




Programmation Web JavaScript

Hicham EL KADIRI
https://github.com/ELKADIRI/L3_MIAW
hicham.elkadiri@univ-evry.fr



Au sommaire

- 1. Les bases de JavaScript
- 2. Les tableaux en JavaScript
- 3. Les objets prédéfinis
- 4. L'objet Window
- 5. L'objet Document
- 6. L'objet Form
- 7. AJAX



Langages et protocoles

HTML (Hyper **T**ext **M**arkup **L**anguage)

Langage à base de balises de type <BALISE> pour la description de documents hypertexte (liens) et hypermédia (images, sons...)

URL (Uniform **R**esource **L**ocator)

Adresse universelle de ressource en 3 parties :

- le protocole (par quelle méthode accéder ?)
- l'adresse DNS du site : www.monsite.com (où trouver l'information ?)
- le chemin pour y accéder et le nom du document (quel document récupérer ?)



Le web statique

- Pour une ressource donnée le serveur renvoie toujours la même réponse
- **HTML** permet seulement de présenter du texte, des liens et des images, il définit le contenu des pages Web
- **CSS** pour spécifier la mise en page des pages Web

Une page HTML/CSS simple est appelée page statique, elle n'offre que peu d'interaction à l'utilisateur



Le web dynamique

- Pour palier à ce manque d'interactivité il existe 3 méthodes :
 - Java permet d'écrire des applets (petites applications) interprétables par le navigateur du client via une machine virtuelle installée sur le poste client, Nécessite une machine virtuelle, gestion de la mémoire.
 - La technologie SERVER-SIDE (PHP, DOT.NET, JSP, AJAX, CGI): langage de script interprété par le serveur (Apache, IIS, ...) en fonction de paramètres passés par le client.
Chaque interaction du Client nécessite une nouvelle requête vers le serveur.
 - La technologie CLIENT-SIDE (JavaScript): langage de script à placer au sein du code HTML et interprété par le client (navigateur)
Code disponible au niveau du client, Problème de compatibilité selon les navigateurs.



Pourquoi le CLIENT-SIDE?

- Améliorer l'interactivité (temps de réponse plus court)
- Améliorer les débits sur le réseau (éviter des envois erronés, économie de requête au serveur web)
- Proposer des pages dynamiques (ergonomie, personnalisation, animation...)
- Aucun environnement ni compilateur nécessaire au développement : un éditeur de texte et un (des) navigateur(s) sont suffisants



Exemples JS

- Test d'un formulaire avant envoi, pour vérifier la validité d'une adresse e-mail ou le format d'une date de naissance par exemple.
- Génération en HTML d'un calendrier, d'une calculatrice, d'un glisser déplacer des éléments

Attention: JavaScript peut être désactivé ou non supporté par le navigateur

- Par conséquent, JavaScript ne doit être utilisé que pour améliorer l'expérience utilisateur, **pas pour la sécurité**. Si un champ est vérifié par JS, il doit éventuellement l'être à nouveau par le script de destination côté serveur, comme PHP.
- J'insiste : JavaScript est à utiliser **exclusivement** pour **faciliter la navigation** dans votre site.



Caractéristiques de JS

JavaScript est un langage :

- interprété (pas de compilation) → Langage de script
- sensible à la casse
- à base d'objets
- N'a pas accès aux fichiers locaux! Pour des raisons de sécurité
- multi-plateforme (ne dépend pas du système d'exploitation)
- **développé par Netscape (nom d'origine LiveScript)**
- **Microsoft (de son côté) a développé Jscript**
- **Problèmes de compatibilité entre les navigateurs**



Limites du JavaScript

- Le JavaScript est difficilement compatible entre les différents navigateurs. Il faut toujours se décider jusqu'à quel point ça doit être compatible.

<https://validator.w3.org>

- Tout le monde n'a pas JavaScript : Il faut toujours que la page contienne l'ensemble de l'information, accessible même sans JavaScript. JavaScript est là pour apporter un plus (ergonomie, dynamisme), mais on doit pouvoir s'en passer.

<https://caniuse.com/>

- JavaScript **n'est pas sécurisé**. Les programmes JS sont exécutés sur le client, on n'est jamais sûr de leurs résultats, il ne faut donc jamais faire confiance à une donnée provenant du client.



Le noyau JavaScript

Au niveau du langage, on distingue :

- le noyau JavaScript (le coeur du langage) comportant les objets de base, les opérateurs, les structures de contrôle...
- un ensemble d'objets prédéfinis associés au navigateur (fenêtres, documents, boutons, zone de saisie, images...)



Notions du langage JavaScript

- **JavaScript est un langage à base d'objets** : chaque objet possède des méthodes (ou fonctions), des propriétés et des objets.
- Dans une page Web, l'**objet** le plus élevé dans la hiérarchie est la fenêtre du navigateur : *window*. Cet objet ***window*** contient entre autres l'objet ***document*** qui lui même contient tous les objets contenus dans la page Web (paragraphe, formulaires, etc...). En plus de ces objets, il existe des objets créés par l'utilisateur.
- **Les méthodes** sont des fonctions qui permettent d'agir sur certaines propriétés de l'objet, les propriétés contiennent les paramètres d'un objet.



Notions du langage JavaScript

- Exemple d'un **objet voiture** : nous allons lui attribuer
 - **des propriétés** : la couleur, la marque, le numéro d'immatriculation,
 - **des méthodes** : tourner(), avancer(), reculer(), changer la couleur(),
 - **des objets** : les phares, les pneus,
- Une méthode permet de changer la couleur de la voiture, par contre aucune méthode ne nous autorise à changer la marque de cette voiture (ce qui entraînerait une modification des autres propriétés et éventuellement l'apparition ou la disparition de méthodes).
- Il en sera ainsi également avec nos objets JavaScript : nous pourrions accéder voire modifier les propriétés (couleur du texte, style de la fonte) des objets grâce aux méthodes



JavaScript n'est pas Java

<u>JavaScript</u>	<u>Java</u>
Code intégré dans la page Html	Module (applet) distinct de la page Html
Code interprété par le navigateur au moment de l'exécution	Code source compilé avant son exécution
Codes de programmation simples mais pour des applications limitées	Langage de programmation beaucoup plus complexe mais plus performant
Permet d'accéder aux objets du navigateur	N'accède pas aux objets du navigateur
Confidentialité des codes nulle (code source visible)	Sécurité (code source compilé)



JavaScript et HTML

- Place de l'élément **SCRIPT**
 - Possibilité d'intégrer du code JavaScript :
 - dans l'entête de la page.
 - dans le corps de la page.
 - dans un fichier externe
- Intégration dans un événement d'un objet de la page
 - Sous la forme d'un couple **attribut=valeur** :
 - Attribut = événement déclencheur

```
...
<FORM name="formulaire" onSubmit="maFonction();" >
...
```



JavaScript et HTML

- Code JavaScript contenu dans les documents HTML entre les balises `<script>` et `</script>`
- Les fonctions JavaScript peuvent être appelées :
 - depuis des événements associés aux balises
 - le click de la souris sur un lien


```
<a href="index.html" onClick="Javascript:PopUp();" >
Ouvre une fenêtre</a>
```
 - comme une URL classique


```
<a href="javascript:mafonction();" > Call my Function </a>
```
 - en insérant le code JavaScript au milieu du code HTML


```
 imprimez la page </a>
```




La balise <SCRIPT>

Le code JavaScript peut être placé :

- dans l'en-tête `<head>...</head>` (déclaration de l'ensemble de fonctions appelées à partir d'événements, d'URL)
- dans le corps du document `<body>...</body>` (pour un traitement synchrone lors de l'interprétation du document)

➤ dans un fichier séparé :


```
<SCRIPT language="JavaScript" SRC="JS/monprogjs">
</SCRIPT>
```

ou `<SCRIPT type="text/javascript">...</SCRIPT>`



Exemple

```
<!DOCTYPE html><html>
<head>
<title>Ex1</title>
<script> function saluer()
{ window.alert("Salut tout le monde !");
}
</script>
</head> <body>
<h4>Exécution immédiate</h4>
<script>
alert("Bonjour tout le monde..... !");
</script>
<h4>Exécution sur événement onClick</h4>
<form>
<input type="button" name="Bouton1" value="Salut" onClick="saluer();">
</form>
<h4>Exécution sur protocole javascript:</h4>
<a href = "javascript:saluer();">pour saluer</a>
</body> </html>
```



Résultat

Exécution immédiate

file://

Bonjour tout le monde..... !

OK


Exécution immédiate

Exécution sur événement onClick

Salut

Exécution sur protocole javascript:

[pour saluer](#)



Le langage JavaScript

- ❑ Toute instruction (ligne de code) se termine par un point virgule ;
- ❑ Variables (faiblement typées) : l'utilisation ne nécessite pas de déclaration (**détection automatique par l'interpréteur**)
- ❑ Opérateurs et Instructions (ceux du langage C)
- ❑ Méthodes : globales (associées à tous les objets), fonctions définies par l'utilisateur
- ❑ Objets : prédéfinis (String, Date, Math...) , liés à l'environnement (window, document...)
- ❑ Les Entrées/Sorties sont déléguées à l'hébergeant (le navigateur)
- ❑ Commentaires comme en langage C : **//commente une ligne,**
/*commente une portion
de code multi-ligne */



Déclaration de variables

- Utilisation de l'instruction **var variable=valeur;**
 - Pas de typage (détection automatique par l'interpréteur)
 - Types atomiques : entier, réel, chaîne de caractères, booléen.
 - Nom de variable sensible à la casse.
 - Portée : déclaration **en dehors** de fonction ⇒ globale
déclaration **dans** une fonction ⇒ locale
- Exemple

```
<HTML>
<HEAD> <TITLE> Exemple 1 </TITLE> </HEAD>
<BODY>
  <SCRIPT>
    var bonjour = "Bonjour !";
    var question = "Comment allez vous ";
    var phrase = bonjour + "<BR>" + question;
    document.write(phrase+ "aujourd'hui ?");
  </SCRIPT>
</BODY> </HTML>
```



Déclaration et création d'objets

- Existence d'objets prédéfinis
 - JavaScript intègre d'origine plusieurs type d'objets.
 - Déclaration : utilisation de **var**.
 - Création : utilisation du mot clé **new**, suivi du type d'objet.
- Exemple
 - Objet **Date**, très utile dans un environnement Internet.

```
...
// création d'un objet Date contenant la date du jour.
var date_jour=new Date();

// création d'un objet Date avec paramètres
var une_date=new Date(annee,mois+1,jour,heure,min);
...
```



Afficher la date et l'heure

```
<html><body>
<script>
var date = new Date();
    // déclaration d'une variable de type Date
    // la variable date contient alors la date courante
    var mois = date.getMonth() + 1;
    // la variable mois contient le N° du mois à partir de 0 pour janvier (à 11)
var jour = date.getDate();
var annee = date.getFullYear(); // ou : date.getFullYear(); pour éviter le problème de
    navigateur et sans condition if
if (navigator.appName == 'Netscape' || navigator.appName == 'IE' )
    annee = annee + 1900 ;
    // getYear() donne le numéro de l'année pour IE et Mozilla à partir de 1900
document.write("Date " + jour+ " / "+mois+ " / "+annee+ "<BR>");
document.write("Heure " + date.getHours()+ " : "+date.getMinutes()+
: "+date.getSeconds()+ "<BR> " + "Objet Date() = "+Date());
</script></body></html>
```

Date 19 / 11 / 2024
 Heure 11 : 6 : 10
 Objet Date() = Tue Nov 19 2024 11:06:10 GMT+0100 (heure normale d'Europe centrale)



Utilisation de tableaux

■ Accès aux éléments d'un tableau

◦ Utilisation des crochets : []

```
...
var tableau=new Array;
tableau[0]=10;
tableau[9]=5;
...
```

Propriétés de l'objet

```
...
// obtention du nombre d'éléments de l'objet tableau
var dimension=tableau.length;
...
```



Tableaux associatifs

■ Principe

- L'indice est une chaîne de caractères

■ Exemple

- Chargement d'une page HTML en fonction du jour de la semaine

```
...
var tab=new Array;
tab["Lundi"] ="semaine.html";
tab["Mardi"] ="semaine.html";
tab["Mercredi"] ="enfant.html";
tab["Jeudi"] ="semaine.html";
tab["Vendredi"] ="semaine.html";
tab["Samedi"]  ="weekend.html";
tab["Dimanche"] ="weekend.html";
```



Tableaux d'objets

■ Principe

- Array permet de stocker des objets de n'importe quel type :
 - atomique : entier, réel, chaîne de caractères, booléen.
 - prédéfini : Date, ...

■ Exemple

```
var animal=new Array;
// création de plusieurs instances d'Animal
var milou=new Animal("Milou","Chien");
var titi=new Animal("Titi","Canari");
// stockage d'instances d'Animal dans un tableau
animal[0]=milou;
animal[1]=titi;
animal[2]=new Animal("Rominet","Chat");
```



Tableaux multi-dimensionnels

■ Principe

- Array permet de stocker des objets, donc des tableaux.

■ Exemple

```
...
var row0=new Array; row0[0]="O"; row0[1]="X"; row0[2]=" ";
var row1=new Array; row1[0]="X"; row1[1]="O"; row1[2]=" ";
var row2=new Array; row2[0]=" "; row2[1]="O"; row2[2]="X";

var morpion=new Array;
morpion[0]=row0; morpion[1]=row1;morpion[2]=row2;
...
morpion[1][2]="Y";
...
```

0	X	
X	0	Y
	0	X




Exemple d'utilisation de tableaux

■ Affichage de la date du jour

```
<HTML>
<BODY>
  <SCRIPT>
    var dt=new Date();
    var jour=dt.getDay( ); // renvoi un jour [0..6]
    var numero=dt.getDate( ); // renvoi le numéro dans le mois
    var mois=dt.getMonth( ); // renvoi le mois [0..11]
    var tab_jour=new Array( "Dimanche","Lundi","Mardi",
                           "Mercredi","Jeudi","Vendredi","Samedi");
    var tab_mois=new Array( "Janvier","Février","Mars","Avril","Mai",
                           "Juin","Juillet","Août","Septembre",
                           "Octobre","Novembre","Décembre");

    document.write("Nous sommes le "+tab_jour[jour]+" "+numero+" "+tab_mois[mois]);
  </SCRIPT>
</BODY>
```

Nous sommes le Mardi 19 Novembre



Sortie écran

- `document.write(...);`

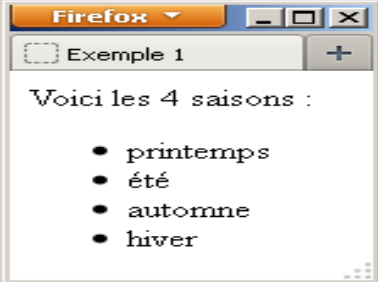
Exemple


```

<HTML>
  <HEAD><TITLE> Exemple 1 </TITLE> </HEAD>
  <BODY>
    <SCRIPT>
      var Les4saisons = new Array("printemps", "été",
        "automne", "hiver");

      document.write("Voici les 4 saisons : <ul>");
      for (var i=0; i<Les4saisons.length; i++)
      { document.write("<li>" + Les4saisons[i]+"</li>");
      }
      document.write("</ul>");
    </SCRIPT>
  </BODY>
</HTML>

```





Structures de contrôle

- Test conditionnel : if ... else ...
- But
 - Permet de diriger l'exécution du script selon des conditions.
- Exemple

```

<SCRIPT>
...
    if(age<18) { alert("Vous devez être majeur");
                window.location.href="mineur.php";
            }

    else        { window.location.href="majeur.php";
            }

...
</SCRIPT>

```

Conditions imbriquées

```
<HTML>
<head><meta charset="utf-8">
</head>
<BODY>
<SCRIPT>
var age=window.prompt("Donnez votre âge", "" )
    if (age <= 0)
        alert("Cela n'a pas de sens !")
    else if (age <=13)
        alert("Vous êtes encore bien trop jeune
...");
    else if (age <18)
        alert("Désolé,vous êtes encore
mineur(e)");
    else if (age <25)
        alert("Vous êtes déjà majeur(e) !");
    else alert("Ne vous vieillissez donc
pas !");
</SCRIPT>
</BODY>
</HTML>
```

Boucle itérative : for

- But
 - Répéter des instructions un nombre déterminé de fois.
- Syntaxe


```
for(initialisation; condition; opération )
  { ... instructions ... }
```
- Exemple


```
var somme=0;
for( var i=1; i<=10; i++ )
{ somme += i; // équivalent à somme = somme +i;
}
...
```

Notations abrégées du C



Boucle conditionnelle : while

- But
 - Répéter des instructions tant qu'une condition est VRAIE.
- Syntaxe


```
while(condition) { ... instructions ... }
```
- Exemple

```
function demander()
{ var nb=0;
  while(nb<=100)
  { nb = prompt("Entrez un nombre supérieur à 100", "");
  }
  alert("Merci !");
}
```



Opérateurs

Ceux du langage C

- arithmétiques : + - * / %
- Incrémentation / décrémentation : i++ , i-- ,...
- logiques : && (ET) | | (OU) ! (NON)
- bit à bit :
 - ❖ & (AND) | (OR) ~ (Not)
- comparaisons: ==, ===, !=, <, >, <=, >=
- concaténation de chaînes : +

L'opérateur + est l'addition ou la concaténation selon qu'il agit sur un numérique ou sur une chaîne de caractère (qui est le type d'une variable par défaut!)



Affectations

affectation simple : =

```
var MyString= "valeur";
```

affectation conditionnelle :

```
var variable = (condition) ? exp_alors : exp_sinon
```

```
var MyOtherString = (a > b) ? "plus" : "moins";
```

Affectation avec opération : += -= *= ...

```
var MyInteger +=3; // équivaut à MyInteger = MyInteger +3;
```

Distinguer l'affectation (=) et la comparaison (==) ou (===)

let permet de définir des variables au sein d'un bloc et des blocs qu'il contient.

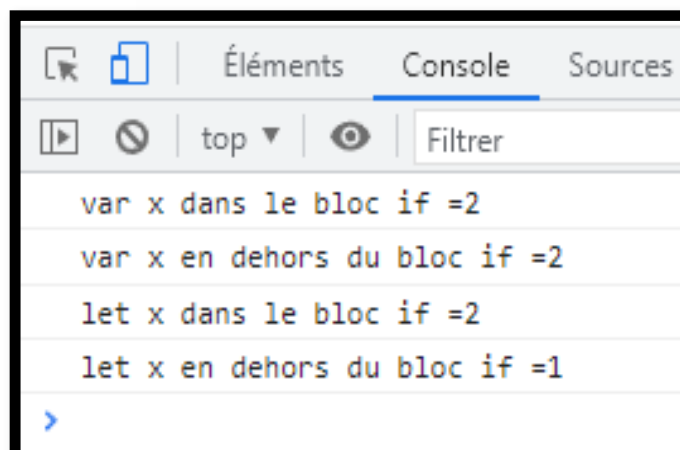
var permet quant à lui de définir une variable dont la portée est celle de la fonction englobante



let et var

```
<!DOCTYPE html>
<html><body>
<p id="p1">Variable : let et var</p>
<script>
function varTest() {
  var x = 1;
  if (x === '1') { // remplacer x=== '1'
    var x = 2; // c'est la même variable !
    console.log('var x dans le bloc if =' + x); // 2
  }
  console.log('var x en dehors du bloc if =' + x); // 2
}

function letTest() {
  let x = 1;
  if (x === '1') { // remplacer x=== '1'
    let x = 2; // c'est une variable différente
    console.log('let x dans le bloc if =' + x); // 2
  }
  console.log('let x en dehors du bloc =' + x); // 1
}
</script>
</body>
<script>
varTest();
letTest();
</script></html>
```





Types primitifs

- Number: Entier, décimal ou hexadécimal ou octal, réel
- Booléen (Boolean) : true ou false
- Chaîne de caractères (String) : 'chaine' ou "chaine"
 - Caractères séparateurs
 - \t (tabulation)
 - \n (passage à la ligne suivante) ou
 - \r (idem)
 - \b (backspace) ou
 - \f (idem)



Conversions

- Typage faible : Type String = type dominant
- JavaScript fait des conversions implicites selon les besoins

```
var N=12;    // N numérique
var T="34";  // T chaîne de caractères
var X=N+T;   // X est la chaîne de caractères "1234"
```

- Il existe des types particuliers pour les variables de type objets:
 - null : le type de données "null" est objet vide
 - undefined : variable déclarée sans valeur
 - infinity,
 - NaN (Not a Number)
- var ou let : déclaration d'une variable
- typeof : type d'une variable





Types de données JS avec typeof

```

var longueur = 16;                // Number
var Prenom = "Johnson";          // String
var x = {Prenom:"John", Nom:"Dia"}; // Object
typeof [1,2,3,4];                 // Return "Object" (pas "Array")

var y;
typeof function myFunc() { }      // Return "Function"
typeof 3.14;                       // Return "Number"
typeof true;                       // Return "Boolean"
typeof false;                     // Return "Boolean"
typeof y;                          // Return "Undefined" (car y sans valeur)
typeof undefined;                 // Undefined
typeof null;                      // Object
null === undefined;               // Return false
null == undefined;                // Return true

```



Types de données JS

```

<!DOCTYPE html>
<html><body>
<p>Undefined et null sont égaux en valeur mais de type différent:</p>
<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>
<p id="demo4"></p>
<script>
document.getElementById("demo1").innerHTML =
typeof undefined + "<br>" +
typeof null + "<br><br>" +
(null === undefined) + "<br>" +
(null == undefined);
var car = "";
document.getElementById("demo2").innerHTML =
"La valeur est : " +
car + "<br>" +
"Le type est : " + typeof car;

```

Undefined et null sont égaux en valeur mais de type différent:

undefined
object

false
true

La valeur est :
Le type est : string

[object Object]

object
object
object
function

Prenom = John - Nom = Dia - age = 50 - colorYeux = blue -
0 = 1 - 1 = 2 - 2 = 3 - 3 = 4 -



Types de données JS

```
document.getElementById("demo4").innerHTML =
typeof {prenom:'John', age:34} + "<br>" +
typeof [1,2,3,4] + "<br>" +
typeof null + "<br>" +
typeof function myFunc(){};
var personne = {Prenom:"John", Nom:"Dia", age:50, colorYeux:"blue"};
//personne = undefined;
//personne = null;
document.getElementById("demo3").innerHTML = personne;
for(i in personne)
    { document.write(i+" = "+personne[i]+" - ");
    for(i in tableau)
        { document.write(i+" = "+tableau[i]+" - "); }

</script>
</body></html>
```



Fonctions

- Une fonction JavaScript est un bloc de code conçu pour effectuer une tâche particulière.
- Une fonction JavaScript est exécutée lorsque "quelque chose" l'invoque.
- **Emplacement de la déclaration**
 - Dans l'entête de la page <HEAD> ou avant la fermeture de la page </BODY>
 - Utilisation de la syntaxe : `function nom_fonction([param1, ...])`
- **Corps de la fonction**
 - Délimité par `{ ... }`
 - Contenu :
 - déclaration des variables locales, propres à la fonction,
 - instructions réalisés par la fonction,
 - **return** pour renvoyer une valeur ou un objet(cette instruction n'est pas obligatoire ⇒ fonction qui ne renvoie pas de valeur)



Fonctions

■ Appel de fonction

- Peut avoir lieu à n'importe quel endroit de la page :
 - dans d'autres fonctions,
 - dans le corps de la page.
- Accéder à une fonction sans () renverra la définition de la fonction au lieu du résultat de la fonction:
- Utilisation de : `nom_fonction([param1, ...])`;

```
<HTML>
  <HEAD> <SCRIPT> // déclaration de fonction
    function bonjour(prenom)
    { document.write("Bonjour"+"<br>"+prenom); }
  </SCRIPT>
</HEAD>
<BODY><SCRIPT>bonjour("Toto")+"<br>"+bonjour();</SCRIPT>
</BODY> </HTML>
```

Bonjour
Toto
Bonjour
undefined



Exemple

```
<!DOCTYPE html><html>
<body>
<p id="demo"></p>
<p id="demo1"></p>
<p id="demo2"></p>
<p id="demo3"></p>
<p id="demo33"></p>
<p id="demo4"></p>
<button onclick="displayDate();">La date d'aujourd'hui est ?</button>
```

```
<script>
function displayDate() {
  document.getElementById("demo4").innerHTML = Date();
}
myFunction();
function myFunction() {
  var carName = "Volvo";
  document.getElementById("demo1").innerHTML = typeof carName + " " + carName;
}
document.getElementById("demo2").innerHTML = typeof carName;
```

La marque de la voiture Fiat

string Volvo

undefined

John Doe - fullName : John Doe

La date d'aujourd'hui est ?

La date est ?



Exemple

```
var car = {marque:"Fiat", modele:"500", color:"white"}; // Créer un objet:
// Afficher certaines données de l'objet:
document.getElementById("demo").innerHTML = "La marque de la voiture " + car.marque;
// Créer un objet:
var personne = {
  firstName: "John",
  lastName : "Doe",
  id      : 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};
// Afficher certaines données de l'objet :
document.getElementById("demo3").innerHTML = personne.fullName();
// Afficher certaines données de l'objet :
document.getElementById("demo33").innerHTML =personne["firstName"] + " " + personne["lastName"];
</script>
<button onclick="this.innerHTML=Date()">La date est ?</button>
</BODY> </HTML>
```



Déclenchement d'instructions javaScript

- **Programmation événementielle**
 - JavaScript = langage réactif
 - L'interaction avec l'utilisateur est gérée via des événements
 - Événement = tout changement d'état du navigateur
- **Production d'événement**
 - Déclenché par l'utilisateur ou par le code javaScript



Déclenchement d'instructions javascript

- **Récupération des événements**
 - Gestionnaire d'événement qui associe une action (fonction javascript) à la détection d'événement
- **Événements détectables**
 - Nom de l'événement précédé de **on** :
onBlur, onChange, onClick, onFocus, onLoad, onMouseover, onSelect, onSubmit, onUnload,...
- **Association événement = action**
 - Dans le code HTML, identique à la déclaration d'une propriété :
`<nom_élément attributi = propriétéi événementj = "actionj" >`



Événements javascript

- onBlur** : Désélection, Détecte l'enlèvement du focus de l'élément HTML (TEXT, TEXTAREA, SELECT)
- onFocus** : Détecte la sélection de l'élément lorsqu'on donne le focus au composant (TEXT, TEXTAREA, SELECT)
- onChange** : la valeur d'un champ de formulaire à été modifiée par l'utilisateur (TEXT, TEXTAREA, SELECT)
- onSelect** : un champ de formulaire est sélectionné (par clic souris ou tabulation) (TEXT, TEXTAREA)
- onClick** : Détecte le clique sur l'élément (A HREF, BUTTON, CHECKBOX, RADIO)
- onMouseOver** : Détecte la souris sur un lien ou une ancre (A HREF, AREA)
- onMouseOut** : Détecte la souris lorsqu'il quitte un lien ou une ancre (A HREF, AREA)
- onLoad** : Détecte le chargement d'une fonction ou d'une page dans le navigateur (BODY)
- onUnload** : Détecte le déchargement (BODY)
- onError** : Détecte l'erreur au chargement (BODY, IMG)
- onAbort** : arrêt du chargement (IMG)
- onSubmit** : Détecte la soumission d'un formulaire (bouton de type SUBMIT)
- onReset** : efface les saisies d'un formulaire (bouton de type RESET)
- onKeyDown** : Détecte lorsque l'utilisateur appuie sur une touche du clavier (TEXT, TEXTAREA)



Liste de sélection SELECT

```

<HTML>
<HEAD>
</HEAD>
<BODY>
  <FORM NAME="F">
    <select NAME="liste" onChange='F.resultat.value+=F.liste.options[F.liste.selectedIndex].value'>
      <option VALUE="matin ... " >Matin </option>
      <option VALUE="midi ... " >Midi </option>
      <option VALUE="soir ... " >Soir </option>
    </select>
    <INPUT TYPE="text" SIZE="40" NAME="resultat" VALUE="Vous avez sélectionné : ">
  </FORM>

</BODY>
</HTML>

```



Propriétés de l'objet SELECT

- **selectedIndex** est une propriété dont la valeur est le numéro de l'élément sélectionné dans la liste
var num = F.liste.**selectedIndex** ---> *num* est le numéro de l'élément sélectionné dans le composant liste situé dans le formulaire F.
- **options[]** est le tableau prédéfini contenant les *objets* de la liste
- **F.liste.options[num]** est l'objet champ situé au N° num (rappel : le 1er a le numéro 0)
- **F.liste.options[num].value** est la valeur de l'option N° num de la liste.
- **F.liste.options[num].text** est le texte suivant l'élément <option>.



Notion d'objet Javascript

Un objet est une collection de propriétés (variables associées à un même objet) et de méthodes (fonctions associées à un même objet)

Un objet dérive d'une classe (Sorte de moule à objets)

JavaScript permet par contre de définir des propriétés après définition via l'objet "prototype"

JavaScript met à disposition des objets prédéfinis (arborescence d'objets) et permet de créer ses propres objets

Création d'un objet par définition de son constructeur (fonction du nom de la pseudo-classe avec affectation des propriétés à partir des paramètres et déclaration des méthodes)



Déclaration et création d'objets

- Deux types d'objets
 - Objets prédéfinis
 - Objets propres
- Création d'objets propres
 - Par appel d'une fonction qui va créer les propriétés de l'objet.
 - Utilisation de **this** pour faire référence à l'objet courant

Exemple

```
var mon_chien=new CreerChien("Milou","Fox Terrier");
function CreerChien(le_nom,la_race)
{ this.nom=le_nom;
  this.race=la_race;
}
document.write(mon_chien.nom);
```




Déclaration et création d'objets

- Accès aux propriétés d'un objet
 - Utilisation de la notion pointée : objet.propriété
- Possibilité de parcourir toutes les propriétés et les fonctions d'un objet et son arborescence
 - Utilisation de la boucle : for (i in object) { ... object[i] ... }
i = nom de la propriété, object[i] = valeur de la propriété

```
// parcours des propriétés de l'objet parent
window
var object=window;
for(i in object)
  { document.write(i+" = "+object[i]+"<br>"); }
```



Déclaration et création d'objets

- Déclaration de méthodes
 - Association de fonctions dans la création de l'objet.
- Exemple

```
function CreerChien(le_nom,la_race)
{ this.nom=le_nom;
  this.race=la_race;
  this.Afficher=AfficherChien();
}
function AfficherChien()
{ document.write("Ce chien s'appelle "+this.nom
                 +". C'est un "+this.race+".");
}
```



Instructions spécifiques Objets

Création d'une instance : **new** // à utiliser dans constructeur

```
var MonObjet = new Objet(x1,x2);
```

Référence de l'objet : **this** //référence l'objet courant this.nom=valeur;

```
//affecte la valeur "valeur" à la propriété nom
```

Parcours : **for** //parcours des propriétés d'un objet

```
for (variable in objet) { instructions }
```

Références à un objet : **with** //Evite l'utilisation du this

```
with (MonObjet){
    var y1=zz; // raccourci de MonObjet.y1
    var y2=vv; // raccourci de MonObjet.y2
}
```

Suppression d'objet :

```
delete MonObjet;
```



Les objets prédéfinis

Structure de données en arbre avec au sommet l'objet **Global** qui définit un ensemble de méthodes communes à tous les objets.

Les objets "prédéfinis" sont

1. Object (création d'un objet)
2. Function (création d'une fonction)
3. Array (création d'un tableau)
4. String (création d'une chaînes de caractères)
5. Boolean (création d'un booléen)
6. Number (création d'un nombre)
7. Date (création d'une date)
8. Math (création de structures mathématiques)

A ces objets, s'ajouteront une hiérarchie d'objets liée à l'hébergeant (selon le navigateur)



Objet String

Type par défaut d'une variable en JavaScript

- Propriété
length (1 seule propriété)
- Méthodes de manipulation de chaînes :
indexOf("chaîne", "s/chn,pos), substring(début,fin+1), charAt(n),
lastIndexOf("chaîne"),toLowerCase(), toUpperCase(), split(),
toString()

Exemple :

```
var T = "Bonjour";
T.indexOf("o"); // 1
T.lastIndexOf("o"); // 4
T.charAt(3); // j
T.substring(3,5); // jo
T.toUpperCase(); // BONJOUR
```



Objet String

■ Exemple : length : longueur de la chaîne

```
var chaine='Exemple de longueur';
document.write(chaine.length);
```

19

Concaténation

☐ Utilisation de +

```
var chaine1='Vive ';
var chaine2='Javascript';
var chaine3=chaine1 + chaine2;

document.write(chaine3);
```

Vive Javascript



Méthodes associées aux chaînes

■ Accès à une sous-partie d'une chaîne

- `charAt(int n)` : renvoie la valeur du n^{ème} caractère
- `substring(int i, int j)` : renvoie de la chaîne comprise entre le i^{ème} caractère (inclus) et le j^{ème} caractère (exclus)

■ Recherche d'une sous-chaîne

- `indexOf(string souschaîne, [int pos])` : renvoie l'indice de la 1^{ère} occurrence de *souschaîne* dans la chaîne
- `lastIndexOf(string souschaîne, [int pos])` : renvoie l'indice de la dernière occurrence de *souschaîne* dans la chaîne
- L'option **pos** permet de n'effectuer la recherche :
 - qu'à partir d'une certaine position pour `indexOf`
 - jusqu'à une certaine position pour `lastIndexOf`
- Si *souschaîne* n'est pas trouvée, les 2 fonctions renvoient -1



Méthodes associées aux chaînes

■ Exemples de recherche de sous-chaînes

```
var date      = "28/02/2002";
document.write(date.indexOf("/"));
document.write("<br>" + date.lastIndexOf("/"));
document.write("<br>" + date.indexOf("/", 3));
document.write("<br>" + date.lastIndexOf("/", 3));
document.write("<br>" + date.indexOf("\\"));
//1er \ caractère échapement
```

```
var mois = date.substring(date.indexOf("/") + 1, date.lastIndexOf("/"));
document.write("<br>mois = " + mois);
```

2
5
5
5
2
-1
02



Méthodes associées aux chaînes

■ Conversions

- toUpperCase() : conversion en MAJUSCULES
- toLowerCase() : conversion en minuscules
- big(), blink(), bold(), fixed(), italics(), small(), sub(), strike(), sup(), fontcolor(string couleur), fontsize(string taille) : ajout de balises HTML de mise en forme

■ Exemple

```
var chaine="Exemple min/MAJ";
document.write("<br>"+chaine.toUpperCase());
document.write("<br>"+chaine.toLowerCase());
document.write("<br>"+chaine.italics());
document.write("<br>"+chaine.strike());
document.write("<br>"+chaine.fontcolor("red"));
document.write("<br>"+chaine.fontsize("+2"));
```

EXEMPLE MIN/MAJ
 exemple min/maj
Exemple min/MAJ
~~Exemple min/MAJ~~
 Exemple min/MAJ
 Exemple min/MAJ



Objet Number / Boolean

Propriétés :

MAX_VALUE + grand nombre pouvant être sauvegardé

MIN_VALUE + petit nombre pouvant être sauvegardé

NaN Not A Number

POSITIVE_INFINITY +infini

NEGATIVE_INFINITY -infini

Méthodes (communes à tous les objets)

toString() retourne une chaîne

valueOf() génère une erreur si l'objet n'est pas un nombre

Objet Boolean (simple encapsulation d'un booléen)

Propriété

Boolean renvoie un booléen

Méthodes :

toString(), valueOf()



Objet Number

■ Une chaîne est-elle un nombre ?

- Utile pour la vérification de la saisie de champ de formulaire : saisie de quantités, de prix...
- `isNaN(string chaîne)` renvoie :
 - `TRUE` si la chaîne n'est pas un nombre
 - `FALSE` sinon

■ Exemple

```
var nombre="3.14";

if(!isNaN(nombre)) document.write(nombre+ "est un
nombre");
else document.write(nombre+ "n'est pas un nombre");
```



3.14 est un nombre



Conversion chaîne → nombre

■ Utilité

- Effectuer des opérations numériques sur des données initialement textuelles (cas des saisies de formulaire notamment)
- `parseInt(string chaîne)` : conversion d'une chaîne en entier
- `parseFloat(string chaîne)` : conversion d'une chaîne en réel

■ Exemple

```
var chaine="3.14";
var entier=parseInt(chaine);
var reel=parseFloat(chaine);

document.write("entier = "+entier+"<br>");
document.write("reel = "+reel);
```



entier = 3
reel = 3.14



Objet Math

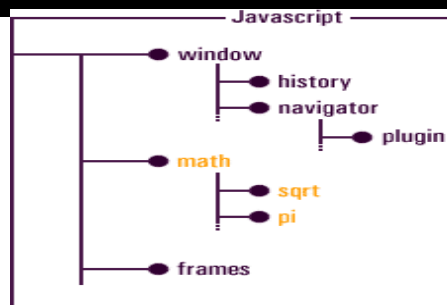
Structures mathématiques

Propriétés

➤ constantes mathématiques :
E, PI, SQRT2, LN2, LN10,...

Listes des méthodes

- `abs(x)`, `acos(x)`, `asin(x)`, `atan(x)`, `cos(x)`, `ln(x)`, `log(x)`, `round(x)`, `sin(x)`, `sqrt(x)`, `tan(x)` : applique la fonction appropriée à x
- `ceil(x)` : renvoie le plus petit entier supérieur ou égal à x
- `exp(x)` : renvoie e (exponentielle) à la puissance x
- `floor(x)` : renvoie le plus grand entier inférieur ou égal à x
- `max(x,y)` : renvoie la plus grande des valeurs de x et y
- `min(x,y)` : renvoie la plus petite des valeurs de x et y
- `pow(x,y)` : renvoie x à la puissance y
- `random(x)` : renvoie un nombre aléatoire compris entre 0 et 1



68



Fonctions mathématiques

Exemple


```

document.write(Math.random()+"<br>");
document.write(Math.min(5,4)+"<br>");
document.write(Math.exp(1)+"<br>");
document.write(Math.ceil(2.2)+"<br>");
document.write(Math.floor(2.2)+"<br>");
document.write(Math.round(2.99)+"<br>");
document.write(Math.pow(2,3));
  
```

```

.8012453357071934
4
2.718281828459045
3
2
3
8
  
```

69



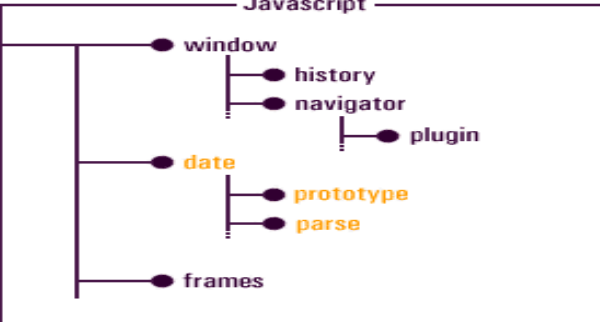
Objet Date

Représentation des dates Propriété

length (valeur initiale 7)

Nécessité d'instancier avec new

```
var X = new Date() --> date du jour
var Y = new Date(args)
```



```
<html>
<body>
  <script language = "javascript">
    var aujourd'hui = new Date();
    var maDate = new Date("November 24, 1981");
    var jour = maDate.getDate(); // jour vaut 24.
    document.write("Nous étions le " +
      jour + "/" + (maDate.getMonth() + 1) + "/" + maDate.getFullYear() + "<br>");
    document.write("Nous sommes le " +
      aujourd'hui.getDate() + "/" + (aujourd'hui.getMonth() + 1) + "/" + aujourd'hui.getFullYear());
  </script>
</body></html>
```

Nous étions le 24/11/1981
 Nous sommes le 06/12/2022



Objet Date

- Méthodes

Elles permettent d'extraire diverses informations d'un objet **Date**

- **set....()** : pour transformer des entiers en Date
- **get....()** : pour transformer en date et heure des objets Date
- **to...()** : pour retourner une chaîne de caractères correspondant à l'objet Date
- après les préfixes **set** et **get** ,
on peut mettre Year, Month, Date , Hours, Minutes, Seconds pour obtenir respectivement : nombre d'années depuis 1900, le numéro du mois, le N° du jour dans le mois, et les heures, minutes et secondes.
- **getDay()** donne le N° du jour de la semaine (le 0 tombe le dimanche)
- **getTime()** donne le nombre de millisecondes, très pratique pour calculer des intervalles entre 2 dates.

Objet Date : méthodes		
<u>Accès (récupération)</u>	<u>Affectation</u>	<u>Valeurs</u>
getDate()	SetDate ()	1-31
GetDay()	SetDay()	0-6
GetMonth()	SetMonth()	0-11
GetYear()	SetYear()	Retourne les 2 derniers chiffres de l'année (aa)
GetFullYear()	SetFullYear()	Retourne l'année (aaaa)
GetHours()	SetHours()	0-23
GetMinutes()	SetMinutes()	0-59
GetSeconds()	SetSeconds()	0-59
GetMilliseconds()	SetMilliseconds()	0-999
getTime()	SetTime()	Nombre de ms depuis 1.1.1970
getTimezoneOffset()	SetTimezoneOffset()	Différence en minutes entre heure locale et GMT

Objet Date : méthodes	
<ul style="list-style-type: none"> ➤ toGMTString() : <i>conversion en date GMT</i> ➤ toLocaleString() : <i>conversion en heure locale</i> ➤ UTC(an,mois,jour,H,M,S) : <i>conversion en variable date (GMT)</i> ➤ Parse("December 25, 1998") : <i>conversion en variable Date</i> ➤ setTimeout("quelle_heure()", 1000) : <i>affichage de l'heure à chaque seconde (1000 ms)</i> <p>quelle_heure() : fonction personnelle qui calcule l'heure</p>	



Objet Array

Tableau d'éléments

Propriété

length nombre d'éléments

Méthodes

join() concatène tous les éléments en une chaîne de caractères séparés par une virgule ou un séparateur passé en argument

reverse() transpose le tableau

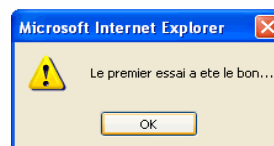
sort() trie les éléments ou trie selon la relation d'ordre passée en argument



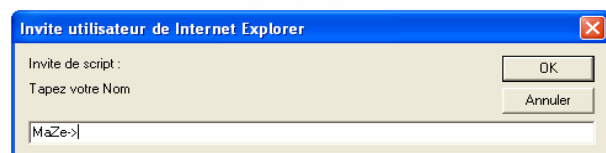
Boîtes de dialogue

Parmi les objets JavaScript du navigateur, on distingue 3 méthodes de type fenêtre (objet window) qui proposent des fenêtres prédéfinies pour faire des Entrées/Sorties :

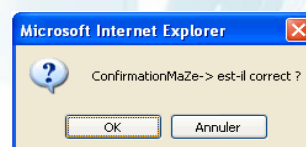
➤ **alert** : fenêtre d'affichage d'un message avec un bouton de fermeture

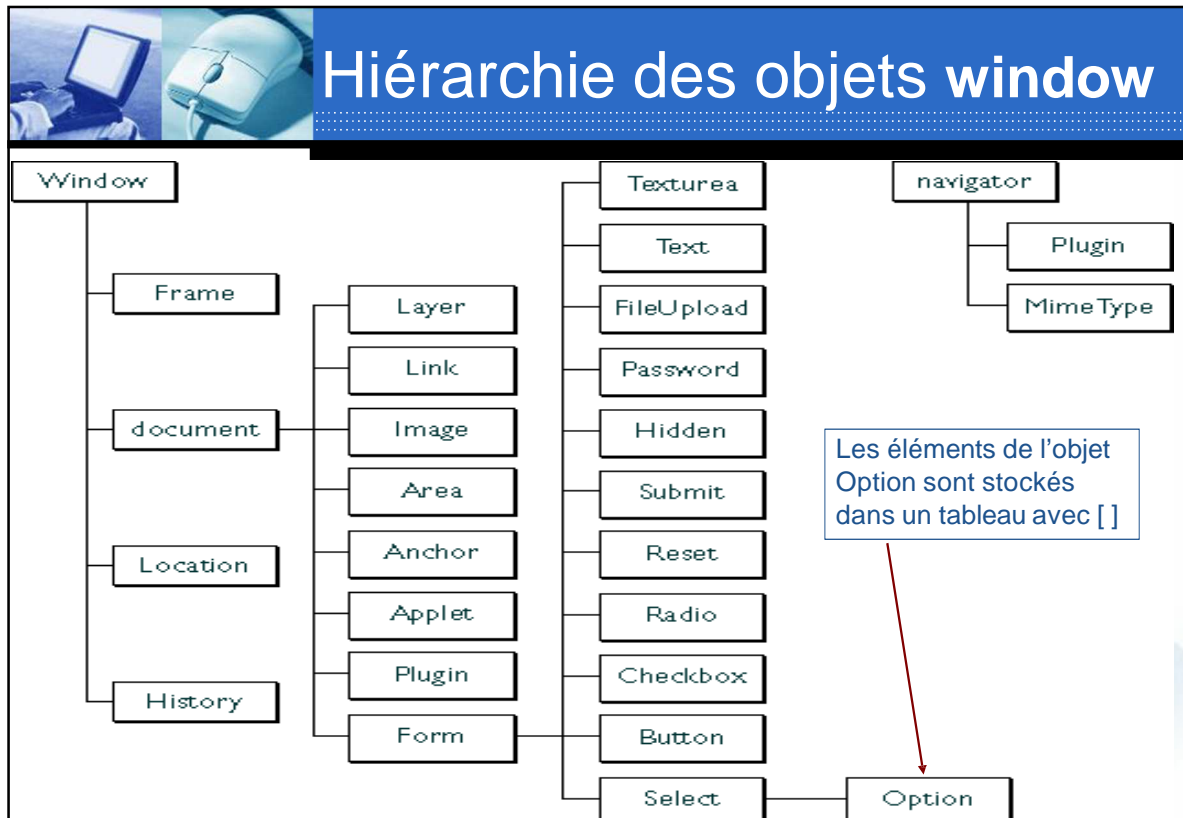


➤ **prompt** : fenêtre de saisie d'une information avec un champ de saisie, et 2 boutons pour valider, et annuler



➤ **confirm** : fenêtre de test avec 2 boutons pour valider ou annuler



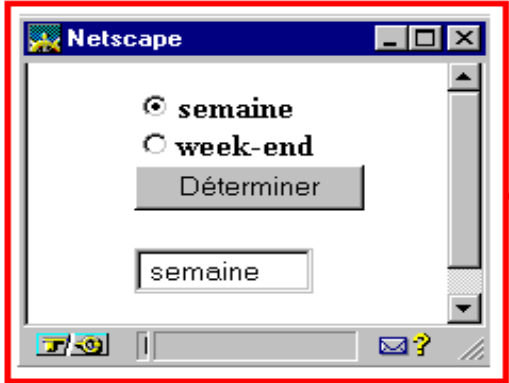


L'objet Window

- Imaginons la page HTML suivante

L'objet Window

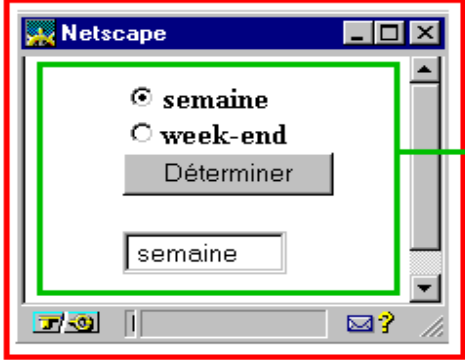
- Cette page s'affiche dans une fenêtre.
C'est l'**objet fenêtre**(window)



objet fenêtre

L'objet Window

- Dans cette fenêtre, il y a un document Html.
C'est l'**objet document**. Autrement dit (et c'est là que l'on voit apparaître la notion de la hiérarchie des objets JavaScript), l'**objet fenêtre** contient l'**objet document**.

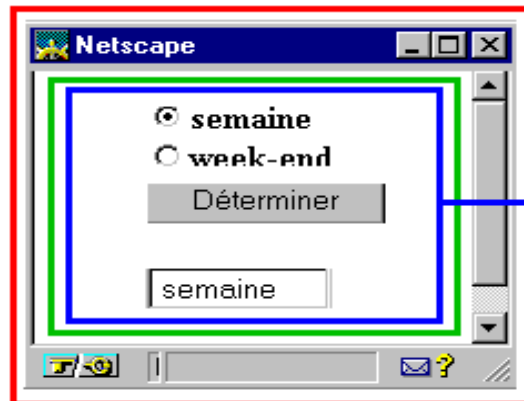


objet document



L'objet Window

- Dans ce document, on trouve un formulaire au sens Html. C'est l'**objet formulaire**. Autrement dit, l'**objet fenêtre** contient un **objet document** qui lui contient un **objet formulaire**.

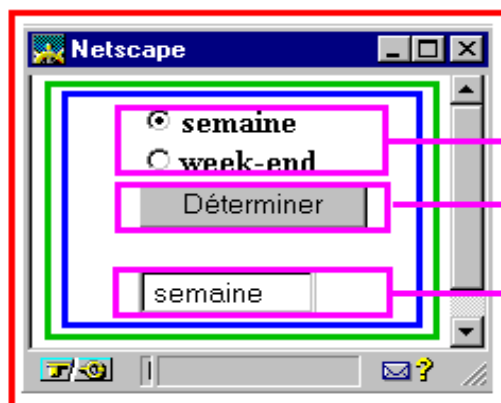


objet formulaire



L'objet Window

- Dans ce document, on trouve trois objets. Des boutons radio, un bouton classique et une zone de texte. Ce sont respectivement l'**objet radio**, l'**objet bouton**, l'**objet texte**.



objet radio

objet bouton

objet texte



L'objet Window

- L'accès à un objet se fait via le chemin qui passe par la hiérarchie dont il fait partie
- Ainsi, on peut accéder à tout objet
- – Ex. `window.document.formulaire.texte`
fait référence à la zone de texte "texte"

Valeur du champ texte:

`window.document.formulaire.texte.value`

- De même, nous pouvons accéder aux valeurs des deux boutons radio que nous avons :
 - `window.document.formulaire.radio.value`
fait référence aux 2 boutons radio avec leurs name radio



DOM : objet window

▪ Modèle Objet de Document (DOM)

Quid ?

- ☐ Modèle associé à un l'environnement client

But

- ☐ Permettre la manipulation des objets :
 - de l'interface
 - du document
 (objets créés automatiquement par le navigateur)

Types d'objets

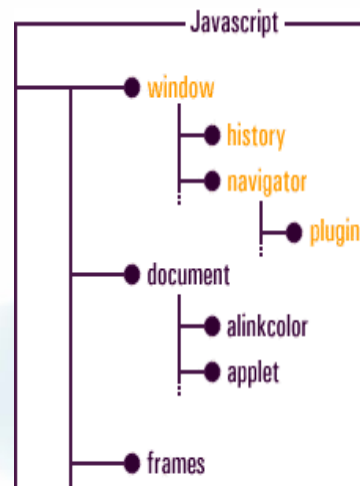
- ☐ Window, Document, Form, Browser , ...



DOM : objet Window

- L'objet window (fenêtre d'affichage) est l'objet par excellence dans Javascript, car il est le parent de chaque objet qui compose la page web, il contient donc:

- l'objet **document**: la page en elle-même
- l'objet **location**: le lieu de stockage de la page
- l'objet **history**: historique des pages visitées
- l'objet **frames**: les cadres



DOM : objet window

Propriétés

- ❑ **frames[]** : tableau de frames
- ❑ **frames.length** : nombre de frames
- ❑ **self** : fenêtre courante
- ❑ **opener** : la fenêtre (si elle existe) qui a ouvert la fenêtre courante
- ❑ **parent** : parent de la fenêtre courante, si la fenêtre courante est une sous-partie d'un frameset
- ❑ **top** : fenêtre principale (qui a créé toutes les fenêtres)
- ❑ **name** : nom de la fenêtre



DOM : objet Window

■ Méthodes

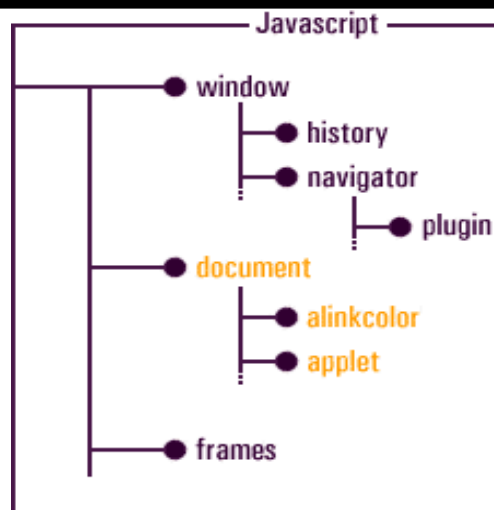
- `alert(string)` : ouvre une boîte de dialogue avec le message passé en paramètre
- `confirm()` : ouvre une boîte de dialogue avec les boutons OK et cancel
- `blur()` : enlève le focus de la fenêtre
- `focus()` : donne le focus à la fenêtre
- `prompt(string,string)` : affiche une fenêtre de saisie
- `scroll(int x, int y)` : positionnement aux coordonnées (x,y)
- `open(URL, string name, string options)` : ouvre une nouvelle fenêtre contenant le document identifié par l'URL
- `close()` : ferme la fenêtre



DOM : objet Document

■ Propriétés

- `title` : titre du document
- `location` : URL du document
- `lastModified` : date de dernière modification
- `referrer` : URL de la page d'où arrive l'utilisateur
- `bgColor` : couleur de fond
- `fgColor` : couleur du texte
- `linkColor`
`vlinkColor`
`alinkColor`



couleurs utilisées pour les liens hypertextes



DOM : objet Document

■ Propriétés

- forms[] : tableau des formulaires de la page
- forms.length : nombre de formulaire(s) de la page
- links[] : tableau des liens de la page
- links.length : nombre de lien(s) de la page
- anchors[] : tableau des ancres internes ()
- anchors.length : nombre d'ancre(s) interne(s)

- images[]
- applets[]
- embeds[]

tableaux des images, applets et plug-ins

Remarque : les tableaux contiennent les éléments dans l'ordre de leur apparition dans le code HTML



DOM : objet Document

■ Méthodes

- write(string) : écrit une chaîne dans le document
- writeln(string) : idem + caractère de fin de ligne
- clear() : efface le document
- close() : ferme le document



DOM : objet Form

■ Propriétés

- `name` : nom (unique) du formulaire
- `method` : méthode de soumission (0=GET, 1=POST)
- `action` : action déclenchée par la validation du formulaire
- `target` : fenêtre de destination de la réponse (si elle existe)
- `elements[]` : tableau des éléments du formulaires
- `elements.length` : nombre d'éléments du formulaire




DOM : objet Form

■ Méthodes

- `submit()` : soumet le formulaire
- `reset()` : ré-initialise le formulaire

■ Événements

- `onSubmit(method)` : action à réaliser lorsque le formulaire est soumis
- `onReset(method)` : action à réaliser lorsque le formulaire est ré-initialisé



Exemple d'utilisation du DOM

- Exemple de formulaire


```
<form name="general">
  <input type="text" name="champ1" id="champ1"
        value="valeur initiale" >

  </form>


<form ...>
</form>

...
```

Accès à l'objet correspondant au formulaire précédent

☐ plusieurs possibilités :

```
document.forms['general']
document.forms[0]
document.general
```



Exemples d'utilisation du DOM

- Accès aux éléments du formulaire
 - plusieurs possibilités :


```
document.forms['general'].elements['champ1']
document.forms['general'].elements[0]
document.forms['general'].champ1
document.general.champ1
document.getElementById('champ1')
document.getElementsByName('champ1')[0]
document.getElementsByTagName('input')[0]
```
 - Accès aux propriétés d'un élément par **.value**

```
document.general.champ1.value
document.getElementById('champ1').value
document.getElementsByName('champ1')[0].value
document.getElementsByTagName('input')[0].value
```



Exemples d'utilisation du DOM

■ Appeler une méthode sur un objet

- Pour donner le focus, par exemple :

```
document.forms['general'].elements['champ1'].focus();
```

■ Associer une action à un événement

```
<FORM name="changer">
  <INPUT type="text" name="zonetexte" value="Valeur initiale">
  <INPUT type="button" value="Changer la valeur"
onClick="document.forms['changer'].elements['zonetexte'].value='NOUVEAU' ">
</FORM>
Ou // onClick= " document.changer.zonetexte.value ='NOUVEAU' " >
Ou // onClick= " document.getElementsByName('zonetexte')[0].value='NOUVEAU' " >
```




Validation d'un formulaire par JavaScript

■ Intérêt

- Vérifier les données saisies avant de valider réellement le formulaire.

■ Principe

- Création d'un bouton de **type "button"** (et pas de type "submit") pour soumettre le formulaire.
- Association d'une fonction Javascript qui
 - contrôle la saisie,
 - soumet ou non le formulaire,
 à l'événement **onClick** de ce bouton.




Validation d'un formulaire par JavaScript

```

<html><head> <script language="javascript">
function validermail(formulaire) {
if(formulaire.mail.value.indexOf("@",3)<0)
    { alert("adresse mail saisie invalide.\n");
      // le formulaire ne sera pas validé
    }
else { alert("formulaire ok\n");
      // pour valider le formulaire en javascript :
      formulaire.submit();
    }
}
</script> </head>
<body>
<form name="coordonnees" action="http://www.google.fr">
  saisissez une adresse mail valide (nom@domaine) :
  <input type="text" name="mail">
  <input type="button" name="bouton" value="valider"
    onclick="validermail(this.form)">
</form>
</body></html>

```



Validation d'un formulaire par JavaScript

Saisissez une adresse mail valide (nom@domaine) :

ss@

file://

Adresse mail saisie invalide.

Saisissez une adresse mail valide (nom@domaine) :

sse@

file://

Formulaire OK



innerHTML : Écrire dans un élément HTML
document.write () : après le chargement d'un document HTML supprimera tout le HTML existant

```

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<script language="javascript">
function fonctionId(p) {
var p=form1.texte1.value;
document.getElementById("prenom").innerHTML="Bonjour "+p+"<br>"+"Bonne journée !";
}
</script>
</head>

<body>
<form name="form1">
<input type="text" name="texte1">
<input type="button" value="Salut" onclick="fonctionId(texte1);">
<div id="prenom"></div>
</form>
</body>
</html>

```

Thomas

Salut

Bonjour Thomas

Bonne journée !



Rappel sur les formulaires

- **Langage HTML**
 - Déclaration de formulaire : `<FORM ...> ... </FORM>`
 - Élément(s) d'un formulaire : `<INPUT ...>`
- **But**
 - Interaction avec l'utilisateur sous la forme d'une saisie d'informations.
- **Intérêt de JavaScript**
 - Augmenter l'interaction du côté client, par exemple pour :
 - assister et guider le visiteur,
 - contrôler la saisie,
 - réaliser des traitements (calcul, ...),
 - envoyer des résultats au serveur.
- Événements associés à `<FORM ...>`
- `onSubmit = " ... "`
 - Détecte la soumission du formulaire
- `onReset = " ... "`
 - Détecte la réinitialisation du formulaire



Événements associés à `<input type = "text">`

Élément `<INPUT type="text" ...>`

Propriétés acceptées

- name, value, defaultvalue, size, maxlength, disabled, readOnly, class, style

Méthodes acceptées

- focus, blur

Événements acceptés

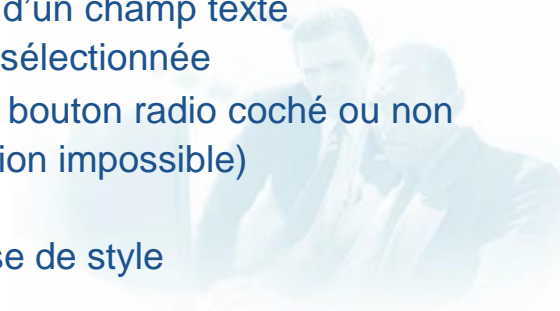
- onFocus, onBlur, onChange




`<input type = "...." >`

■ Propriétés



- name : nom du champ
- type : type du champ (text, button, radio, checkbox, submit, reset)
- value : valeur textuelle
- size : taille du champ
- maxlength : taille maximale d'un champ texte
- selected : texte ou option sélectionnée
- checked : case à cocher / bouton radio coché ou non
- disabled : grisé (modification impossible)
- readOnly : lecture seule
- class : nom de la classe de style
- style : nom du style





<input type = "...." >

- Méthodes
 - focus() : donne le focus (curseur)
 - blur() : enlève le focus
 - click() : simule un click
 - select(): sélectionne le champ
- Événements
 - onFocus : détecte la prise de focus
 - onBlur : détecte la perte de focus
 - onClick : détecte un click
 - onChange : détecte les changements

Exemple

```

<!DOCTYPE html>
<html>
<body>
<form>
<p>input a un nombre entre 1 et 10:</p>
<input id="numb">
<input type="button" value="ENVOYER" onclick="myFunction()">
<p id="demo"></p>
</form> <script>
function myFunction() {
  var x, text;
  // x est la valeur de input avec id="numb"
  x = document.getElementById("numb").value;
  if (isNaN(x) || x < 1 || x > 10) {
    text = "Input non valide";
  } else {
    text = "Input OK";
  }
  document.getElementById("demo").innerHTML = text;
}
</script> </body></html>
  
```

input a un nombre entre 1 et 10:

Input OK

input a un nombre entre 1 et 10:

Input non valide



L'objet Window - Méthode open : les pop up

window.open("URL", "nom", "options")

window.open("URL", "Nouvelle_fenetre", "options_de_la_fenetre");

- URL : l'url de la page qui sera affichée dans la nouvelle fenêtre, c'est-à-dire l'emplacement physique de celle-ci.
- Nouvelle_fenetre : nom de la fenêtre pop-up sans espace

options

directory

description (par défaut la valeur des options = yes)

= yes/no Affiche ou non les boutons de navigation

location

= yes/no Affiche ou non la barre d'adresse

menubar

= yes/no Affiche ou non la barre de menu (fichier, édition, ...)

resizable

= yes/no Définit si la taille de la fenêtre est modifiable ou non

scrollbars

= yes/no Affiche ou non les ascenseurs (barres de défilement)

toolbar

= yes/no Affiche ou non la barre d'outils

width, height

= largeur, hauteur (en px)

top = px, left = px

= positionnement vertical et horizontal




window.open(URL [, nom],[,options])

▪ Exemple

```
// Popup minimaliste, position fixe en haut à gauche
window.open('popup.html','','top=10,left=10')

// Aucune barre de menu, non redimensionnable,taille fixe
window.open('popup.html', '',
'resizable=no, location=no, menubar=no,
scrollbars=no, width=200, height=100')

// Popup fullscreen
window.open('popup.html','','fullscreen=yes')
```




window.open(URL [, nom][,options])

- Exemples de déclenchements d'ouvertures de fenêtre


```
// ouverture fenêtre au chargement d'une page <body>
<BODY onLoad="window.open( ... )">
...

// ouverture fenêtre à la fermeture d'une page
<BODY onUnload="window.open( ... )">
...
```
- Fenêtre toujours visible


```
// forcer le focus
<BODY onBlur="window.focus()" >
...
```



L'objet history

history :

Contient l'historique des pages visitées via le navigateur,

Propriétés et méthodes

- length : nombre d'objets dans l'historique
- méthode back : aller à l'URL précédent dans l'historique
- méthode forward : aller à l'URL suivant dans l'historique
- méthode go(variable) : aller à un des URL de l'historique.

Où variable : nombre entier qui détermine le nombre de pages relatif auquel se trouve l'URL désiré, ou chaîne de caractère

JavaScript

- window
 - history
 - navigator
 - plugin
 - history
 - current
 - length
- frames



L'objet history

Avancer et reculer

- Pour reculer dans l'historique, il vous suffit de faire : `window.history.back();`

Cela agira exactement comme si l'utilisateur cliquait sur le bouton Retour de la barre d'outils de son navigateur.

- De la même manière, il est possible d'avancer (comme si l'utilisateur cliquait sur le bouton Avancer) : `window.history.forward();`

- Se déplacer à un élément précis de l'historique

Utiliser la méthode `go()` pour charger une page dans l'historique identifiée par sa position relative par rapport à la page en cours (la page courante étant, évidemment, d'index 0).

- Pour reculer d'une page `back()` équivaut à : `window.history.go(-1);`
- Pour avancer d'une page `forward()` équivaut à : `window.history.go(1);`

De la même manière, vous pouvez avancer de 2 pages en passant la valeur 2, et ainsi de suite. Vous pouvez déterminer le nombre de pages de l'historique en accédant à la valeur de la propriété `length` (*longueur*) :

`Var nb_page= window.history.length;`



Objet Navigator

■ Propriétés

- `appName` : nom de code interne du navigateur
retourne Mozilla
- `appName` : nom réel du navigateur retourne Netscape
- `appVersion` : version du navigateur
- `userAgent` : des détails sur les :
 - l'`appName`,
 - l'`appVersion`
 - le système d'exploitation utilisé
- `plugins[]` : tableau des plugins installés chez le client pour lire certains éléments de la page (comme applet Java)

■ Méthodes

- `javaEnabled` : retourne TRUE si le navigateur supporte Java (et que l'exploitation de Java est actif)

L'élément de sortie <output>

<!DOCTYPE html>
 <html>
 <body>

0100 +

=

L'élément de sortie représente le résultat d'un calcul. </p>
 <form action = "/exemple.html" onInput = "x.value = parseInt(a.value) + parseInt(b.value)">
 0 <input type = "range" id = "a" name = "a" value = "50" min="0" max="1000" > 100 +
 <input type = "number" id = "b" name = "b" value = "50"> =
 <output name = "x" for = "a b"> </output>

 L'élément de sortie représente le résultat d'un calcul.
 </form>
 </body>
 </html>

0100 +

= 1050

Une **API** (application programming interface ou « interface de programmation d'application ») est une interface logicielle qui permet de « connecter » un logiciel ou un service à un autre logiciel ou service afin d'échanger des données et des fonctionnalités.

API de validation JavaScript

Méthodes DOM de validation des contraintes :

checkValidity() : Renvoie true si un élément d'entrée contient des données valides.

setCustomValidity(): Définit la propriété validationMessage d'un élément d'entrée.

Si un champ de saisie contient des données non valides, affichez un message:

```

<!DOCTYPE html><html><body><p>Entrer un nombre et cliquer sur OK:</p><form>
<input id="id1" type="number" min="10" max="30" required>
<input type="button" onclick="myFunction()" value="OK"></form>
<p>Si le nombre est inférieur à 10 ou supérieur à 30,
un message d'erreur s'affiche.</p>
<p id="demo"></p>
<script>
function myFunction() {
  var inpObj = document.getElementById("id1");
  if (inpObj.checkValidity()) {
    document.getElementById("demo").innerHTML = "Input OK";
  } else {
    document.getElementById("demo").innerHTML = inpObj.validationMessage;
  }
}
</script> </body></html>

```

Entrer un nombre et cliquer sur OK:

Si le nombre est inférieur à 10 ou supérieur à 30, un message d'erreur s'affiche.

Entrer un nombre et cliquer sur OK:

Si le nombre est inférieur à 10 ou supérieur à 30, un message d'erreur s'affiche.

Veuillez sélectionner une valeur inférieure ou égale à 30.

Entrer un nombre et cliquer sur OK:

Si le nombre est inférieur à 10 ou supérieur à 30, un message d'erreur s'affiche.

Input OK



API validation de contrainte

- **Validity** : Contient des propriétés booléennes liées à la validité d'un élément d'entrée.
- **validationMessage** : Contient le message qu'un navigateur affichera lorsque la validité est fausse.
- **willValidate** : Indique si un élément d'entrée sera validé.

Propriétés de validité

- La **propriété de validité** d'un élément d'entrée contient un certain nombre de propriétés liées à la validité des données :
- **customError** : Définit true, si un message de validité personnalisé est défini.
- **patternMismatch** : Définit true, si la valeur d'un élément ne correspond pas à son attribut pattern.
- **rangeOverflow** : Définit true, si la valeur d'un élément est supérieure à son attribut max.
- **rangeUnderflow** : Définit true, si la valeur d'un élément est inférieure à son attribut min.
- **stepMismatch** : Définit true, si la valeur d'un élément n'est pas valide par son attribut step.
- **tooLong** : Définit true, si la valeur d'un élément dépasse son attribut maxLength.
- **typeMismatch** : Définit true, si la valeur d'un élément n'est pas valide selon son attribut de type.
- **valueMissing** : Définit true, si un élément (avec un attribut obligatoire) n'a pas de valeur.
- **valid** : Définit true, si la valeur d'un élément est valide.



Exemple API

```
<!DOCTYPE html><html><body><form>
<p>Entrer un nombre et cliquer sur OK:</p>
<input id="id1" type="number" max="10">
<button onclick="myFunction()">OK</button>
<p>Si le nombre est supérieur à 10 (l'attribut max de l'entrée), un message d'erreur sera affiché.</p>
<p id="demo"></p></form>
<script>
function myFunction() {
  var txt = "";
  if (document.getElementById("id1").validity.rangeOverflow) {
    txt = "Valeur plus grande";
  } else {
    txt = "Input OK";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script></body></html>
```

Entrer un nombre et cliquer sur OK:

10 OK

Si le nombre est supérieur à 10 (l'attribut max de l'entrée), un message d'erreur sera affiché.

Input OK

Entrer un nombre et cliquer sur OK:

11 OK

Si le nombre est supérieur à 10 (l'attribut max de l'entrée), un message d'erreur sera affiché

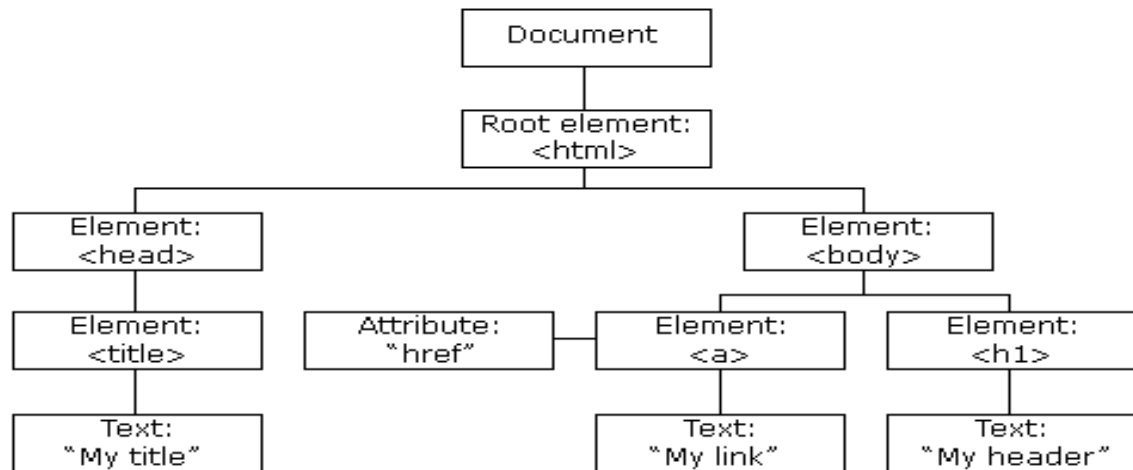
Valeur plus grande



DOM HTML (Document Object Model)

- Lorsqu'une page Web est chargée, le navigateur crée un **Document Objet Modèle** de la page.
- Le modèle **HTML DOM** est construit comme une arborescence d'**objets** :

L'arbre d'objets HTML DOM :



Le DOM

Avec le modèle objet, JavaScript obtient toute la puissance dont il a besoin pour créer du HTML dynamique:

- JavaScript peut modifier tous les éléments HTML de la page
- JavaScript peut modifier tous les attributs HTML de la page
- JavaScript peut changer tous les styles CSS de la page
- JavaScript peut supprimer les éléments et attributs HTML existants
- JavaScript peut ajouter de nouveaux éléments et attributs HTML
- JavaScript peut réagir à tous les événements HTML existants de la page
- JavaScript peut créer de nouveaux événements HTML dans la page

Le DOM HTML est un modèle d'**objet** standard et **une interface de programmation** pour HTML. Il définit:

- Les éléments HTML comme **objets**
- Les **propriétés** de tous les éléments HTML
- Les **méthodes** pour accéder à tous les éléments HTML
- Les **événements** pour tous les éléments HTML



Le DOM

- Le DOM permet de se représenter le document sous forme d'arborescence de balises
- Il permet de manipuler n'importe quel élément (balise) de notre page Web via les propriétés et les méthodes suivantes.
- Le DOM est la méthode d'accès aux éléments d'une page Web
- Le DOM fait très bon ménage avec CSS et XHTML
- Le DOM est lié à l'objet document (Premier nœud)
- En d'autres termes: **le DOM HTML est une norme pour savoir comment obtenir, modifier, ajouter ou supprimer des éléments HTML.**



Méthodes et attributs liés à l'objet Document

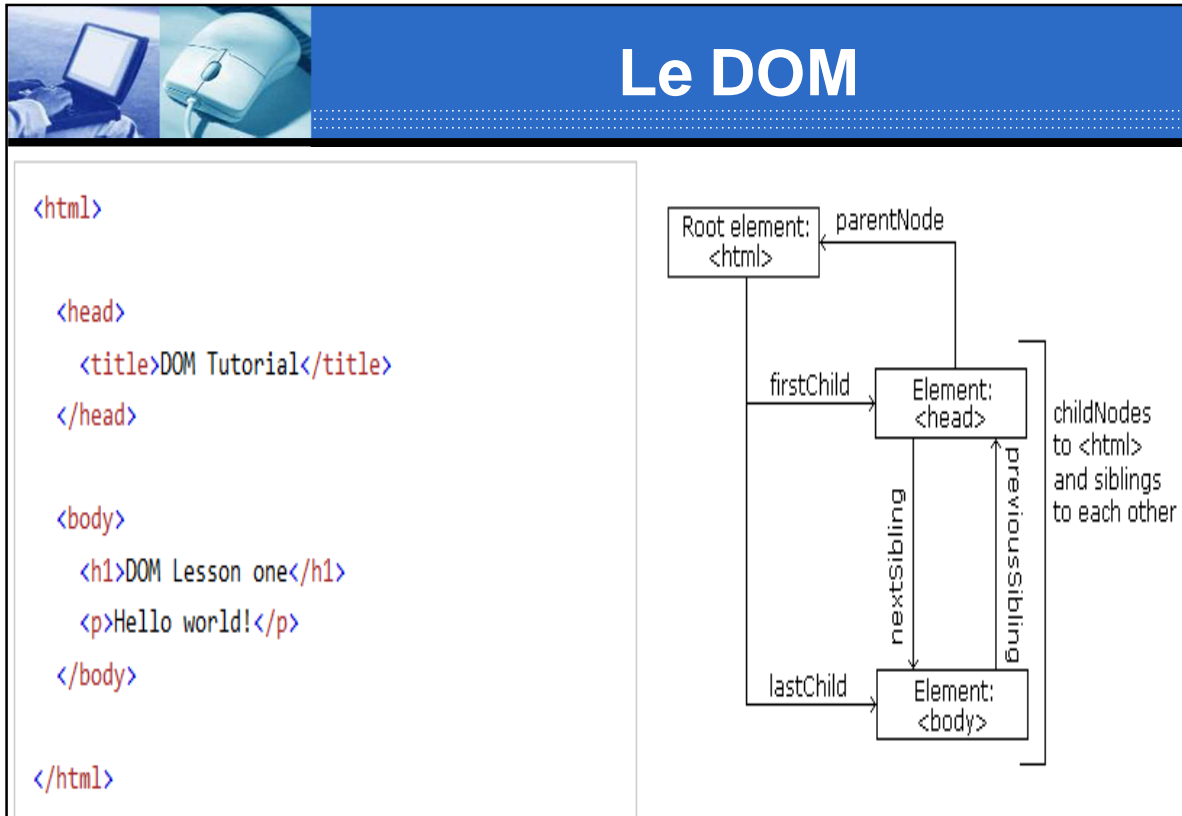
Avec le DOM HTML, JavaScript peut accéder et modifier tous les éléments d'un document HTML. Lorsqu'une page Web est chargée, le navigateur crée un modèle d'objet de document de la page. Le modèle DOM HTML est construit comme une arborescence d'objets:

Propriétés

parentNode (nœud parent du nœud courant)
childNodes (Tableau)
firstChild (premier nœud enfant)
LastChild (denier nœud enfant)

Méthodes

createAttribute() (créer un nœud d'attributs)
createElement() (créer un nœud d'éléments)
createTextNode() (créer un nœud de texte)
getElementById('nom_id').value (Accès à l'élément HTML par l'attribut Id)
getElementsByName('name_balise')[0].value
 (tableau d'éléments, Accès à l'élément HTML par l'attribut name de la balise)
getElementsByName('name_tag')[0].value
 (Accès au tag de l'élément HTML par liste d'éléments)
 Pour supprimer un élément il suffit de lui affecter la valeur spéciale: **null**



Le DOM

À partir du HTML ci-dessus, vous pouvez lire:

- <html> est le nœud racine
- <html> n'a pas de parent
- <html> est le parent de <head> et <body>
- <head> est le premier enfant de <html>
- <body> est le dernier enfant de <html>
- <head> a un enfant: <title>
- <title> a un enfant (un nœud de texte): "DOM Tutorial"
- <body> a deux enfants: <h1> et <p>
- <h1> a un enfant: "DOM Lesson one"
- <p> a un enfant: "Hello world!"
- <h1> et <p> sont frères et sœurs



Exemple manipulation DOM

```

<!DOCTYPE html><html><body>
<div id="div1">
<p id="p1">Voici un paragraphe.</p>
<p id="p2">Un autre paragraphe.</p>
</div>
<button onclick="myFunction()">Remove Element</button>
<script>
function myFunction() {
var elmnt = document.getElementById("p1");
elmnt.remove();
var h1 = document.createElement("h1");
var node = document.createTextNode("This is new.");
h1.appendChild(node);
var div1 = document.getElementById("div1");
var p2 = document.getElementById("p2");
div1.insertBefore(h1,p2); }
</script> </body></html>

```

Voici un paragraphe.


Un autre paragraphe.

Remove Element

This is new.

Un autre paragraphe.

Remove Element



Création de nouveaux éléments HTML (nœuds)

- Pour ajouter un nouvel élément au DOM HTML, vous devez d'abord créer l'élément (nœud d'élément), puis l'ajouter à un élément existant.
- Pour supprimer un élément HTML, utilisez la méthode: `remove()`

Ce code crée un nouvel élément `<p>`:

```
var h1 = document.createElement("h1");
```

Pour ajouter du texte à `<h1>` élément, vous devez d'abord créer un nœud de texte. Ce code crée un nœud de texte:

```
var node = document.createTextNode("This is new");
```

Ensuite, vous devez ajouter le nœud de texte à l'élément `<h1>`:

```
para.appendChild(node);
```

Enfin, vous devez ajouter le nouvel élément à un élément existant. Ce code trouve un élément existant:

```
var div1 = document.getElementById("div1");
```

Ce code ajoute le nouvel élément à l'élément existant:

```
div1.appendChild(h1);
```

appendChild() méthode de l'exemple précédent a ajouté le nouvel élément en tant que dernier enfant du parent.

Si vous ne le souhaitez pas, vous pouvez utiliser la méthode: **insertBefore()**



CSS (cascading style sheet)

- Idée de base séparé la structure de la présentation d'un document
 - HTML/XHTML structure le document (paragraphe, grand titre, éventuellement table ...)
 - CSS regroupe toutes les déclarations liées aux styles des éléments du document
- Dans les balises HTML on n'utilise plus que les attributs suivant : id, name, class
- Les tableaux ne sont pas des élément de mise en forme ... on préférera les calques (balises DIV)



Utilisation de CSS

- On peut regrouper les déclarations CSS dans un fichier.css ou entre les balises `<style>...</style>` dans l'entête de la page web (`<head>...</head>`)
- On peut redéfinir le style par défaut d'une balise HTML:

```
P {
  color: #666666;
  margin-left: 2 cm;
  border-top: 1px solid gray;
}
```




Utilisation de CSS

- On peut définir un style pour une balise HTML ayant un id spécifique:

```
#BoiteVerte {
  position : absolute;
  top : 20px;
  left : 100px
  color: green;
  border: 1px solid gray;
}
```

Appeler ce style en HTML

<div id="BoiteVerte"> le contenu de la boite verte </div>


- Le DOM permet entre autre de manipuler les styles CSS via la propriété style de chaque balise

```
document.getElementById("BoiteVerte").style.color="red";
```




Création des expressions régulières

- Lorsque vous recherchez des données dans un texte, vous pouvez utiliser un modèle de recherche pour décrire ce que vous recherchez.
- Nécessite la création d'un objet de type **RegExp**
var reg=new RegExp(motif, option)
 - Le paramètre **motif** est le cœur de l'expression, il définit le masque de la recherche, ce motif qu'on cherche à faire correspondre pour l'expression rationnelle.
 - Le paramètre **option** est une chaîne de caractères affinant l'action de l'expression

 Création des expressions régulières	
motif	Signification
^	Début de ligne ou de chaîne
\$	Fin de ligne ou de chaîne
.	N'importe quel caractère
[abc]	Groupe de caractères parmi ceux entre crochets
[a-z]	Groupe de caractères parmi les lettres minuscules de a à z
[A-Z]	Groupe de caractères parmi les lettres majuscules de A à Z

 var reg=new RegExp(motif, option)	
motif	Signification
[0-9]	Groupe de caractères parmi les chiffres de 0 à 9
[^0-9]	Groupe de caractères tous sauf de 0 à 9
(x)	Expression mémorisée
n*	Correspond à toute chaîne contenant 0 ou x occurrences de n
n+	Correspond à toute chaîne qui contient au moins un n
n?	Correspond à toute chaîne contenant 0 ou 1 occurrence de n

 var reg=new RegExp(motif, option)	
motif	Signification
{n}	Caractère précédent au moins n fois
{n,m}	Caractère précédent entre n et m fois
{n,m}	Expression mémorisée
\	N'est pas un caractère, sert de caractère d'échappement
\\	Caractère \
\d	Chiffre (équivalent à [0-9])

 var reg=new RegExp(motif, option)	
motif	Signification
\D	Sauf les chiffres (équivalent à [^0-9])
\b	Frontière de mot (espace, alinéa.....)
\s	Caractère d'espacement (espace, tabulation, saut de page...), équivalent à [\f\n\r\t\v]
\S	Un seul caractère sauf un espacement
\w	Recherche caractère word , n'importe quel caractère alphanumérique, y compris underscore _, équiv[A-Za-z0-9_]
\W	Tout sauf un caractère alphanumérique, équivalent à [^A-Za-z0-9_]



var reg=new RegExp(motif, option)

- Le paramètre **option** est une chaîne de caractère affinant l'action de l'expression, **option** peut prendre les valeurs suivantes :
 - "g" pour forcer une recherche globale sur toute la chaîne et poursuivre la recherche ou le remplacement dans toute la chaîne
 - "i" pour ignorer la distinction en majuscules et minuscules
 - "gi" pour combiner les options définies par "g" et "i"
 - "" chaîne vide pour n'appliquer aucune option

- **Exemple :** expression entre 3 et 10 caractères composés de lettres de a à z en minuscules ou majuscules sans accent et 0 à 9

```
var exp = new RegExp ("^[a-zA-Z0-9]{3,10}$ ", ""); // ou
var exp = new RegExp ("[a-z0-9]{3,10} ", "gi");
```



Exemple Surligne un mot

Exemple :

Chaîne d'origine : Un texte normal, et texte avec fond, Un texte normal, et texte avec fond
 Chaîne traitée : Un texte normal, et **texte avec fond**, Un texte normal, et **texte avec fond**

```
<HTML><HEAD><SCRIPT language="javascript">
var chaine="Un texte normal, et texte avec fond, Un texte normal, et texte avec fond ";
var reg1 = new RegExp("(texte avec fond)", "g");
document.write("Chaîne d'origine : " + chaine + "<BR>");
document.write("Chaîne traitée : " + chaine.replace(reg1,"<FONT><SPAN
style='background-color:gray'>$1</SPAN></FONT>") + "<BR>");
</SCRIPT></HEAD><BODY></BODY></HTML>
```

Explication :

Le motif (texte avec fond) de l'expression régulière permet de trouver tous les mots (texte avec fond).

Dans l'appel à replace(), le second paramètre indique comment remplacer texte avec fond.

Le symbole **\$1** représente la première expression entre parenthèse du motif.



\$1 à \$9 sont utilisés dans la méthode **replace()** pour modifier une chaîne

Exemple :

```
<SCRIPT language="javascript">
  var reg=new RegExp("((http://)[a-z0-9/.]+)","gi");
  var chaine="Cliquez sur ce lien <A href='http://www.google.fr'>http://www.google.fr ou sur celui-ci  !";
  document.write\("Chaîne d'origine : " + chaine + "<BR>"\);
  document.write\("Chaîne traitée : " + chaine.replace\(reg, "<A href='\$1' target=\_blank>\$1</A>"\) + "<BR>"\);
</SCRIPT>
```

Résultat :

Chaîne d'origine : Cliquez sur ce lien <http://www.google.fr> ou sur celui-ci <http://www.google.com> !

Chaîne traitée : Cliquez sur ce lien <http://www.google.fr> ou sur celui-ci <http://www.google.com> !

Explication :

Repère les sous-chaînes de caractères commençant par **http://** et composées de lettres, de chiffres, et des caractères possibles dans un url.

Remplace alors dans la chaîne ces sous-chaînes par les sous-chaînes trouvées encadrées de la balise **<A>** permettant le clic.



Méthode : **RegExp.exec()**
Retourne la première sous-chaîne correspondant au motif

Syntaxe :

String **reg.exec(String chaîne)**

exec() : Retourne la première sous-chaîne correspondant au motif et renvoie le texte trouvé en tant qu'objet.

Si aucune correspondance n'est trouvée, elle renvoie un objet vide (null).

Exemple :


Il y a 60 secondes dans 1 minute
60

```
<SCRIPT language="javascript">
  var reg=new RegExp("[0-9]+","g");
  var chaine="Il y a 60 secondes dans 1 minute";
  document.write(chaine+"<BR>");
  document.write(reg.exec(chaine));
</SCRIPT>
```

Explication :

Retourne la première sous-chaîne correspondant au motif.

Affiche le premier chiffre trouvé dans cette chaîne



Méthode : **RegExp.test()**

Teste l'expression régulière sur une chaîne


Exemple :

```
<SCRIPT language="javascript">
var reg=new RegExp("[0-9]{2}/[1]{1}[0-9]{2}/[1]{1}[0-9]{4}", "g");
var chaine1="15/12/2003";
var chaine2="1j/mm/2003";
document.write(chaine1+" ");
if (reg.test(chaine1)) {document.write("est bien au format date<BR>")}
else {document.write("n'est pas au format date<BR>")}
document.write(chaine2+" ");
if (reg.test(chaine2)) {document.write("est bien au format date")}
else {document.write("n'est pas au format date")}
</SCRIPT>
```

Explication :

15/12/2003 est bien au format date
1j/mm/2003 n'est pas au format date

test() : Teste l'expression régulière sur le paramètre chaîne.
Retourne **true** si la chaîne correspond au motif de l'expression régulière sinon retourne **false**.
Contrôle le format date et la date est valide : jj/mm/aaaa de chaîne1 et chaîne2



AJAX

Asynchronous JavaScript And XML

Contexte

- Pour rendre plus accessible les applications clientes, on s'est tourné vers les sites Web (intranet, extranet et Internet). Cela était nécessaire pour envisager la prestation électronique de services.
- Avec cette tendance irrésistible, la convivialité des applications a régressée. Avec le besoin d'accélérer et d'enrichir les pages Web, est arrivé **Ajax**.



Qu'est-ce que Ajax ?

- Ensemble de techniques de développement Web permettant de créer des « applications Web » interactives.
- Ajax est un acronyme pour **A**synchronous **J**avascript **A**nd **X**ML.
- Ainsi, Ajax n'est pas une nouvelle technologie, mais plutôt une nouvelle façon d'utiliser celles qui existaient déjà.



Qu'est-ce que Ajax ?

- AJAX est un type de programmation rendu populaire en 2005 par Google.
- AJAX n'est pas un nouveau langage de programmation, mais une nouvelle façon d'utiliser les standards existants.
- AJAX est basé sur JavaScript et HTTP.
- Concept inventé en 2004 et reposant sur des fondements bien plus anciens. **Le principe de base est d'intercepter en utilisant JavaScript les évènements survenant sur la page web, et d'insérer dynamiquement dans la page un contenu provenant d'un serveur web, véhiculé par un document XML, ou JSON toujours en utilisant JavaScript. La pierre angulaire de cette méthode est l'objet XMLHttpRequest qui permet à JavaScript d'effectuer une requête vers le serveur sans que l'utilisateur ne la voie, et ce de façon asynchrone.**



Avantages et inconvénients d'AJAX

Avantages :

- interactivité : les interfaces utilisant Ajax offrent une interactivité et une réactivité bien plus importantes que les pages habituelles, ou l'utilisateur doit attendre le rechargement complet de sa page.
- portabilité : tous les navigateurs actuels proposent l'ensemble des outils nécessaires à la mise en place d'un moteur Ajax

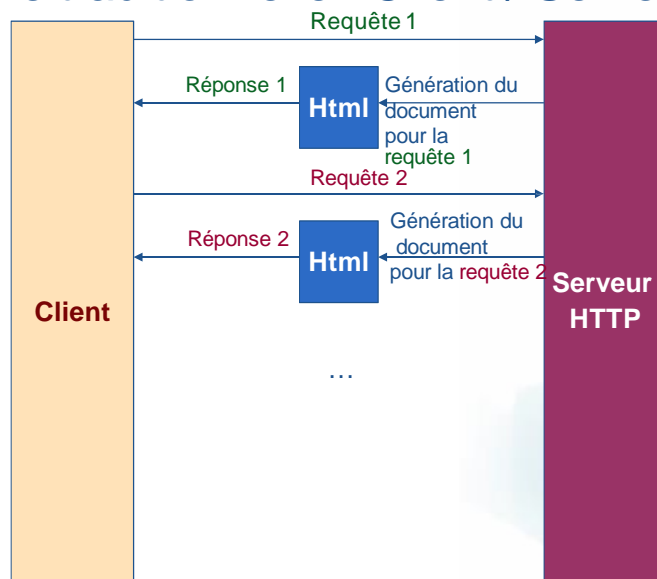
Inconvénients :

- ergonomie : l'utilisation d'Ajap entraîne une impossibilité pour l'utilisateur d'utiliser son bouton "Retour" de façon attendue. De la même façon, Ajax pose des problèmes pour la mise en place de signets (bookmarks) sur les pages, ainsi que pour l'indexation du contenu des pages.
- temps d'attente : les appels vers le serveur peuvent avoir des temps d'attente importants qui sont mal perçus et compris par les utilisateurs.
- utilisation de JavaScript : le moteur Ajax fait fortement appel au JavaScript. Il faut prévoir pour les utilisateurs ayant désactivé Javascript ou ne pouvant pas l'utiliser, une solution de repli acceptable.
- complexité des développements : comme tout composant additionnel Ajax offre des possibilités, mais la mise en place peut se révéler coûteuse au moment du développement.



Approche traditionnelle : Client / Serveur

Approche traditionnelle : Client / Serveur





Approche traditionnelle : Client / Serveur

- À chaque fois que l'utilisateur interagit avec la page, le navigateur doit envoyer une requête au serveur et attendre sa réponse avant de rafraîchir la page.
- Ce délai rend au mieux difficiles à implanter pour le développeur, sinon pénibles pour l'utilisateur beaucoup de choses qui sont pourtant monnaie courante dans les applications de bureau. Entre autres, certains événements, tel le mouvement de la souris, sont impensables à traiter dans la pratique.
- Ne serait-il pas intéressant d'accélérer le processus en, par exemple, téléchargeant d'avance les données susceptibles d'être consulté par la suite alors que l'utilisateur celles à l'écran



Asynchrone en JavaScript

En informatique, on dit que deux opérations sont **synchrones** lorsque la seconde attend que la première ait fini son travail pour démarrer. Ce qu'il faut retenir de cette définition est le concept de dépendance (la notion de « synchronisation » dans la première définition donnée de synchrone au-dessus) : le début de l'opération suivante dépend de la complétude de l'opération précédente.

Au contraire, deux opérations sont qualifiées d'**asynchrones** en informatique lorsqu'elles sont indépendantes c'est-à-dire lorsque la deuxième opération n'a pas besoin d'attendre que la première se termine pour démarrer.

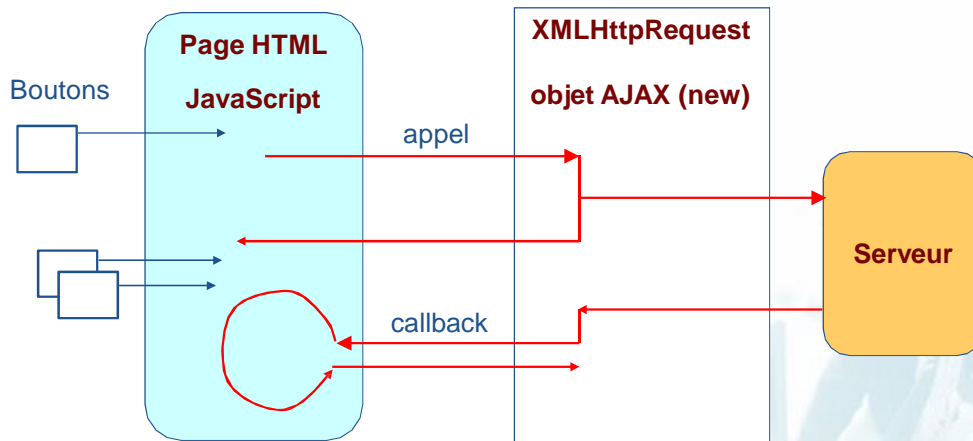
AJAX signifie *Asynchronous JavaScript and XML*. L'AJAX n'est pas un langage de programmation mais correspond plutôt à un ensemble de techniques utilisant des technologies diverses pour envoyer et récupérer des données vers et depuis un serveur de façon asynchrone, c'est-à-dire sans avoir à recharger la page.

AJAX permet d'envoyer et récupérer des données d'un serveur de manière asynchrone (en arrière-plan) sans interférer avec l'affichage et le comportement de la page existante. AJAX nous permet de modifier de manière dynamique le contenu d'une page, c'est-à-dire sans qu'il soit nécessaire de recharger l'intégralité de la page.



Qu'est-ce que Ajax ?

Appel AJAX Asynchrone

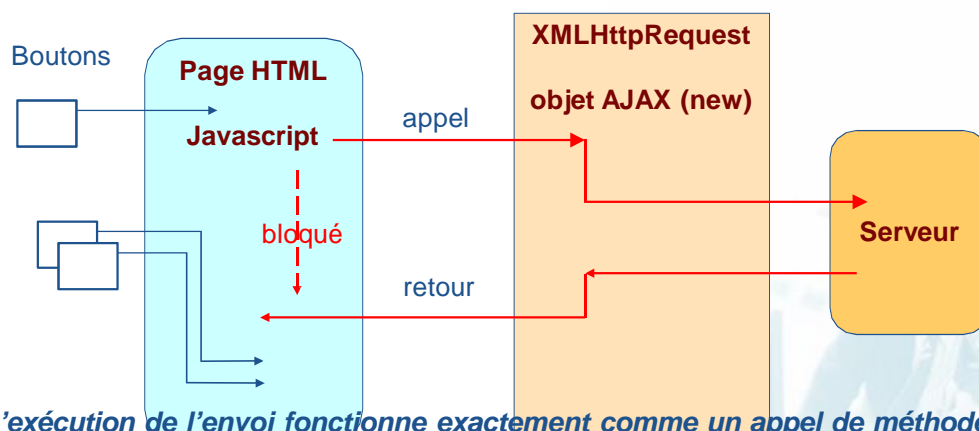


Le callback fonctionne exactement comme le timeout. Il appelle une fonction désignée par l'appelant. Une fonction de rappel (callback) est une fonction qui est passée en argument à une autre fonction, avec l'hypothèse que la fonction de rappel sera appelée à un moment adéquat



Qu'est-ce que Ajax ?

Appel AJAX synchrone



L'exécution de l'envoi fonctionne exactement comme un appel de méthode simple. Il faut ici noter que le navigateur exécute le programme ligne par ligne, selon l'ordre dans lequel elles ont été écrites. Pour chaque ligne, le moteur attend que la ligne ait été exécutée avant de passer à la prochaine. En effet, chaque ligne dépend du travail exécuté dans les lignes précédentes.

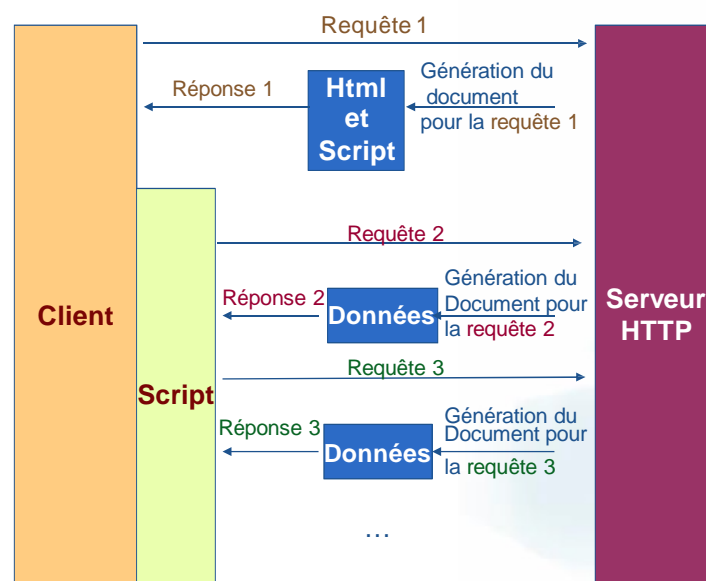


Approche asynchrone d'Ajax

- Ajax permet de faire une requête au serveur sans recharger la page. Ainsi, cela permet de ne rafraîchir qu'une partie de la page.
- Impression d'instantanéité pour l'utilisateur lorsque qu'implanté astucieusement (et sur un réseau assez rapide).
- La programmation asynchrone est une technique qui permet à un programme de démarrer une tâche à l'exécution potentiellement longue et, au lieu d'avoir à attendre la fin de la tâche, de pouvoir continuer à réagir aux autres événements pendant l'exécution de cette tâche. Une fois la tâche terminée, le programme en reçoit le résultat.



Approche asynchrone d'Ajax





Qu'est-ce que Ajax ?

- Repose sur des technologies et standards déjà connus et bien établis, entre autres : langage Javascript, objet XMLHttpRequest, format XML.
- Les techniques Ajax sont, en soi, indépendantes de la plateforme utilisée.
- Un grand nombre de cadres d'application (*frameworks*) sont disponibles et ceux-ci sont généralement compatibles avec les principaux navigateurs Web.



Comment fonctionne Ajax ?

- Le serveur HTTP envoie au client une page Web incluant un script.
- Le script utilise un objet **XMLHttpRequest**, ou fenêtre pop-up ou un autre moyen pour communiquer avec le serveur sans télécharger de nouveau la page.
- Le script met à jour la page.



Comment fonctionne Ajax ?

Objet : **XMLHttpRequest**

- Provient de Microsoft. Standard *de facto*: Implémenté par la plupart des principaux navigateurs Web.
- Certaines des premières API asynchrones utilisaient les événements de cette façon. L'API [XMLHttpRequest](#) permet d'envoyer des requêtes HTTP à un serveur distant en JavaScript. Étant donné qu'une requête peut prendre beaucoup de temps, il s'agit d'une API asynchrone, et on reçoit une notification sur l'avancement (voire la complétion) de la requête en attachant des gestionnaires d'événements à l'objet XMLHttpRequest.



Comment fonctionne Ajax ?

Principaux formats d'échange de données

- **XML**: Standard W3C. <http://www.w3.org/XML/>
- **JSON** (Javascript object notation): Format compact, facile à lire et écrire pour l'humain et facile à traiter pour l'ordinateur. <http://json.org/>
JSON est un format de stockage et de transport de données.
JSON est souvent utilisé lorsque des données sont envoyées d'un serveur à une page Web.

Exemple JSON :

```
{
  "employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
  ]
}
```

- Texte / HTML: Une requête peut, en fait, obtenir n'importe quel type de document.



Comment fonctionne Ajax ?

Considérations à ne pas perdre de vue lors de développements Ajax

- Accessibilité du contenu.
- Dégradation gracieuse *versus* amélioration progressive d'un site Web.
- Facilité d'entretien du code.
- Dans le contexte de la prestation électronique de services, Ajax peut contribuer à fournir des sites Web plus attrayants et conviviaux.



Comment fonctionne Ajax ?

- **Côté serveur**

Commande HTTP

<http://domaine.com/rep/view.php?id=376>

- adressant des fichiers HTML
- des cgi, des servlets, des asp (Microsoft) ou php...

L'appelant ne peut pas savoir par l'analyse des messages si ce sont des fichiers qui sont retournés ou des programmes qui répondent



Comment fonctionne Ajax ?

- Côté navigateur

- On peut traiter la réponse en l'interprétant comme une structure XML:

```
var http_request = new XMLHttpRequest();
...
var xml = http_request.responseXML;
xml.getElementsByTagName("aaaa")[0].firstChild.nodeValue
```

ou comme une structure Text:

```
var http_request = new XMLHttpRequest()
...
var x = http_request.responseText;
document.getElementById("Display").innerHTML = x
```



Lecture d'un arbre XML

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<data>
```

```
<aaaa>
```

```
  mmmm
```

```
    <bbbb>b111
```

```
  </bbbb>
```

```
    <bbbb>b222
```

```
  </bbbb>
```

```
    <bbbb>b333
```

```
  </bbbb>
```

```
</aaaa>
```

```
<aaaa>
```

```
  <cccc>c111
```

```
  </cccc>
```

```
  <cccc>c222
```

```
  ...
```

```
xml.getElementsByTagName("aaaa")[0]
    .firstChild.nodeValue: mmmm
xml.getElementsByTagName("bbbb")[1]
    .lastChild.nodeValue: b222
xml.getElementsByTagName("aaaa")[0]
    .getElementsByTagName("bbbb")[2]
    .firstChild.nodeValue: b333
```



Lecture d'un arbre XML

```
<?xml version='1.0' encoding='UTF-8'?>
```

```
<data>
```

```
<aaaa>
```

```
mm mm
```

```
<bbbb>b111
```

```
</bbbb>
```

```
nn
```

```
<bbbb>b222
```

```
</bbbb>
```

```
pp
```

```
<bbbb>b333 \nspace
```

```
</bbbb>
```

```
</aaaa>
```

```
<aaaa>
```

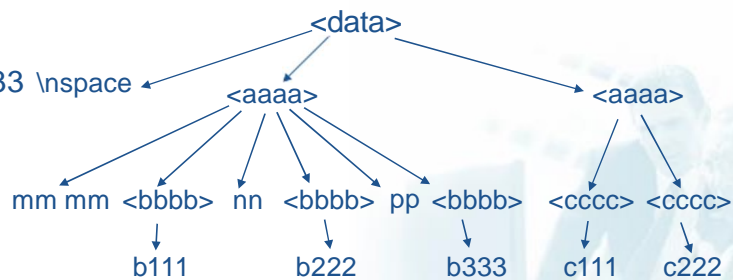
```
<cccc>c111
```

```
</cccc>
```

```
<cccc>c222
```

```
...
```

Arbre XML



Affichage de l'arbre XML

```
function print(indent, t) {
    if (t.nodeValue !== undefined) {           // soit affiche un nœud
        txt.push(indent + t.nodeValue)
    } else {                                   // soit parcourt les enfants
        txt.push(indent + "<-->" + "<" + t.tagName + ">")
        for (var j=0; j<t.childNodes.length; j++) {
            print(indent+"<\<\<\<", t.childNodes[j])
        }
    }
}
```



Asynchronous call JSON

```
function makeRequest(type, alertFunction) {
    // ...
    http_request.onreadystatechange = function() {
        alertFunction(http_request) // calls the user-defined function
    }
    http_request.open('GET', URL, true)
    http_request.send(null)
}

// user-defined function
var alertContents = function (http_local_request) {
    document.getElementById("Display").innerHTML
        = http_local_request.responseText
}

makeRequest("text.html", alertContents)

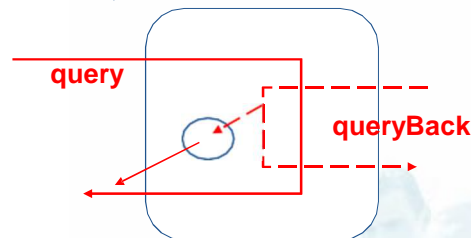
// On return, calls alertContent() to display returned value
```

asynchronous



Appels synchrone – asynchrone

```
// ...
DataBaseService.query(database.readyQuery,{
    callback: database.queryBack,
    errorHandler: database.handleDBError,
    async: false
})
if (database.error != null) {
    throw database.error
}
return database.data
}
/*
Call back function, just stores the result for the synchronous function
*/
database.data = null
database.queryBack = function (data) {
    database.data = data
}
```





Création d'un arbre XML

Ecrire un arbre est beaucoup plus facile que le lire, il suffit de concaténer les éléments de texte:

```
txt = [ ]
txt.push("<?xml version='1.0' encoding='UTF-8'?>")
txt.push("<data>")
txt.push("<aaaa>" + uneValeur)
...
File.write("toto.xml", txt.join("\n"))
```



Les sites intégrant AJAX

Exemples de sites utilisant Ajax:

- Gmail <http://www.gmail.com/>
- Google Maps <http://maps.google.com>
- Mappy <http://www.mappy.com>
- Ratp www.ratp.info/orienter/cv/carteparis.php
- Easyjet <http://www.easyjet.com>
- Netvibes <http://www.netvibes.com>



Exemple AJAX

Lorsque la page est rafraîchie, il y a un vote de plus et le nombre d'étoiles représente le score courant.



Après un minimum de 3 lettres, un maximum de 5 résultats sont affichés. Les résultats contiennent le nom de la personne et son extension.



Exemple AJAX

Si la personne recherchée est suggérée, on peut la sélectionner à l'aide de la souris (ou des flèches). Sinon, on peut préciser la requête en ajoutant des caractères.



Exemple avec moins de 5 résultats

Exemple sans résultat