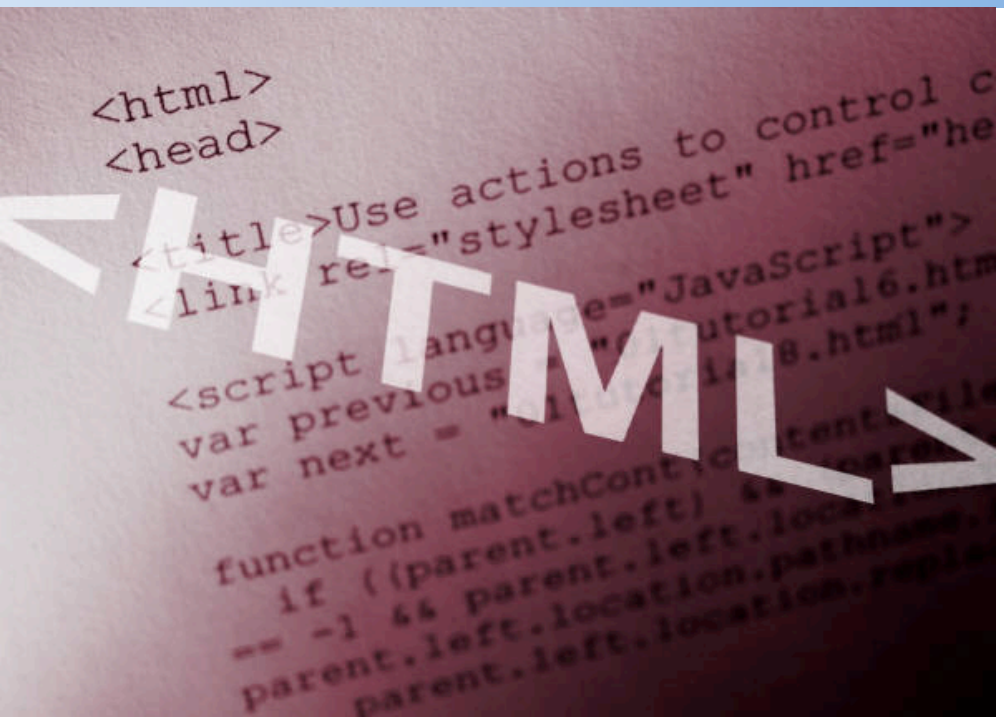
MODULE
CONCEPTION DE DOCUMENTS WEB



HTML, CSS

JAVASCRIPT, JQUERY,

AJAX

Plan du module

- DOM : vision arborescente d'un document HTML
- Javascript : langage de programmation côté client
- Fonctions, objets, variables, structures de contrôle, boucles,
- Accès au DOM
- Intégration dans le source HTML
- Javascript pour HTML5

DOM

Un document web = un fichier texte
= un arbre d'éléments

Composé de :

- Nœuds de type élément = élément HTML
- Nœuds de type texte = contenu
- Liens = composition des éléments

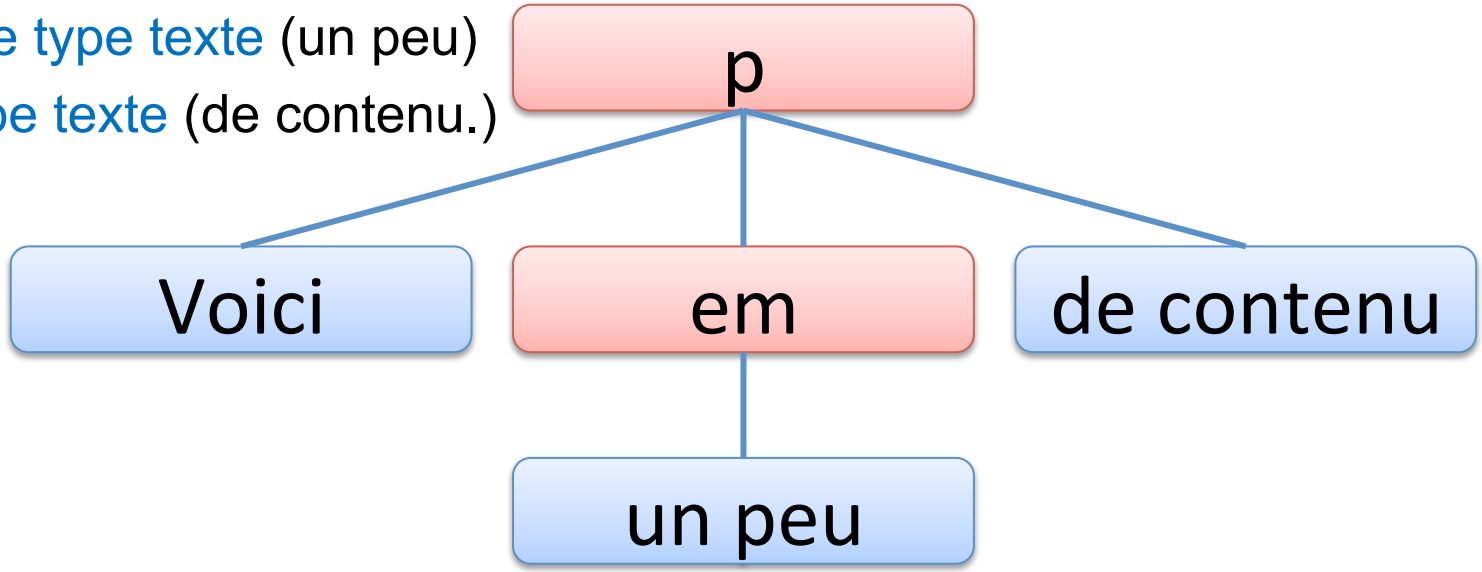
DOM

Exemple, l'extrait de code HTML suivant :

```
<p>Voici <em>un peu</em> de contenu</p>
```

Se décompose, selon le DOM en :

- Un nœud de type élément (p) contenant
  - un nœud de type texte (Voici)
  - un nœud de type élément (em) contenant
    - un nœud de type texte (un peu)
  - un nœud de type texte (de contenu.)



DOM

- Chaque « nœud » = un élément ou un texte à l'intérieur d'un élément.
- La racine du document se nomme « document », il s'agit d'un nœud.
- Tous les éléments du documents sont des nœuds « descendants » du nœud « document ».
- Chaque élément comprend lui même un nœud de type text nommé #text

DOM

- Il est possible d'interagir dynamiquement avec ces éléments (modifier / supprimer / créer / déplacer… un nœud) par programme
- Ce programme s'exécute sur la machine cliente
- Le langage qui permet ces manipulation s'appelle Javascript

Javascript

Javascript comporte :

- Une présentation structurée et « orientée objet » des éléments et du contenu d'une page.
- Des méthodes permettant d'accéder aux objets composants le document et de les modifier.
- Des méthodes permettant d'ajouter ou supprimer des objets.
- Des accès à certaines propriétés des objets
- Une gestion d'événements (click de souris, choix dans une liste, changement de la valeur d'un champ,….)

Javascript

Exemples d'utilisation :

- Vérifier la validité de la saisie (plage de valeur, date,…)
- Activer / désactiver des éléments (bouton OK actif uniquement si saisie valide)
- Afficher / cacher des éléments (coche verte visible uniquement si les deux saisies du mot de passe sont égales)
- Déplacer des éléments (animations)
- Modifier des ressources (changer le CSS d'une page)
- Créer /supprimer des éléments sur la page
- …

Javascript

- JavaScript = langage de programmation
- Exécuté par un interpréteur à l'intérieur d'un navigateur Web (donc sur le client !)
- introduit en V2 du navigateur de Netscape. Généralisé à tous les navigateurs depuis (sauf Lynx)
- Désactivable par l'utilisateur (règles de sécurité de l'entreprise, choix personnel,…)

Bases du langage

En javascript, tout est "objet" !

Deux types d'objets :

- les **objets de type primitif**, qui se résument à une valeur unique : les nombres, les booléens et les chaînes.
- les **objets de type composé** (tableaux, fonctions ou... objets) qui comportent plusieurs propriétés, chacune bénéficiant d'un nom et d'une valeur, l'ensemble réuni en une seule entité (l'objet), elle aussi identifiée par un nom.

Bases du langage

Autrement dit, un objet composé peut contenir :

- Des propriétés qui sont des objets de type primitif
- Des propriétés qui sont des tableaux
- Des propriétés qui sont des fonctions

Ex : l'objet "personne3" a une propriété "nom" qui vaut "Pierre", une propriété "alerter" qui est une fonction qui émet un message d'alerte avec le nom, une propriété "achats", de type tableau associant à une date un objet de type "voiture" etc.

Variables

Déclarer une variable JavaScript

⇨mot-clé « var »

```
var uneVariable = "Hello";
var uneAutreVariable = 100;
```

NB : il n'est pas obligatoire d'affecter une valeur à une variable lors de sa déclaration

```
var uneTroisiemeVariable;
```

variable non affectée (valeur = undefined)

Variables

Quelques exemples :

Instruction	Effet
`var entier;`	*Portée locale, déclaration explicite, type undefined, valeur undefined*
`var i = 0;`	*Portée locale, déclaration explicite, type numérique, valeur 0*
`valeur = i;`	*Portée globale, déclaration implicite, type numérique, valeur 0*
`entier = "0123";`	*Portée globale, déclaration implicite, type chaîne de caractères, valeur 0123*
`i = entier + valeur;`	Portée globale, i *devient la chaîne de caractères "01230". L'affectation a nécessité la conversion de* valeur *en "0" pour la concaténation.*

Types de données

Une variable peut contenir différents types de valeurs : un nom, une date, une heure, un prix

= des données de types primitifs / des instances d'objets

JavaScript prend en charge 5 types de données primitifs :

- **Nombre entier** : nombre positif ou négatif sans virgule
- **Nombre en virgule flottante** : nombre positif ou négatif avec virgule ou nombre écrit en notation scientifique
- **Valeur booléenne** : valeur logique vrai ou faux
- **Chaîne de caractères** : texte
- **Valeur Null** : pas de valeur

Typage dynamique

JavaScript est un langage à **typage dynamique**.

- Avec les langages de programmation à typage statique, lors de la déclaration d'une variable, il est nécessaire de donner le type de la variable. Le type de la variable ne change plus.
- Avec JavaScript, on ne peut pas déclarer le type de variable.
- L'interpréteur JavaScript détermine automatiquement le type en fonction des données stockées dans une variable.
- Le type de la variable est susceptible de changer chaque fois qu'on lui assigne une nouvelle valeur.
- L'opérateur typeof() permet d'obtenir le type d'une variable

Fonctions

- Mot clé réservé : function.

```
function uneFonction (parametre1, parametre2){
    instructions ...
}
```

- NB : les paramètres sont optionnels **y compris à l'appel de la fonction**

Fonctions

- Pour exécuter une fonction, elle doit être "invoquée", c'est a dire appelée en créant une commande du nom de la fonction.

- Les fonctions peuvent, ou non, retourner une valeur.

```
function calculeSomme(parametre1,parametre2){
    return parametre1 + parametre2;
}
```

Objets

- Les objets JavaScript se fondent sur des fonctions "constructeur".

~ définition d'un objet

- Lorsqu'un nouvel objet est créé à partir d'une fonction constructeur

⇨on crée une instance d'objet. L'instance d'objet a accès à l'ensemble des données et procédures de la fonction constructeur.

Objets : fonctions "constructeurs"

Elles comprennent :

- les propriétés
- les méthodes (voir plus loin)

Exemple :

```
function Etudiant (unNumero, unNom, unPrenom){
  this.numero = unNumero;
  this.nom = unNom;
  this.prenom = unPrenom;
}
```

NB : **this** fait référence à l'instance courante.

Objets : instanciation

L'instanciation se fait avec le mot-clé **new**.

```
unEtudiant = new Etudiant (45685, "Testeur", "Toto");
```

NB : il n'est pas nécessaire d'attribuer immédiatement une valeur aux propriétés.

```
unEtudiantBis = new Etudiant ();
```

Tant que les propriétés n'ont pas de valeur, si l'on essaie d'y accéder, la valeur **undefined** est retournée.

Il n'est sans doute pas souhaitable de le faire !

Objets : accès aux propriétés

Il y a deux façons de déclarer des propriétéss :

```
function Etudiant (nom, prenom){
  this.nom = unNom; // propriété PUBLIQUE
  this.prenom = unPrenom;
  var id = generateId(); // propriété PRIVÉE
}
```

Objets : accès aux propriétés publiques

On accède aux propriétés PUBLIQUES d'un objet avec le nom de l'instance de l'objet, suivi d'un **point**, suivi du nom de la propriété.

```
unEtudiant.nom
unEtudiant.prenom
```

Il est ainsi possible de **lire** la valeur de la propriété ou de lui affecter une valeur (**écrire**).

Objets : accès aux propriétés privées

Il faut des accesseurs, autrement dit des fonctions publiques qui renvoient la valeur de la propriété :

```
function Etudiant (nom, prenom){
    var id = generateId(); // propriété PRIVÉE
    this.getId = function() {
        return id ;
    };
}
```

Appel : `var sonId = unEtudiant.getId();`

`unEtudiant.id` ==> renvoie `undefined` !

Pour modifier la valeur de la propriété, il faut une fonction publique qui accepte une valeur en entrée et l'affecte à la propriété.

Objets : ajout de propriétés

Une fois une instance d'objet créée, il est possible de lui affecter de nouvelles propriétés (nécessairement publiques) : il suffit d'indiquer un nom et une valeur.

```
unEtudiant.age=25;
```

NB : **même si ces propriétés n'existent pas dans la fonction « constructeur »**

Objets : destructeurs

Il n'y a **pas de fonction destructeur** explicite en JavaScript

L'opérateur **delete** permet de supprimer des valeurs (sauf celles déclarées avec var)

null peut être utilisé pour supprimer une référence

```
monobj = new qqChose();
delete monObj; // supprime monObj
mon obj=null; // "vide" monObj
```

Objets : propriétés de classe (statiques)

Il est possible d'associer une **propriété à une classe** (ou **propriété statique**).

- Elle n'est pas copiée pour chaque instance.
- Les propriétés de classe représentent souvent des constantes communes à toutes les instances.

```
Etudiant.statut = "étudiant";
```

Par exemple : Pi en mathématique (objet Math).

Objets : méthodes d'objet

= une méthode associée à un objet.

technique 1 : déclaration dans la fonction "constructeur"

```
function Etudiant (unNumero, unNom, unPrenom{
  this.numero = unNumero;
  this.nom = unNom;
  this.prenom = unPrenom;
  this.indiquerNumero = function() {
    alert(this.prenom + " " + this. nom + " : " +
        this.numero);
  };
}
```

Objets : méthodes d'objet

technique 2 : accès à une fonction externe (permet la factorisation)

```
function alerteNumero() {
   alert (this.prenom + " " + this.nom + " : " + this.numero);
}
function Etudiant (unNumero, unNom, unPrenom{
   this.numero = unNumero;
   this.nom = unNom;
   this.prenom = unPrenom;
   this.indiquerNumero = alerteNumero;
}
```

Objets : méthodes d'objet

Dans les 2 cas, on pourra invoquer la méthode pour une instance donnée

```
// Corps du programme
unEtudiant = new Etudiant(45685, "Durant", "Paul");
unEtudiant.indiquerNumero();
```

Objets : méthodes d'objet

Ces manières de faire présentent un inconvénient : ainsi déclarées, **les méthodes occuperont de l'espace mémoire pour chaque instance créée, que les méthodes soient utilisées ou non**.

Objet prototype

Objet prototype = un objet (propriété de l'objet Object).

Object : +/- superclasse dont tous les objets héritent.

Function est aussi un objet JavaScript, il a donc accès à la propriété prototype.

Les instances d'objets créées par une fonction constructeur héritent de l'ensemble des propriétés du prototype.

Objet prototype

Objet prototype

```
function Etudiant (unNumero, unNom, unPrenom){
   this.numero = unNumero;
   this.nom = unNom;
   this.prenom = unPrenom;
}
Etudiant.prototype.indiquerNumero() {
   alert (this.prenom + " " + this.nom + " : " + this.numero);
}
```

Les différentes instances des objets Etudiants hériteront des propriétés du prototype.

La méthode ne sera pas copiée dans chaque instance, mais héritée du prototype.

Cela implique une moindre consommation de mémoire.

Héritage

Une définition d'objet peut descendre d'une autre définition d'objet. Pour cela, on utilise la propriété prototype :

```
function MonObjet (arg1) {
  this.prop1 = arg1;
}
function MonObjetDerive (arg2) {
  this.prop2 = arg2;
}
MonObjetDerive.prototype = new MonObjet();
```

Les instances de MonObjetDerive incluront les propriétés et les méthodes de MonObjet et de MonObjetDerive.

Objets prédéfinis

Objet	Méthodes / propriétés
Date (dates et heures)	getMonth, getDay, getTime...
Document (document courant)	close, open, write...
Form (formulaire)	target, length, reset, submit, ...
History (historique du navigateur)	length, back, forward...
Image (images)	border, width, height...
Location (URL courant)	hostname, port, protocol...
Math (propriétés et méthodes mathématiques)	PI, SQRT2, abs, atan, cos, random, round…
Navigator(navigateur courant)	appName, appVersion, language…
Window (fenêtre courante)	document, history, name, status, alert, confirm, prompt...

Tableaux

- Objets prédéfinis en JavaScript.
- **new** pour l'objet constructeur Array()

```
monTableau = new Array (nbElements);
```

- NB : le premier élément d'un tableau a le rang 0

- Pour accéder à la valeur d'un élément du tableau (ou pour affecter une valeur), on utilise les crochets [ ] avec l'indice de l'élément.

Tableaux

```
unTableau = new Array ();
unTableau[0] = "printemps";
unTableau[1] = "été";
unTableau[2] = "automne";
unTableau[3] = "hiver";
```

Tableaux

Il n'y a pas de restriction quand au type des données d'un tableau (plusieurs types peuvent cohabiter dans le même tableau).

```
unTableau = new Array ();
unTableau[0] = "printemps";
unTableau[1] = 2;
unTableau[2] = 5.5;
unTableau[3] = true;
unTableau[5] = array("chaud", "froid");
```

Tableaux

- Il n'est pas obligatoire de déclarer au préalable la taille du tableau.
- La taille d'un tableau peut changer dynamiquement.
- Si l'on crée un élément à un rang qui n'existe pas, il est créé, ainsi que les éléments aux rangs qui le précède (ils ont une valeur null).

```
unTableau = new Array ();
unTableau[6] = "printemps";
==> unTableau [0..5] valent null
```

Tableaux

- On peut créer un tableau en assignant directement des valeurs.

```
unTableau = new Array ("printemps", "été");
unTableau = ["printemps", "été"];
```

- Les tableaux associatifs acceptent des indices textuels :

```
unTableau = new Array (2);
unTableau["saison"] = "printemps";
unTableau["meteo"] = "pluvieux";
```

Structures de contrôle et de boucle

if

```
if (expression conditionnelle) {
      commandes
}
```

if ... else

```
if (expression conditionnelle) {
      commandes;
} else {
      commandes;
}
```

if ... else if

```
if (expression conditionnelle) {
      commandes;
} else if (expression conditionnelle) {
      commandes;
}
```

Structures de contrôle et de boucle

switch

```
switch (expression) {
  case etiquette1 :
    commandes;
    break;
  case etiquette2 :
    commandes;
    break;
  default :
    commandes;
    break;
}
```

Structures de contrôle et de boucle

while

```
while (expression conditionnelle) {
        commandes;
}
```

do ... while

```
do {
   commandes;
} while (expression conditionnelle)
```

Structures de contrôle et de boucle

for

```
for (expression d'initialisation; expression conditionnelle; mise à jour) {
   commandes;
}
```

Ex :

```
for (var i=0; i < 10; i++) {
   commandes;
}
```

Structures de contrôle et de boucle

for ... in

Exécute la même commande ou le même groupe de commande pour toutes les propriétés d'un objet.

```
for (element in groupe) {
 commandes;
}
```

Exemple 1 :

```
tableau = array("printemps", "été", "hiver");
for (num in tableau) {
    console.log(tableau[num];
}
```

Structures de contrôle et de boucle

Exemple 2 :

```
function Etudiant (unNumero, unNom, unPrenom) {
  this.numero = unNumero;
  this.nom = unNom;
  this.prenom = unPrenom;
}

unEtudiant = new Etudiant (45685, "Testeur", "Toto");

for (propriete in unEtudiant) {
  document.write(propriete + "<br />");
}
```

Structures de contrôle et de boucle

With

Permet de créer un regroupement de commandes pour lesquelles l'objet est défini puis implicite (inutile de saisir son nom pour accéder aux propriétés).

```
with (objet) {
    commandes;
}
```

Exemple :

```
with (unEtudiant) {
  numero = 12;
  nom = "Testeur";
  prenom = "Toto";
}
```

Structures de contrôle et de boucle

break

Permet de quitter une boucle (sous réserve qu'une condition soit remplie par exemple).

```
while (expression conditionnelle) {
  commandes;
  if (condition) {
    break;
  }
}
```

Structures de contrôle et de boucle

continue

Permet de passer directement à l'itération suivante de la boucle.

```
while (expression conditionnelle) {
   commandes;
   if (condition) {
      continue;
      // si la condition vaut true, les commandes ci
      // dessous ne seront pas exécutées pour cette boucle
   }
   commandes;
}
```

Evènements

JavaScript permet d'agir dans un document HTML en réaction à des événements

- déclenchés "spontanément" **par le navigateur** (chargement du document, fermeture du document…)
- déclenchés **par les actions l'utilisateur** (click, déplacements du curseur de la souris…)

Exemples d'évènements

- **blur** : un élément perd le focus
- **change** : un élément a été modifié
- **click** : l'utilisateur clique sur un élément
- **dblclick :** l'utilisateur double-clique sur un élément
- **load** : le navigateur a chargé la page HTML
- **mouseover** : l'utilisateur place le pointeur de la souris sur un élément
- **reset** : un formulaire est réinitialisé
- **resize** : l'utilisateur redimensionne la fenêtre du navigateur
- **scroll** : l'utilisateur se sert des barres de défilement de la page
- **select** : l'utilisateur sélectionne du texte
- **submit** : un formulaire est validé
- **unload** : le navigateur ferme le document

Gestionnaire d'évènements

- Les gestionnaires d'évènements permettent d'exécuter du code JavaScript en réponse aux événements.
- En JavaScript, le gestionnaire d'événement s'écrit avec le nom de l'événement préfixé de " on "
- Illustration :

  click = événement

  onclick = gestionnaire d'événement
- Exemple :

```
<a href="pagesuivante.html" onclick="return
window.confirm('On change de page');">
```

Gestionnaire d'évènements

- Dans un gestionnaire d'événement, il est possible d'appeler une fonction
- Lors de l'appel de la fonction, on peut si besoin passer l'évènement en paramètre (par ex. pour obtenir la position de la souris, savoir si c'était un clic gauche ou droit etc.)

  Exemple :

```
<a href="page.html" onclick="gererEvt(event)">
```

- On peut passer l'objet cliqué en paramètre, par ex. pour obtenir son id (en plus de l'évènement ou pas)

  Exemple :

```
<a href="page.html" onclick="gererEvt(event, this)">
```

Gestionnaire d'évènements

```
function gererEvt(evenement, elementClic) {
  if (evenement.ctrlkey) { // touche Ctrl enfoncée
      …
      elementClic.innerHTML = "" ;
      …
  }
}   Z
```

Les commentaires

Les commentaires sont :

–encadrés par des /* et des */ lorsqu'ils tiennent sur plusieurs lignes ;

–précédés de // lorsqu'ils apparaissent sur une seule ligne.

–Exemple

```
/*
Mes commentaires tiennent sur plusieurs
lignes ...
*/

Code JavaScript
// Un commentaire sur une ligne
Code JavaScript
```

Intégration dans le document

Entouré de tags :

```
<script type="text/javascript">
// Code JavaScript …
</script>
```

- Dans l'en-tête ou le corps du document : il est théoriquement chargé en mémoire du navigateur avant l'affichage du corps du document => déclarer dans l'en-tête les fonctions/variables invoquées ultérieurement dans le document.
- Attention, les navigateurs utilisent le JavaScript dans l'ordre d'apparition des instructions.
- Pour éviter tout problème d'exécution intempestif, appeler une fonction explicitement après chargement du document : **<body onload="commencer();">**

Intégration dans le document

Code JavaScript placé dans des fichiers externes.

⇨ extension **.js** (fichiers au format texte contenant les lignes de code, **sans** balise <SCRIPT>…)

Exemple d'intégration :

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

L'intégration d'un fichier externe se fait généralement dans l'en-tête d'un document.

Accéder au DOM

Accéder à un élément avec getElementById

Accéder à un élément HTML qui possède l'attribut id correspondant à celui reçu en paramètre.

```
var elt=document.getElementById("uid");
```

Accéder au DOM

Accéder à un tableau d'éléments avec getElementByTagName

Accéder à tous les éléments dont le tag (le nom du tag, ou de la balise) correspond à celui reçu en paramètre.

```
var tableauDesParagraphes =
document.getElementsByTagName("p");
```

Accéder au DOM

Exemple : je souhaite retrouver tous les liens d'un document et créer un gestionnaire d'événement pour chacun d'entre eux.

```
function installerGestionnaireDeLiens() {
   // je cherche toutes les balises a (<a href=…) dans le document
   var liens = document.getElementsByTagName("a");

   for (var i=0; i < liens.length; ++i) {
       liens[i].onclick = function() {
              alert("ceci est un lien") ;
              return false ;
          }
   }
}
```

Accéder au DOM

- Cette fonction (fantaisiste) pourra être appelée au chargement du document (lorsque le document est chargé) avec

```
<body onload="installerGestionnaireDeLiens();">
```

- ou, si l'on souhaite que le code JavaScript ne soit pas "intrusif" (cette notion indique que l'on sépare le code JavaScript du code HTML), le gestionnaire d'événement peut être placé dynamiquement :

```
window.onload=function(){
   installerGestionnaireDeLiens();
}
```

Accéder au DOM

Accéder à un tableau d'éléments avec getElementsByClassName

Accéder à tous les éléments dont l'attribut class correspond à celui reçu en paramètre.

```
var eltsArray=
  document.getElementsByClassName("maClasse");
```

DOM : accéder aux attributs

Gérer les attributs d'une balise avec getAttribute, setAttribute, removeAttribute

getAttribute :

récupérer la valeur de l'attribut d'un élément.

s'applique à un élément, le nom de l'attribut est passé en paramètre.

```
var attValeur = elt.getAttribute("value");
```

DOM : accéder aux attributs

Voici la fonction du chapitre précédent revue pour que la boite de dialogue affiche la valeur de l'attribut href des liens

```
function installerGestionnaireDeLiens() {
   var liens=document.getElementsByTagName("a");
   for (var i=0; i<liens.length; ++i) {
     liens[i].onclick=function() {
          alert(liens[i].getAttribute("href")) ;
          return false ;
      }
   }
}
```

DOM : accéder aux attributs

Si l'on souhaite que la fonction ne s'applique qu'aux liens d'une classe (CSS) particulière, on peut utiliser la même méthode, en ajoutant par exemple :

```
//...
if (liens[i].getAttribute("class")=="maclasse") {
 //...
}
//...
```

DOM : accéder aux attributs

La méthode setAttribute permet de fixer la valeur de l'attribut d'un élément. Elle s'applique donc à un élément, le nom de l'attribut est passé en paramètre, ainsi que la valeur souhaitée pour cet attribut.

```
elt.setAttribute("att","val");
```

par exemple

```
//...
liens[i].setAttribute("title","un titre") ;
//...
```

DOM : suppression d'attribut

La méthode removeAttribute permet de supprimer un attribut d'un élément. Elle s'applique donc à un élément, le nom de l'attribut est passé en paramètre.

```
elt.removeAttribute("att");
```

- NB : selon les navigateurs, tous les attributs ne peuvent pas être tous supprimés (id ou class, par exemple, avec IE – A vérifier toutefois).

DOM : trouver le style d'un élément

Il est possible de trouver une directive de style appliquée à un nœud dans le DOM.

ATTENTION : la syntaxe des propriétés changent par rapport à celle des déclarations CSS. Les propriétés comportant un trait d'union sont réécrites en un seul « bloc » en supprimant le trait d'union et en mettant en majuscule la première lettre du mot suivant (par exemple : border-bottom devient borderBottom).

```
var element = document.getElementById("…");
var computedStyle = element.currentStyle ||
window.getComputedStyle(element, null);
console.log computedStyle.backgroundColor;
```

DOM : modifier le style

Il est possible d'intervenir sur une directive de style appliquée à un nœud dans le DOM.

Seule la syntaxe des propriétés changent par rapport à celle des déclarations CSS : les propriétés comportant un trait d'union sont réécrites en un seul « bloc » en supprimant le trait d'union et en mettant en majuscule la première lettre du mot suivant (par exemple : border-bottom devient borderBottom).

```
var elt = document.getElementById("monID");
elt.style.borderBottom = "solid 1px #000000";
```

DOM : créer des éléments

Plusieurs méthodes permettent de créer des éléments de type nœud ou de type texte dans le DOM, certaines méthodes sont complémentaires.

En résumé :

– **createElement** : permet de créer un nœud de type élément
– **createTextNode** : permet de créer un nœud de type #text

DOM : créer des éléments

Après la création de l'élément, il faut le "placer" dans le DOM

- **appendChild** : permet d'ajouter un noeud enfant à un noeud existant
- **insertBefore** : permet d'ajouter un nœud enfant à un nœud existant, le nœud étant inséré avant un autre nœud enfant. Par exemple : parent.insertBefore (nouveauNoeud, noeudExistant) ;

DOM : créer des éléments

– **cloneNode** : cloneNode permet la copie soit de la structure d'un élément, soit de la structure et du contenu d'un élément vers un autre élément.
Par exemple :

```
var noeud = autreNoeud.cloneNode(false); //
false indique que seule la structure est
copiée
var noeud = autreNoeud.cloneNode(true); //
true indique que la structure et le contenu
sont copiés
```

DOM : créer des éléments

```
var elt=document.getElementById("unId");
// je crée la liste (ul)
var newUl = document.createElement("ul");
// je l'ajoute comme enfant du nœud principal
elt.appendChild(newUl);
// je crée un élément de liste (li)
var newLi = document.createElement("li");
// je l'ajoute comme enfant à l'élément ul
newUl.appendChild(newLi);
// Je crée un nœud de type text
li_text = document.createTextNode("Hello world");
// je l'ajoute comme enfant de l'élément de liste
newLi.appendChild(a_text);
```

Exemple : créer une liste (ul) d'éléments (li) dans un nœud donné dont l'id est "unId"

Remarque : utilisation de la méthode createElement pour déclarer les nouvelles balises HTML5 pour les navigateurs ne les supportant pas
(IE < 9 notamment)

```
<!--[if lt IE 9]>
      <script>
          document.createElement("header");
          document.createElement("footer");
          ….
       </script>
<![endif]-->
```

On crée l'élément sans faire d'appendChild,

→ De cette manière ces éléments sont alors connus dans le DOM

DOM : supprimer des éléments

removeChild : on peut enlever un nœud, enfant d'un autre nœud

Par exemple :

```
eltParent. removeChild(eltEnfant);

Ou en utilisant firstChild (voir plus loin)

eltParent.removeChild(eltParent.firstChild);
```

DOM : méthodes et propriétés relationnelles

- **ParentNode** : retourne le nœud parent d'un nœud.

```
eltParent = elt.parentNode;
```

- **ChildNodes** : tableau de tous les nœuds enfants d'un nœuds donné (ou undefined s'il n'y a pas d'enfants)

```
var tousLesEnfants = elt.childNodes;
var nbEnfants = elt.childNodes.length // nombre
d'éléments enfants
```

- **hasChildNode**() : indique si un nœud a des enfants (retourne true) ou non (retourne false)

```
elt.hasChildNode();
```

DOM : méthodes et propriétés relationnelles

- **firstChild** : retourne le premier enfant d'un noeud
```
var premierEnfant = elt.firstChild;
```

- **lastChild** : retourne le dernier enfant d'un nœud
```
var dernierEnfant = elt.lastChild;
```

- **nextSibling** : renvoie le frère d'un élément (nœud adjacent suivant)
```
var frereSuivant=elt.nextSibling ;
```

- **previousSibling** : renvoie le frère d'un élément (nœud adjacent précédent)
```
var frerePrecedent=elt.nextSibling ;
```

DOM : méthodes et propriétés

Propriétés d'un nœud

- **nodeName** renvoie le nom du nœud courant.
  Par exemple, si elt désigne un élément img, elt.nodeName renvoie la chaîne de caractères "img".

- **nodeValue** contient la valeur du nœud. Cette valeur dépend du type de nœud. Pour les nœuds de type texte, il s'agit du contenu.

```
var valeur = elt.nodeValue;
```

- **nodeType** contient le type du nœud, selon un codage numérique (1 pour un nœud de type élément, 3 pour un nœud de type texte…)

```
var type = elt.nodeType;
```

DOM : méthodes et propriétés

•innerHTML vs Méthodes DOM

innerHTML est une propriété facile à utiliser pour lire ou écrire le code HTML d'un élément, propriété généralement reconnue par les navigateurs (attention, cela ne concerne pas les documents qui sont d'un content type application/xhtml+xml)

Exemple :

```
<div id="test">
<p>Voici <em>un peu</em> de contenu</p>
</div>
```

Si je fais :

```
var monTest=document.getElementById("test");
alert(monTest.innerHTML);
```

La boite de dialogue m'affichera : Voici un peu de contenu

DOM : méthodes et propriétés

innerHTML

– C’est une manière rapide est facile de modifier le contenu d’un élément mais elle n’a pas la « granularité » des méthodes DOM, et le contenu inséré n’est pas manipulable dans toute sa hiérarchie d’éléments (on insère du texte et pas des éléments du DOM)

– C’est donc une méthode à réserver pour « aller vite », dans des cas ne nécessitant pas de manipulation ultérieure des éléments insérés.

Canvas 1/2

```
<canvas id="monCanvas" width="200" height="400"></canvas>
```

- **Définit un "espace de dessin"**
- Associé à un **"contexte"** permettant de tracer des éléments
- Le dessin (formes, traits, courbes, texte) se fait en Javascript (API)
- Les attributs sont gérés en Javascript (remplissage, ombres, …)
- L'animation se fait en redessinant l'image (ou une partie)

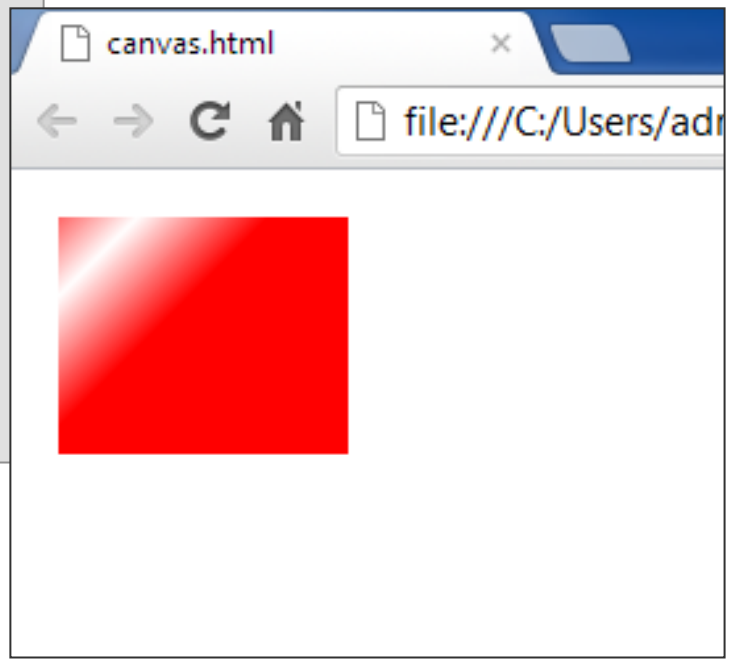
- **Avantages** : rapidité, portabilité (concurrent de Flash)
- **Inconvénients** : sort du DOM (accessibilité ? Cohérence HTML ?)

Canvas 2/2

```
<canvas id="myCanvas" width="200" height="100"></canvas>
…
var canvas  = document.getElementById("myCanvas");
var context = canvas.getContext("2d");

var linearGradient1 = context.createLinearGradient(0,0,50,50);
linearGradient1.addColorStop(0  , 'rgb(255, 0, 0)');
linearGradient1.addColorStop(0.5, 'rgb(255, 255, 255)');
linearGradient1.addColorStop(1  , 'rgb(255, 0, 0)');

context.fillStyle = linearGradient1;
context.fillRect(10,10,110, 110);
```



Utilisation de Javascript en HTML
Canvas : exemple (1/2)

http://www.williammalone.com/articles/html5-canvas-example/

```
var width = 125; // Triangle Width
var height = 105; // Triangle Height
var padding = 20;

// Draw a path
context.beginPath();
context.moveTo(padding + width/2, padding); // Top Corner
context.lineTo(padding + width, height + padding); // Bottom Right
context.lineTo(padding, height + padding);  // Bottom Left
context.closePath();

// Fill the path
context.fillStyle = "#ffc821";
context.fill();
```

...

...

```
context.textAlign = "center";
context.textBaseline = "middle";
context.font = "bold 60px 'Times New Roman', Times, serif";
context.fillStyle = gradient;
try{
    context.fillText("!", canvasWidth/2, padding + height/1.5);
}catch(ex){}
```

Utilisation de Javascript en HTML
Canvas : exemple (2/2)

http://www.effectgames.com/demos/canvascycle/

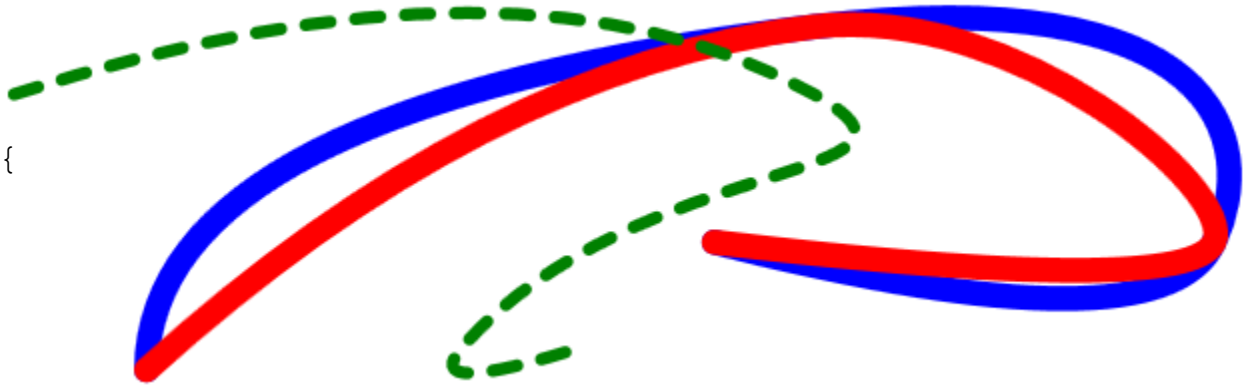
Utilisation de Javascript en HTML
Canvas : KineticJS

```
<script src="http://d3lp1msu2r81bx.cloudfront.net/kjs/js/lib/kinetic-v4.7.2.min.js"></
script>
      ...
    <div id="container"></div>
      ...
    <script>
      var stage = new Kinetic.Stage({
        container: 'container',
        width: 578,
        height: 200
      });

      var layer = new Kinetic.Layer();

      var blueSpline = new Kinetic.Spline({
        points: [{ x: 73, y: 160}, {x: 340, y: 23}, {x: 500, y: 109}, {x: 300,y: 109}],
        stroke: 'blue',
        strokeWidth: 10,
        lineCap: 'round',
        tension: 1
      });

      layer.add(blueSpline);
      stage.add(layer);
          layer.draw();
    </script>
```



Principal avantage : les éléments du canvas deviennent manipulables comme des éléments du DOM en javascript.

Les nouveautés dans le code Javascript
L'API de géolocalisation

- **Geolocation API**
  - Fonctionne avec Javascript et l'objet `navigator.geolocation`
  - Récupère les coordonnées du poste de l'utilisateur en longitude, latitude et altitude
    - `getCurrentPosition`
    - `watchPosition (suivi en continu)`

Les nouveautés dans le code Javascript
L'API de géolocalisation

- Exemple: fonction de callback nommée (maPosition) et passée en argument à getCurrentPosition() => fournit une série de propriétés (dans l'exemple la position, mais aussi la vitesse, la précision, etc.)

```
interface Coordinates {
    readonly attribute double latitude;
    readonly attribute double longitude;
    readonly attribute double? altitude;
    readonly attribute double accuracy;
    readonly attribute double? altitudeAccuracy;
    readonly attribute double? heading;
    readonly attribute double? speed;
};
```

```
interface Position {
    readonly attribute Coordinates coords;
    readonly attribute DOMTimeStamp timestamp;
};
```

```
function maPosition(position) {
  var infopos = "Position déterminée :\n";
  infopos += "Latitude : "+position.coords.latitude +"\n";
  infopos += "Longitude: "+position.coords.longitude+"\n";
  infopos += "Altitude : "+position.coords.altitude +"\n";
  document.getElementById("infoposition").innerHTML = infopos;
}

navigator.geolocation.getCurrentPosition(maPosition);
```

http://www.alsacreations.com/tuto/lire/926-geolocalisation-geolocation-html5.html

Les nouveautés dans le code Javascript

Le Drag and Drop

```
<li draggable="true">Element de ma liste</li>
<div ondrop="manageDrop(event);"/>
```

- Le drag (le drop) génèrent des évènement :
  - `dragStart` *(on commence à déplacer un élément)*
  - `dragEnter` *(on commence le survol d'une cible potentielle)*
  - `dragOver` *(on survole d'une cible potentielle)*
  - `dragLeave` *(on sort d'une cible potentielle)*
  - `drop` *(on drop dans cible potentielle)*
  - ...

Les nouveautés dans le code Javascript

Le Drag and Drop

- On peut dragger un élément ou autre chose (son contenu textuel par exemple)
  - Il faut donc mémoriser ce qui doit être déplacé
  - `event.dataTransfer.setData("idDrag", ev.target.getAttribute('id'));`
- On peut déplacer, copier, créer un lien vers l'élément,...
  - Il faut définir l'effet du drag&drop
  - `event.dataTransfer.effectAllowed = 'move';`
- On peut définir l'image qui représente l'élément déplacé
  - `event.dataTransfer.setDragImage(ev.target, 0, 0);`

Les nouveautés dans le code Javascript
L'API Fichier

- Sélectionner des fichiers, accéder à leurs données...
  - Améliorer l'upload
    - Recueil d'informations au lancement de l'upload
    - Barre de progression
  - **Accéder au contenu de fichiers choisis par l'utilisateur**
  - Déplacer, copier, supprimer, écrire...

https://developer.mozilla.org/en-US/docs/Using_files_from_web_applications

L'API Fichier : exemple

```
<input id="file" type="file" onchange="lireFichier()"/>
[…]
var files = $('#file').prop('files');
if (files.length == 0) {
    alert('Vous devez choisir un fichier') ;
    return false ;
}
var reader = new FileReader();
reader.onload = function(progressEvent){
    var lines = this.result.split('\n');
    for (var iLine = 0; iLine < lines.length; iLine++){
        $('#lines').append(lines[iLine]) ;
    }
}
for (var iFile = 0, f; f = files[iFile]; iFile++) {
    reader.readAsText(f);
}
```

Les nouveautés dans le code Javascript
Stockage local

- **Web Storage**
  - API `sessionStorage`
    - Stockage de données de session
      - Durée de vie courte
  - API `localStorage`
    - Stockage de données sur la machine cliente
      - Restent stockées même près la fermeture du navigateur
  - = Alternatives aux sessions coté serveur et aux cookies
  - = Scripts exécutés sur le client uniquement (sans requêtes au serveur)

```
localStorage.setItem("name", "John");
alert(localStorage.getItem("name"));
```

http://geotribu.net/node/218

Les nouveautés dans le code Javascript
API Push

- **Push API**
  - Événements envoyés par le serveur
    - A l'initiative du serveur
      SSE : Server-Sent Events
  - C'est le client qui initie la connexion et va ensuite rester à l'écoute, la connexion est alors permanente
  - Le serveur peut alors envoyer des informations au format texte / json

Server Sent events : exemple

```
if (!!window.EventSource)  { // Serveur Sent Events dispo
      source = new EventSource('ajax/getServerInformation.php?userId=' + userId);


       source.addEventListener('eventName1', function(e) {
             console.log("eventName1 occured : " + e.data) ;
             var data = JSON.parse(e.data);
             …
       }
       source.addEventListener('eventName2', function(e) {
             console.log("eventName2 occured : " + e.data) ;
             …
       }

}
```

```
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');

function sendMsg($eventName, $data) {
      echo "event: $eventName\n" ;
      echo "data: $data\n\n" ;
      ob_flush();
      flush();
}

while (true) {
      …
      sensMsg(  "eventName1",
                json_encode($tableau)) ;
      …
}
```

Les nouveautés dans le code Javascript
Web Sockets

- **Web Sockets**
  - Communications par sockets avec un serveur
    - Connexion **bidirectionnelle permanente**
    - Informations échangées en temps réel (≠ push)
  - Applications :
    - Ce qui nécessite du « temps réel » : chat, jeux, travail collaboratif…
  - Contraintes :
    - Nécessite un gestionnaire côté serveur : Kaazing ; Jetty Netty, ou JWebSocket (Java) ; Websocket (Python) ; …

http://blog.zenika.com/index.php?post/2011/02/25/Html5-et-les-webSockets

Les nouveautés dans le code Javascript
Web Workers

- **Web Workers**
  - Javascript en multithread
  - Javascript en tâche de fond
    - Non bloquant pour le navigateur
  - Chaque Worker s'exécute dans un thread séparé
  - Pourront être mis à profit pour les traitements lourds

http://blogs.msdn.com/b/davrous/archive/2011/07/08/introduction-aux-web-workers-d-html5-le-multithreading-version-javascript.aspx