un document HTML ou XML
 - ▶ Le DOM est une **API** (Application Programming Interface) totalement indépendante de la plateforme et du langage qui la manipule
 - ▶ Un document se présente sous la forme d'un **arbre** et la notion de **nœud** a été retenue pour représenter les **éléments** qui le compose
- ▶ Il y a plusieurs types de nœuds selon les différents objets HTML :
 - ▶ le type de nœud **ELEMENT_NODE** (pour les éléments HTML ou balises)
 - ▶ le type de nœud **ATTRIBUTE_NODE** (pour les attributs)
 - ▶ le type de nœud **TEXT_NODE** (pour le texte)

Accéder aux éléments

Méthodes	Résultat
getElementById()	Sélectionne un élément en fonction de son identifiant
getElementsByName()	Sélectionne un ou plusieurs éléments en fonction d'un nom passé en argument
getElementsByTagName()	Sélectionne un ou plusieurs éléments en fonction d'un nom de balise passé en argument
getElementsByClassName ()	Sélectionne un ou plusieurs éléments en fonction d'un nom de classe passé en argument
querySelector()	Retourne le premier élément dans le document correspondant au sélecteur
querySelectorAll()	Renvoie la liste des éléments dans le document qui correspondent au groupe de sélecteurs
innerHTML()	Permet de lire ou d'assigner une valeur à un élément

Accéder aux éléments : exemple

```
// Définit le chemin objet par objet (syntaxe pointée)
document.forms[0].login.value

// Recherche l'élément selon son id (unique !)
document.getElementById("login").value

// Recherche l'élément par son nom (non unique !)
document.getElementsByName("login")[0].value

// Recherche l'élément par la balise transmise en argument
document.getElementsByTagName("input")[0].value

// Recherche l'élément à l'aide d'un sélecteur CSS
document.querySelector("#logo").src
document.querySelectorAll(".redBox")[0].src

// Ajoute un paragraphe à une balise div
document.getElementById("myDiv").innerHTML='<p>Hello</p>';
```

Créer et manipuler les éléments

Méthodes/Propriétés	Résultat
createElement()	Crée un élément du type spécifié
createTextNode()	Crée un nouveau nœud de texte
appendChild()	Ajoute un nœud au nœud parent spécifié
removeChild()	Retire un nœud enfant et retourne le nœud retiré
insertBefore()	Insère le nœud spécifié juste avant un élément de référence parmi les enfants du nœud courant
childNodes	Renvoie une collection de nœuds enfants
firstChild, lastChild	Renvoie le premier/dernier nœud enfant du nœud courant
nextSibling, previousSibling	Renvoie le nœud suivant/précédent immédiatement le nœud spécifié
nodeValue	Renvoie ou définit la valeur du nœud courant
attributes	Renvoie une collection des nœuds d'attribut du nœud spécifié

Créer et manipuler les éléments : exemple

```
// Crée un élément <h1>
var h = document.createElement('h1')

// Crée un noeud texte
var t = document.createTextNode('Hello World');

// Ajoute le texte à <h1>
h.appendChild(t);
document.body.appendChild(h);

// retire tous les enfants d'un élément
var element = document.getElementById("haut");
while (element.firstChild) {
    element.removeChild(element.firstChild);
}
```



Les évènements

Développer des sites Web dynamiques avec JavaScript

Les évènements

- ▶ C'est grâce aux évènements que la page devient **interactive**
- ▶ Les **événements** s'appliquent aux objets présents dans la page : contrôles d'un formulaire, images, boutons ou la page elle-même
- ▶ Le déclenchement de l'action est conditionné par un évènement correspondant à un élément HTML de la page
- ▶ Pour cela, on utilise les attributs HTML de **type évènement** et on associe un script contenant le code de l'action que l'on souhaite attacher à l'évènement (gestionnaires d'évènement DOM-0)

```

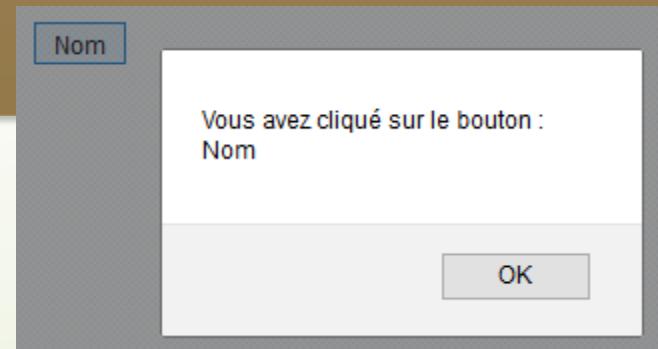
<input type="text" onchange="validateText();" />
<body onload="sayHello();">...</body>
...
window.onload = function(){alert('Hello World!');};
```

Les évènements

Nom de l'évènement	Action pour le déclencher
onclick	Cliquer sur l'élément
ondblclick	Double-cliquer sur l'élément
onmouseover	Faire entrer le curseur sur l'élément
onmouseout	Faire sortir le curseur de l'élément
onmousedown	Maintenir le bouton gauche de la souris sur l'élément
onmouseup	Relâcher le bouton gauche de la souris sur l'élément
onmousemove	Déplacer le curseur sur l'élément
onkeydown	Maintenir une touche de clavier sur l'élément
onkeyup	Relâcher une touche de clavier sur l'élément
onkeypress	Taper une touche de clavier sur l'élément
onfocus	"Cibler" l'élément
onblur	Annuler le "ciblage" de l'élément
onchange	Changer la valeur d'un élément spécifique aux formulaires

Les évènements : exemple

```
<html>
<head>
<title>Affiche le nom du bouton</title>
<script language="javascript">
function affiche(nombouton) {
    alert("Vous avez cliqué sur le bouton : "+nombouton);
}
</script>
</head>
<body>
<input type="button" name="Bouton" value="Nom"
onclick="affiche(this.value);"
/>
</body>
</html>
```



L'objet Event

- ▶ Lorsqu'un évènement se produit, un objet **Event** est créé dynamiquement puis est passé aux écouteurs d'évènements
 - ▶ Il va se propager dans l'arbre DOM selon le **flux d'évènement**
 - ▶ Il contient des **données spécifiques** au type d'évènement (position de la souris, code de la touche pressée)
- ▶ L'interface DOM Event est alors accessible par le gestionnaire de fonction via l'objet évènement passé comme unique argument

```
function envoiForm(event) {  
    if (event.target.elements.adresse.value === "") {  
        alert("Le champ "+event.target.name+" est vide.");  
        event.preventDefault(); // annule l'action  
    }  
}
```

L'objet Event

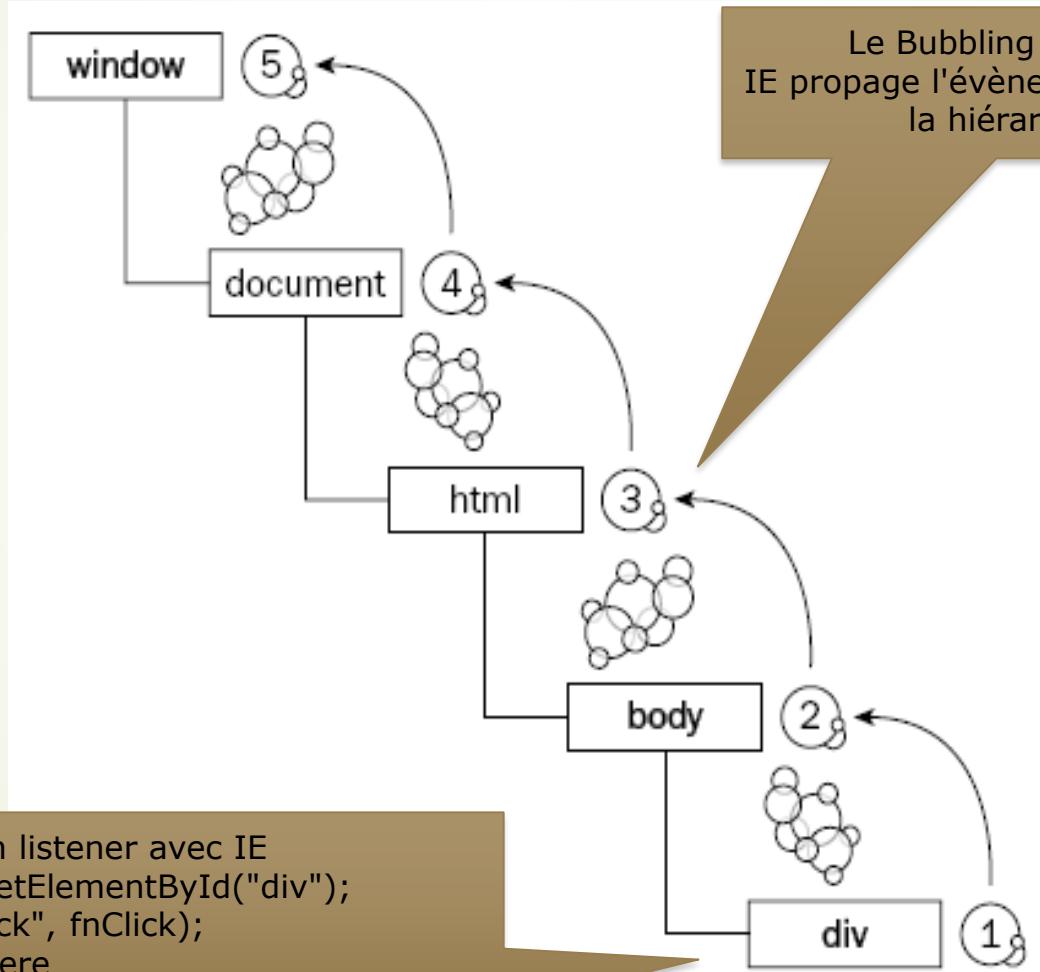
- ▶ Propriétés

- ▶ **height, width** : nouvelles dimensions de la fenêtre
- ▶ **layerX, layerY** : nouvelle position horizontale/verticale de la souris
- ▶ **pageX, pageY** : position horizontale/verticale de la souris dans le document
- ▶ **screenX, screenY** : position horizontale/verticale de la souris dans la fenêtre du navigateur
- ▶ **target** : référence à l'élément qui a déclenché l'évènement
- ▶ **type** : nom de l'évènement qui vient de se produire
- ▶ **which** : touche ou le bouton à l'origine de l'évènement

<https://developer.mozilla.org/fr/docs/Web/API/Event>

Le flux d'évènement (IE8 et inf.)

```
Ajout d'un listener avec IE  
var oDiv = document.getElementById("div");  
oDiv.attachEvent("onclick", fnClick);  
//do some other stuff here  
oDiv.detachEvent("onclick", fnClick);
```

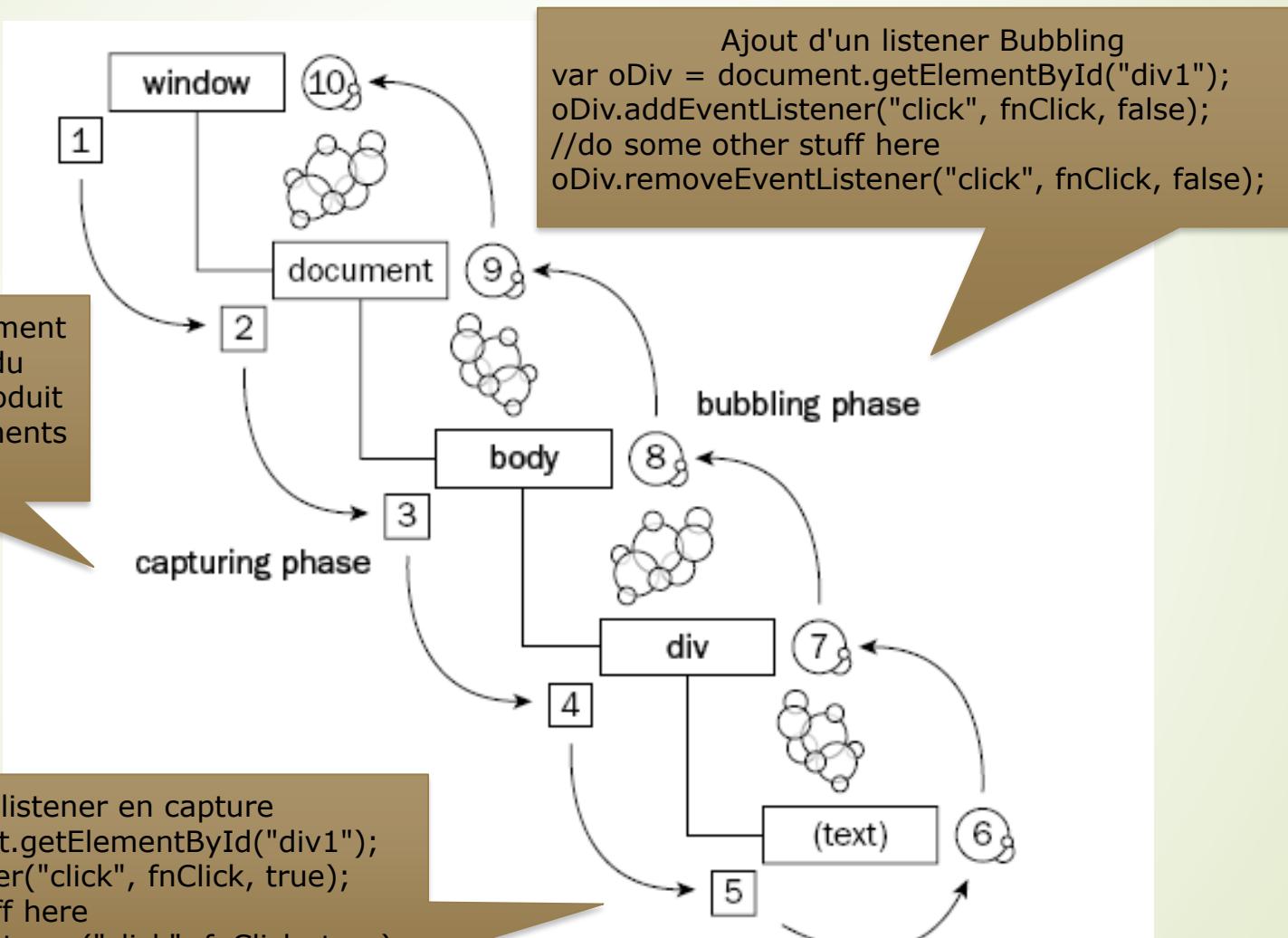


Le flux d'évènement (Browsers DOM)

La Capture d'évènement
En complément du
Bubbling, DOM introduit
la capture d'évènements

Ajout d'un listener en capture

```
var oDiv = document.getElementById("div1");
oDiv.addEventListener("click", fnClick, true);
//do some other stuff here
oDiv.removeEventListener("click", fnClick, true);
```



Les gestionnaires d'évènement DOM-2

- ▶ **addEventListener** est la manière d'enregistrer un écouteur telle que spécifiée dans le DOM du W3C avec les avantages suivants :
 - ▶ Permet d'ajouter plus d'un seul gestionnaire pour un évènement
 - ▶ Donne un contrôle plus fin sur la phase d'activation de l'écouteur (capture ou propagation)
 - ▶ Fonctionne sur tout élément DOM et pas uniquement les éléments HTML
- ▶ Dans IE8 et inf. on utilise plutôt la méthode **attachEvent**

```
if (el.addEventListener) {  
    el.addEventListener('click', modifieTexte, false);  
} else if (el.attachEvent) {  
    el.attachEvent('onclick', modifieTexte);  
}
```

<https://developer.mozilla.org/fr/docs/Web/API/EventTarget/addEventListener>

<https://www.w3.org/TR/DOM-Level-3-Events/#event-flow>

https://www.quirksmode.org/js/events_order.html#link4

Les formulaires

Développer des sites Web dynamiques avec JavaScript

L'objet Form

- ▶ La création d'un formulaire passe par l'ajout dans la page HTML d'un objet **Form** contenant des contrôles (liste, checkbox, bouton)
- ▶ C'est un sous-objet de **Document** qui dispose de propriétés et de méthodes

Propriétés	Description
action	Action exécutée par le formulaire (script serveur: PHP, ASPX)
enctype	Type de codage des données à employer lors de l'envoi
method	Méthode d'envoi du formulaire (GET ou POST)
name	Nom du formulaire

Méthodes	Description
reset()	Efface le contenu des champs du formulaire
submit()	Envoie le formulaire

Les éléments de formulaire

- ▶ Les éléments de formulaire ou **contrôles** permettent de récupérer les informations saisies ou choisies

Balise HTML	Objet JavaScript	Description
<input type ="text">	text	Zone de saisie monoligne
<textarea>	textarea	Zone de saisie multiligne
<input type="checkbox">	checkbox	Case à cocher
<input type="radio">	radio	Bouton radio
<select>	select	Liste de sélection
<input type="submit">	submit	Permet l'envoi des données
<input type="reset">	reset	Réinitialise le formulaire
<input type="password">	password	Zone de mot de passe
<input type="hidden">	hidden	Champ caché
<input type="file">	fileUpload	Zone de sélection de fichier

Les éléments de formulaire

- Grâce à l'apport de HTML5 et des **Web Forms**, de nouveaux éléments et types d'éléments arrivent dans nos formulaires

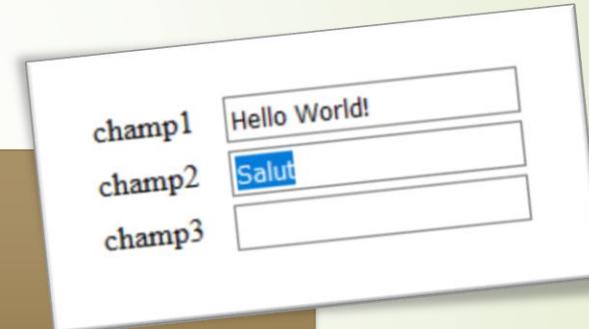


Balise HTML	Description
<input type="tel">	Basculement vers un clavier de type numérique
<input type="url">	Format spécifique respectant un pattern de type url
<input type="email">	Format spécifique respectant un pattern de type mail
<input type="search">	Saisie d'un mot-cléf (à coder)
<input type="date">	Aide au remplissage de type datepicker
<input type="month">	Choix du mois dans une année
<input type="number">	Incrémantation/Décrémentation d'un nombre
<input type="range">	Contenu évalué approximatif (ou curseur)
<input type="color">	Sélection d'un code couleur dans une palette

Manipulation de champ text

- ▶ Permet de recevoir des valeurs saisies par l'utilisateur ou d'afficher des résultats à l'aide de la **syntaxe pointée**
 - ▶ **value** : informations saisies dans le champ
 - ▶ **select()** : affiche en vidéo inverse le champ texte
 - ▶ **focus()** : positionne le curseur sur le champ texte

```
function manipchamptexte () {  
    form1.champ1.value = "Hello World!";  
    form1.champ2.select ();  
    form1.champ3.focus ();  
}
```



- ▶ Les champs **textarea** sont identiques aux champs **text**, mais ils sont utilisés dans le cas où la chaîne de caractères saisie est trop longue

Contrôle des cases à cocher

- ▶ Pour les **checkbox**, il est possible de savoir si ils sont cochés ou non et donc d'orienter le déroulement du script
 - ▶ **checked** : égale à true lorsque la case est cochée et à false sinon

```
function controlecaseacocher() {  
    if((form1.marche.checked == true)  
    && (form1.vp.checked == true))  
        alert("C'est bien de marcher de temps en temps");  
    else if(form1.vp.checked == true)  
        alert("Ce n'est pas bon pour l'environnement");  
    else if(form1.marche.checked == true)  
        alert("La marche est bonne pour la santé");  
    else  
         alert("Répondez en cochant au moins une case");  
}
```

Quel mode de transport utilisez-vous pour vous rendre sur votre lieu de travail ?

Marche

Véhicule personnel

Contrôle des boutons radio

- ▶ Les boutons **radio** doivent avoir le même attribut **name** afin de pouvoir les identifier en fonction de leur numéro d'index
 - ▶ La sélection d'un bouton radio désélectionne tous les autres
 - ▶ **checked** : égale à true lorsque la case est cochée et à false sinon

```
function controleboutonradio() {
    if(form1.connaitre_Javascript[0].checked) {
        alert("C'est très bien");
    }
    if (form1.connaitre_Javascript[1].checked) {
        → alert("Vous pouvez encore progresser");
    }
    if (form1.connaitre_Javascript[2].checked) {
        alert("Nous n'en sommes qu'au début");
    }
}
```

Connaissez-vous bien le Javascript ?

- Oui
- Un peu
- Pas du tout

Contrôle des valeurs d'une sélection de liste

- Le contrôle **select** se rapproche de celui des **checkboxes**, les éléments de la liste sont identifiés par un numéro d'index
 - length** : nombre de valeurs dans la liste
 - selectedIndex** : valeur dans la liste identifiée par son numéro d'index

```
function controleliste() {  
    if(form1.Question.selectedIndex == 0)  
        → alert("C'est moins");  
    else if(form1.Question.selectedIndex == 1)  
        alert("C'est plus");  
    else {  
        alert("C'est la bonne vitesse");  
    }  
}
```

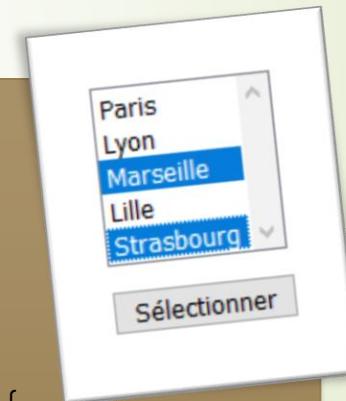
Quelle est la vitesse maximum du TGV lors de son record le 3 avril 2007 ?

630,1 Km/h ▾

Contrôle des valeurs d'une liste à sélections multiples

- Le contrôle **select multiple** permet de sélectionner plusieurs valeurs de la liste en utilisant :
 - la touche **[Shift]** pour les sélections contiguës
 - et la touche **[Ctrl]** pour les sélections qui ne le sont pas

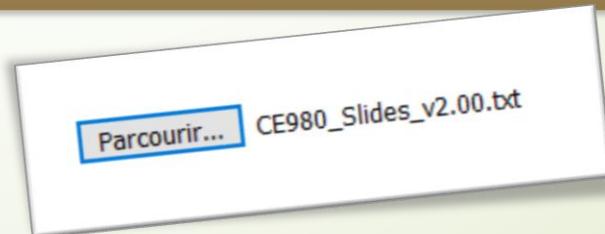
```
function maniplistemultiple() {  
    var ville="";  
    nb=form1.liste_ville.length;  
    i=form1.liste_ville.selectedIndex;  
    for (i = 0;i<nb;i++){  
        if(form1.liste_ville.options[i].selected){  
            ville=form1.liste_ville.options[i].value;  
            alert(ville+" est sélectionnée");  
        }  
    }  
}
```



L'envoi de fichier par formulaire

- ▶ Le contrôle **file** permet de sélectionner un ou plusieurs fichiers depuis l'appareil et de les uploader vers un serveur :
 - ▶ via un formulaire (script serveur: PHP, ASPX)
 - ▶ ou grâce à du code JavaScript via l'API File
- ▶ L'**API File** permet d'accéder à la **FileList** contenant les objets **File** qui correspondent aux fichiers sélectionnés
- ▶ Si on ne sélectionne qu'un seul fichier, on ne prendra en compte que le premier élément de la **FileList**

```
var fichierSelectionne = form1.choix_fichiers.files[0];
```



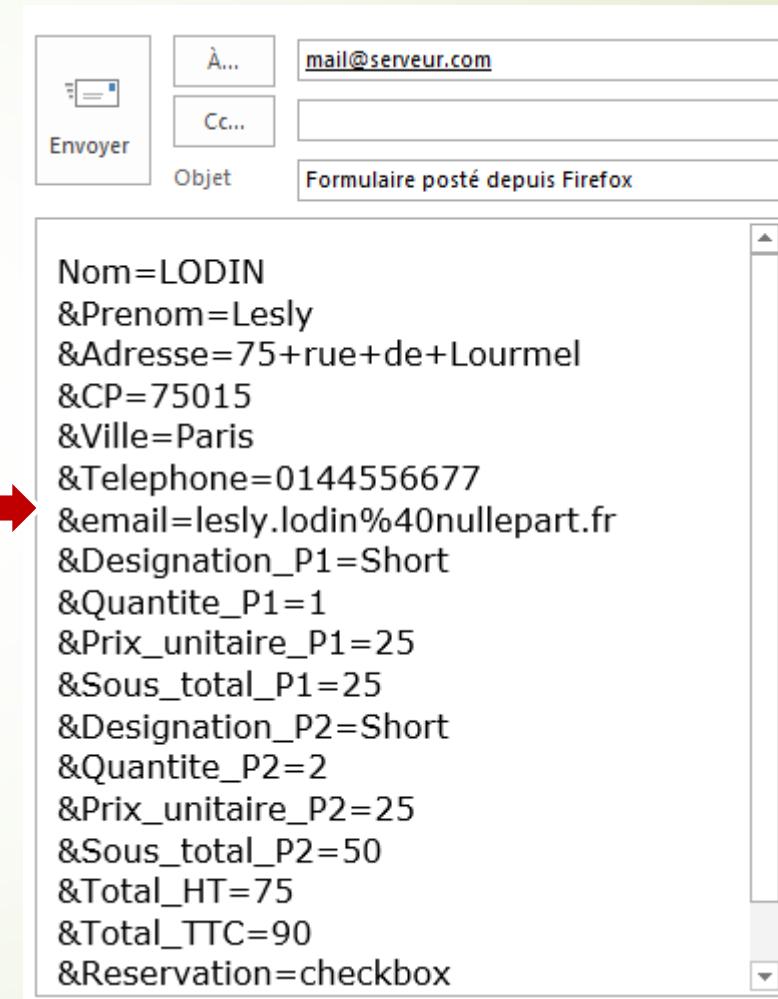
Valider et envoyer un formulaire par email

- Dans cet exemple, le formulaire permet de réserver des articles : les informations ne seront pas stockées dans une base de données mais seront envoyées par email à un destinataire prédéfini

Nom *	LODIN		
Prénom	Lesly		
Adresse	75 rue de Lourmel		
CP *	75015		
Ville *	Paris		
Téléphone	0144556677		
e-mail	lesly.lodin@nullepart.fr		
Désignation du Produit	Quantité	Prix unitaire	Sous-total
Short	1	25	25
Short	2	25	50
Total HT			75
Total TTC			90
Je déclare avoir pris connaissance des conditions générales de réservation			
<input type="checkbox"/> Valider <input type="button" value="Réinitialiser"/> <input type="button" value="Transmettre"/>			

Valider et envoyer un formulaire par email

```
<form  
action="mailto:mail@serveur.com"  
method="post"  
enctype="text/plain"  
name="Formulaire_reservation">  
...  
<input type="button" name="Submit"  
value="Valider"  
onclick="control()">  
<input type="reset" name="Submit2"  
value="Réinitialiser">  
<input type="submit"  
name="bouton_transmettre"  
id="bouton_transmettre"  
value="Transmettre">  
...  
</form>
```



A screenshot of an email client interface showing a message being composed. The recipient field contains "mail@serveur.com". The subject field contains "Formulaire posté depuis Firefox". The message body displays the form data as URL-encoded parameters:

Nom=LODIN
&Prenom=Lesly
&Adresse=75+rue+de+Lourmel
&CP=75015
&Ville=Paris
&Telephone=0144556677
&email=lesly.lodin%40nullepart.fr
&Designation_P1=Short
&Quantite_P1=1
&Prix_unitaire_P1=25
&Sous_total_P1=25
&Designation_P2=Short
&Quantite_P2=2
&Prix_unitaire_P2=25
&Sous_total_P2=50
&Total_HT=75
&Total_TTC=90
&Reservation=checkbox

A red arrow points from the bottom right of the code block towards the message body of the email window.



Améliorer l'interactivité avec JavaScript

Développer des sites Web dynamiques avec JavaScript

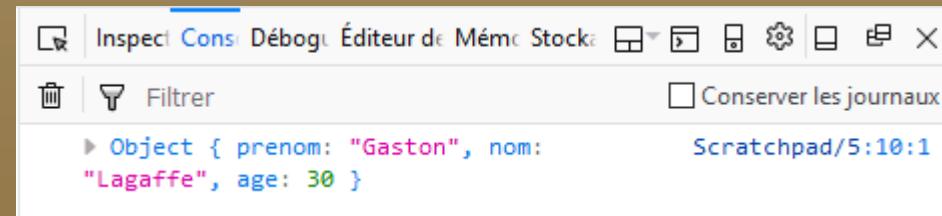
Création d'un objet avec le mot clé new

- ▶ Outre les objets natifs, il est également possible de créer nos propres objets en JavaScript
- ▶ On peut utiliser le mot clef **new** pour créer un nouvel objet à partir d'une classe

```
// Création d'un objet en utilisant new Object()
var personnage = new Object();

// Définition des propriétés du nouvel objet
personnage.prenom = 'Gaston';
personnage.nom = 'Lagaffe';
personnage.age = 30;

// Affichage d'une valeur
console.log(personnage);
```

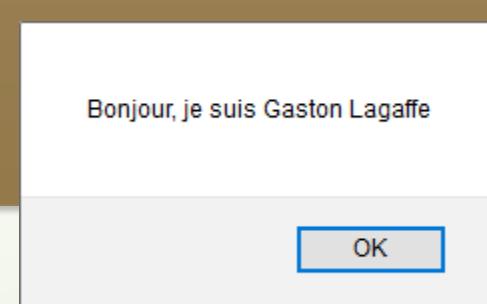


Création d'un objet littéral

- ▶ C'est la manière la plus simple de créer un objet JavaScript

```
// Création d'un objet littéral
var personnage = {
    prenom: 'Gaston',
    nom: 'Lagaffe',
    age: 30,
    presentation: function () {
        return 'Bonjour, je suis ' + this.prenom + ' ' +
            this.nom;
    }
};

// Affichage d'une valeur
alert(personnage.presentation());
```



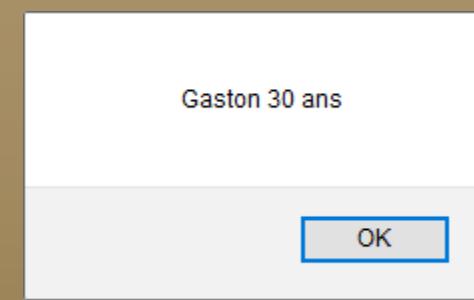
Création d'un objet via un constructeur

- Cette dernière méthode permet de créer un **constructeur** puis ensuite de créer des objets à partir de celui-ci

```
// Création d'un constructeur
function Personnage(p, n, a) {
    this.prenom = p;
    this.nom = n;
    this.age = a;
};

// Création d'un objet à partir du constructeur
var gaston = new Personnage('Gaston', 'Lagaffe', 30);

// Affichage d'une valeur
alert(gaston.prenom + ' ' + gaston.age + ' ans');
```



- Grâce à ce constructeur, on peut ensuite créer autant d'objets que nécessaire

Utilisation de JSON

- ▶ Lorsqu'il s'agit de stocker des données complexes, comme des objets, il faut d'abord les linéariser en chaînes de texte
- ▶ Le format **JSON** (JavaScript Object Notation) est la solution de prédilection :
 - ▶ **JSON.stringify()** : transforme un objet en une chaîne de texte linéarisée
 - ▶ **JSON.parse()** : reforme un objet à partir de la chaîne linéarisée

```
var monobjet = {  
    propriete1 : "valeur1",  
    propriete2 : "valeur2"  
};  
var monobjet_json = JSON.stringify(monobjet);  
var monautreobjet = JSON.parse(monobjet_json);
```



JavaScript et les cookies

- ▶ Les **cookies** sont des petits fichiers qui sont laissés sur le disque dur du visiteur
 - ▶ On y enregistre différentes informations concernant le visiteur (son identifiant, le but de sa visite, etc.)
 - ▶ Ils sont stockés sur le disque dur sous la forme d'un ou plusieurs fichiers
 - ▶ Dès la 1^{ère} visite, le navigateur y écrit des informations qu'il pourra lire lors de la prochaine connexion
 - ▶ Ils peuvent être modifiés et, une fois l'utilité des cookies passée, il est possible de les supprimer
 - ▶ Enfin pour tester convenablement les scripts comprenant des cookies, il convient de régler le niveau de sécurité du navigateur



Création d'un cookie

- On accède aux cookies via la propriété **cookie** de l'objet document
 - Leur taille ne doit pas être supérieure à **4 Ko**
 - Il est impossible d'en stocker plus de **20** par domaine
 - Leur nombre total, stockés sur l'ordinateur, ne doit pas dépasser **300**
 - Le stockage se fait sous la forme d'une chaîne de caractères **sans espace**

```
var name = "moncookie";
var value = "mavaleur";
document.cookie = name + "=" + value + "; expires=" +
"Thu, 31-Dec-2099 00:00:01 GMT";
```

Nom : moncookie
Contenu : mavaleur
Hôte : www.google.fr
Chemin : /
Envoi pour : Tout type de connexion
Expire : jeudi 31 décembre 2099 à 01:00:01

Ajouter des informations optionnelles au cookie

- ▶ Le domaine de validité
 - ▶ En spécifiant le domaine, il est normalement impossible de modifier le contenu du cookie à partir d'une page n'appartenant pas à ce domaine
- ▶ Le chemin d'accès
 - ▶ Permet d'indiquer pour quelle partie de l'URL le cookie est valable
- ▶ L'attribut de sécurité
 - ▶ Si cette option est active, le cookie ne sera transmis que si la connexion est sécurisée (protocole https)

```
document.cookie = "name=value; expires=dateexpiration;  
domain=nomdudomaine; path=chemin; secure";
```

Lecture, mise à jour et suppression d'un cookie

- ▶ **Lecture** : on manipule la chaîne de caractères dans le cookie, identifié par son nom

```
var debut = document.cookie.indexOf("moncookie");
if( debut == -1 ) {
    alert("il n'y a pas de cookie");
} else {
    var fin = document.cookie.length;
    alert(document.cookie.substring(debut, fin));
}
```

- ▶ **Mise à jour** : elle s'effectue simplement par la lecture puis l'affectation d'une nouvelle valeur au cookie
- ▶ **Suppression** : on rappelle le cookie en lui affectant une date d'expiration antérieure à la date du jour

Web Storage

- ▶ **Web Storage** est une solution adaptée aux besoins actuels de stockage de données variées dans le navigateur
 - ▶ Cette technique est plus puissante que les cookies, limités en taille
 - ▶ Les données sont stockées sous la forme de paires **clé/valeur** et une page ne peut accéder qu'aux données qu'elle a stockées elle-même
- ▶ Les données ne sont pas cryptées
 - ▶ Elles sont donc accessibles à tout utilisateur ayant accès au navigateur
 - ▶ Elles sont modifiables de la même façon
 - ▶ Il ne faut donc pas y placer d'informations sensibles
- ▶ Met à disposition deux interfaces
 - ▶ **sessionStorage** : stocke les données sur la durée d'une session de navigation
 - ▶ **localStorage** : stocke les données sans limite de durée de vie

Web Storage : exemple

```
// Enregistrer des données  
localStorage.setItem('prenom', 'Lesly');  
sessionStorage.setItem('age', 50);  
// Récupérer des données  
var person = localStorage.getItem('prenom');  
var age = sessionStorage.getItem('age');  
// Supprimer des données  
localStorage.removeItem('prenom');  
sessionStorage.removeItem('age');  
// Supprimer toutes les données  
localStorage.clear();  
sessionStorage.clear();
```



A screenshot of the Firefox developer tools interface, specifically the "Storage" panel. The panel lists various storage types: Cookies, Stockage de session, Stockage cache, Indexed DB, and Stockage local. Under Stockage local, there is a section for "http://localhost" which contains a table with two rows. The first row has "Clé" in the header and "Valeur" in the header. The second row shows a key "prenom" with a value "Lesly". There is also a "Filtrer les éléments" (Filter elements) input field at the bottom of the table.

https://developer.mozilla.org/fr/docs/Web/API/Web_Storage_API

Indexed Database

- ▶ **IndexedDB** est un autre moyen de stocker des données de manière persistante dans un navigateur
- ▶ Il permet de créer des applications web avec de riches possibilités de requêtes indépendamment de la disponibilité du réseau puisqu'elles peuvent fonctionner en ligne ou hors-ligne
- ▶ La base de données locale est organisée comme une collection d'objets insérés dans la base en utilisant une syntaxe JSON de manière similaire à des bases **NoSQL**
- ▶ Chaque objet est identifié par une clé unique générée au moment de l'insertion
- ▶ Un système d'indexation permet d'optimiser l'accès aux objets

Indexed Database

- ▶ Le modèle utilisé par **IndexedDB** est le suivant :
 - ▶ Ouvre une base de données (**Database**)
 - ▶ Crée un objet de stockage dans la base de données (**ObjectStore**)
 - ▶ Démarre une **transaction** et ajoute ou récupérer des données
 - ▶ Attend que l'exécution soit terminée, en écoutant l'évènement DOM
 - ▶ Exploite les **résultats** qui peuvent être trouvés dans l'objet de la requête
- ▶ Fonctionnement asynchrone :
 - ▶ Chaque opération va renvoyer un résultat via une **fonction de rappel**
 - ▶ Ainsi chaque opération rend la main immédiatement et est non bloquante
 - ▶ Les opérations sont transactionnelles
 - ▶ On ne peut pas manipuler de données en dehors d'une transaction

Indexed Database : exemple

```
var myDB = window.indexedDB;
// Ouvre une base de données
var open = myDB.open("MyDatabase", 1);
// Crée un objet de stockage
open.onupgradeneeded = function() {
    var db = open.result;
    var store = db.createObjectStore("MyObjectStore", {keyPath: "id"});
    var index = store.createIndex("NameIndex", ["name.last", "name.first"]);
};

open.onsuccess = function() {
    // Démarre une transaction
    var db = open.result;
    var tx = db.transaction("MyObjectStore", "readwrite");
    var store = tx.objectStore("MyObjectStore");
    var index = store.index("NameIndex");
    // Ecrit les données
    store.put({id: 12345, name: {first: "Bianca", last: "Castafiore"}, age: 45});
    store.put({id: 67890, name: {first: "Archibald", last: "Haddock"}, age: 50});
    // Exploite les données lues
    var getBianca = store.get(12345);
    var getArchi = index.get(["Haddock", "Archibald"]);
    getBianca.onsuccess = function() {console.log(getBianca.result.name.first)};
    getArchi.onsuccess = function() {console.log(getArchi.result.name.first)};
    // Ferme la BDD en fin de transaction
    tx.oncomplete = function() {
        db.close();
    };
}
```

Indexed DB	
http://localhost	
MyDatabase (default)	
MyObjectStore	
C	Valeur
12345	{"id":12345,"name":{"first":"Bianca","last":"Castafiore"},"age":45}
67890	{"id":67890,"name":{"first":"Archibald","last":"Haddock"},"age":50}

Le Drag & Drop : déclaration

- ▶ Le **drag and drop** (glisser-déposer en français) consiste à :
 - ▶ Saisir un objet en cliquant dessus
 - ▶ Déplacer l'objet en gardant le bouton gauche de la souris enfoncé
 - ▶ Poser l'objet en relâchant le bouton, une fois la position désirée atteinte
- ▶ Pour rendre un élément déplaçable :
 - ▶ On utilise son attribut **draggable** que l'on met à **true**
 - ▶ On branche son évènement **dragstart** à la fonction qui sera appelée dès qu'on commencera à le déplacer

```

```

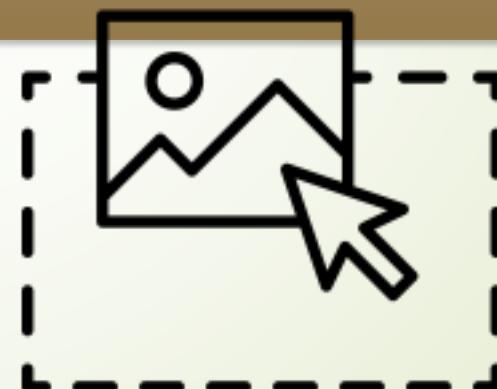


Le Drag & Drop : déplacement

- ▶ Pour initialiser un déplacement :

- ▶ On utilise l'objet **dataTransfer** qui sauvegarde la chaîne de caractères, transmise à l'élément cible qui accueillera l'élément déplacé via **setData()**
- ▶ Cette méthode prend deux arguments en paramètres :
 - ▶ Le premier est le type MIME des données ("text/plain")
 - ▶ le deuxième est **obligatoirement** une chaîne de caractères

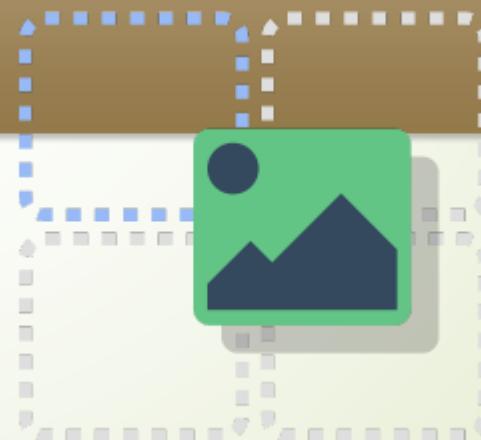
```
function demarre(evt) {  
    evt.dataTransfer.setData("text/plain", evt.target.id);  
}
```



Le Drag & Drop : survol

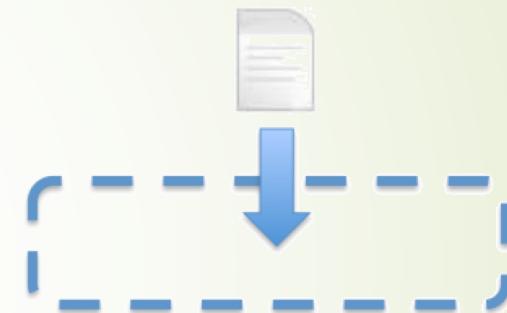
- ▶ Pour définir une zone de dépôt :
 - ▶ Par défaut le navigateur interdit de déposer un élément quelconque où que ce soit dans la page Web
 - ▶ Il faudra donc annuler cette action par défaut en utilisant la méthode `preventDefault()` via l'évènement `dragover` de l'élément survolé
 - ▶ Le curseur n'affichera plus d'interdiction en survolant la zone de dépôt

```
function survole(evt) {  
    evt.preventDefault();  
}
```



Le Drag & Drop : dépôt

- ▶ Pour terminer un déplacement :
 - ▶ On utilise à nouveau l'objet **dataTransfer** pour récupérer, grâce à la méthode **getData()**, le texte sauvegardé lors de l'initialisation
 - ▶ Cette méthode permet aussi de récupérer les éventuels fichiers qui ont été déposés par l'utilisateur : tableau **evt.dataTransfer.files**



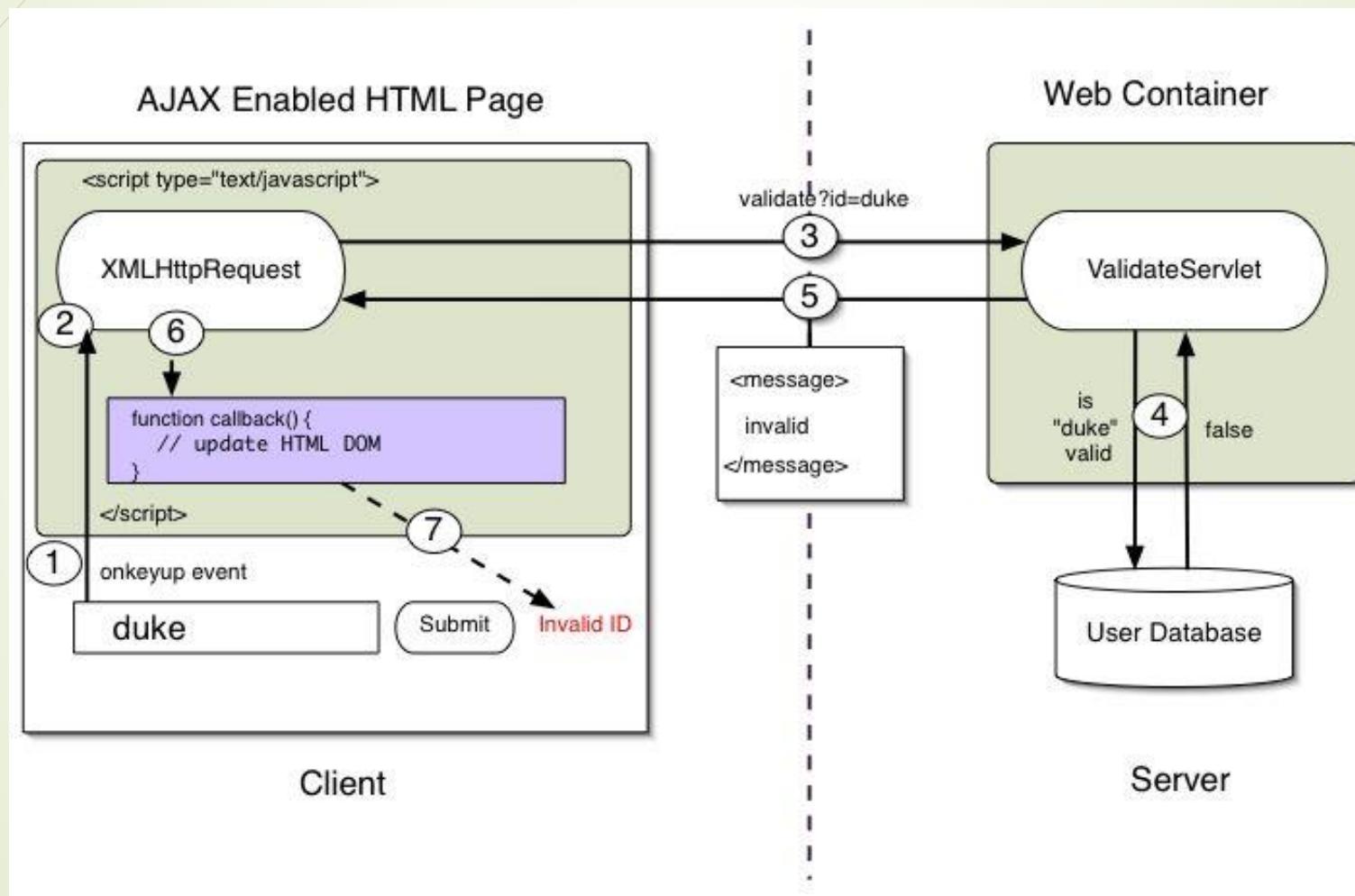
```
function depose(evt) {  
    var myID = e.dataTransfer.getData("text/plain");  
    var photo = document.getElementById(myID);  
    var cible = evt.target;  
    cible.appendChild(photo);  
}
```

https://developer.mozilla.org/fr/docs/Web/API/HTML_Drag_and_Drop_API

AJAX et JavaScript

- ▶ Tout comme DHTML, **AJAX** (Asynchronous JavaScript And XML) est un mélange de plusieurs technologies
 - ▶ HTML qui reste au cœur de la page web, aidé par les feuilles de style CSS pour toute la partie de présentation des données
 - ▶ DOM pour l'accès aux éléments de la page
 - ▶ Code de programmation côté serveur (PHP ou ASP), qui permet de récupérer des informations du serveur
 - ▶ Données renvoyées sous la forme d'un fichier texte ou JavaScript ou XML ou encore JSON qui sont traitées pour apparaître convenablement sur la page
- ▶ **AJAX** permet d'effectuer des rechargements de données, en provenance du serveur, tout en restant sur la même page
- ▶ Ce type de fonctionnement est appelé **asynchrone**

AJAX : exemple d'interaction



AJAX : processus

- ▶ Quand l'utilisateur génère un évènement, celui-ci appelle une fonction JavaScript dans la page en cours
- ▶ Cette fonction crée et configure un objet utilisé pour l'échange de données entre le client et le serveur :
 - ▶ Un objet **ActiveXObject** pour le navigateur IE7- (objet ActiveX)
 - ▶ Un objet **XMLHttpRequest** pour les autres navigateurs
- ▶ On assigne à un gestionnaire d'évènement une fonction de rappel à exécuter lorsque le serveur a répondu à la requête
- ▶ Le serveur traite la requête et renvoie la réponse, en général au format texte (souvent JSON) ou XML
- ▶ La fonction de rappel côté client traite la réponse et modifie la présentation de la page Web

AJAX : instantiation

```
function getXMLHttpRequest() {  
    var xhr = null;  
    if (window.XMLHttpRequest || window.ActiveXObject) {  
        if (window.ActiveXObject) {  
            try {  
                xhr = new ActiveXObject('Msxml2.XMLHTTP');  
            } catch (e) {  
                xhr = new ActiveXObject('Microsoft.XMLHTTP');  
            }  
        } else {  
            xhr = new XMLHttpRequest();  
        }  
    } else {  
        alert('Votre navigateur ne supporte pas AJAX');  
        return null;  
    }  
    return xhr;  
}
```

AJAX : envoi d'une requête HTTP

- Il faut d'abord définir les modalités d'envoi de la requête HTTP :

1. GET ou POST
2. URL du serveur
3. Une valeur booléenne : **asynchrone** (true) ou synchrone (false)

```
xhr.open("GET", "http://api.zippopotam.us/us/90210", true);
```

- Si la requête est du type **GET** :

```
xhr.send(null);
```

- Si elle est du type **POST** :

```
xhr.setRequestHeader("Content-Type",
                     "application/x-www-form-urlencoded");
xhr.send("nom=Gaston&age=30&ville=Bruxelles");
```

AJAX : changement d'état

- ▶ L'objet XHR une fois créé passe par plusieurs états différents :
 - ▶ **0** : L'objet XHR a été créé mais pas encore initialisé (pas open)
 - ▶ **1** : L'objet XHR a été créé mais pas encore envoyé (pas send)
 - ▶ **2** : La méthode send vient d'être appelée
 - ▶ **3** : Le serveur traite les informations et commence à renvoyer des données
 - ▶ **4** : Le serveur a fini son travail et toutes les données sont réceptionnées
- ▶ Il faudra donc détecter les changements d'état pour savoir où en est la requête et quand intervenir

```
xhr.onreadystatechange = function () {  
    if (xhr.readyState == 4 && (xhr.status == 200 ||  
        xhr.status == 0)) {  
        exploiteResultat();  
    }  
};
```

AJAX : lecture des données

- Il suffit d'utiliser une des deux propriétés suivantes :
 - responseText** : pour récupérer les données sous forme de texte brut
 - responseXML** : pour récupérer les données sous forme d'arbre XML

post code:	"90210"
country:	"United States"
country abbreviation:	"US"

```
function exploiteResultat() {  
    console.log(xhr.responseText);  
}
```

- Pour une réponse XML, on analyse le résultat à l'aide de l'API DOM

```
function exploiteResultat() {  
    var oXML = xhr.responseXML.getElementsByTagName("cp") [0];  
    var oDOM = oXML.childNodes [0].nodeValue;  
    console.log(oDOM);  
}
```