

Objectifs

Ce cours a pour objectif de :

- comprendre les mécanismes de la programmation événementielle
- comprendre les concepts de base du javascript afin de
 - créer des interactions simples
 - réutiliser de **manière pertinente** du code déjà écrit
 - être capable de découvrir par soi-même de nouveaux concepts

Présentation

Javascript est un langage de programmation, créé par les sociétés Netscape et Sun Microsystems vers la fin de l'année 1995 (sous le nom de livescript).

Son objectif principal : introduire de l'interactivité avec les pages HTML, et effectuer des traitements simples **sur le poste de travail de l'utilisateur**. Jusqu'alors la programmation ne pouvait être exécutée que sur le serveur.

Le moyen utilisé : introduire de petits programmes, appelés **SCRIPTS**, dans les pages HTML.

Présentation

On peut aussi être amené à mettre en place des animations sur notre page HTML :

- Déplacement d'éléments
 - Changement de dimension d'un élément
 - Modification de l'opacité
-
- Ces capacités d'animations existent déjà en CSS 3 (transitions, animation)

Le javascript va augmenter cette capacité d'interaction et d'animation

Présentation

Des exemples d'utilisation :

[Zoombox](#)

[Des dessins et des animations](#)

[Galerie 1](#)

[Galerie2](#)

[Appareilphoto](#)

[Autocompletion](#)

[Jeu 2d](#)

[Vérification de formulaire](#)

La notion de programmation événementielle

Le principe de la programmation événementielle est très différente de la programmation séquentielle :

- Il n'y a pas, comme dans la programmation séquentielle , de fonction main. Nos scripts sont exécutés par le programme principal, dans notre cas le navigateur.
- On ne sait pas systématiquement, au moment où l'on programme, si le code sera exécuté ou non. Un même code peut être exécuté plusieurs fois. Le code que l'on met en place est en effet associé à la survenue d'un événement
- Le code va le plus souvent être découpé en fonctions indépendantes les unes des autres

La notion de programmation objet

Les principes de programmation objet vont s'appliquer à ce cours javascript :

Rappel :

Avant d'afficher la page, le navigateur va d'abord analyser le code HTML et créer en mémoire un **objet** pour chaque élément trouver dans ce code (balise, texte, attribut, commentaire,...). Ces objets sont liés les uns aux autres par une arborescence :

Cette arborescence s'appelle **l'ARBRE DOM**

Chaque élément de cet arbre est appelé nœud de l'arbre.

C'est à partir de cet arbre DOM que la page est ensuite affichée

Toute sélection d'élément HTML afin de le mettre en forme ou de le rendre interactifs se fera dans cet arbre DOM.

La notion de programmation objet

Donc tout élément visible (ou invisible) sur votre page correspond à un objet. Comme tout objet, chaque objet a des caractéristiques(attributs) et un comportement défini(méthode) .

Par contre, un certain nombre d'étapes nécessaires à la programmation objet sont **prises en charge par le navigateur** :

- **L'écriture des classes** correspondant à chaque objet ("modèle") .
Le modèle que doit respecter chaque objet est défini par la documentation du W3C
- **L'instanciation des objets** : les objets seront en grande majorité instanciés par le navigateur lors de l'analyse du flux HTML(il sera malgré tout possible d'instancier de nouveaux objets par code)

La notion de programmation objet

Question1 : Combien d'objets de type nœud élément ou nœuds textes sont créés par le DOM suite à l'analyse du flux HTML suivant:

```
<div>
```

```
  <div>
```

```
    <p> Quel beau site </p>
```

```
  </div>
```

```
</div>
```


La notion de programmation objet

Question 2 : Citer des attributs d'un élément img

Question 3 : Citer des comportements possibles d'un élément img

Comment programmer en javascript

Tout le code javascript que vous écrirez sera stocké dans un (ou plusieurs fichiers) , ces fichiers ayant **.js** comme extension.

ex: td1Exercice1.js

Bien évidemment, comme pour les fichiers css , ce fichier devra être associé à la page html dans la partie <head> ou <body>.

`<script type="text/javascript" src="chemin relatif du fichier.js"></script>`

Ex : `<script type="text/javascript" src="td1Exercice1.js"></script>`



Souvent, votre code ne fonctionne pas car vous ne l'avez pas ou vous l'avez mal associé à votre page. Les outils de développement vous permettront très rapidement de vérifier que votre fichier javascript a bien été chargé par le navigateur

QUI - QUAND - COMMENT

La 1ere étape consiste à lister les différentes interactions que vous souhaitez mettre en place sur votre site.

Pour chaque interaction souhaitée, vous allez devoir répondre à 3 questions **AVANT** de pouvoir la mettre en place :

QUI : Quelle est la source de l'interaction.

En d'autres termes, l'interaction a lieu sur quel(s) **éléments de ma page**

Ex : sur toutes les images, sur un élément précis , sur tous les enfants de tel élément ,

QUAND : Quel est l'événement déclencheur

Est-ce une **action utilisateur** : cliquer, passer la souris, redimensionner, taper sur le clavier

Ou bien **une action "système"** : chargement de la page, chargement de photo,

Ou bien encore **une action régulière** (toutes les x secondes) ou différée dans le temps....

QUI - QUAND - COMMENT

COMMENT : Quel(s) résultat(s) souhaite-t-on obtenir

- Modifier le contenu de la page :
 - ajouter un(des) élément(s) HTML
 - supprimer un(des) élément(s) HTML
 - modifier la valeur d'un ou plusieurs attributs d'élément(s) HTML)
- Modifier la mise en forme de la page
 - modifier la valeur d'une ou plusieurs propriétés css d'élément(s) :
 - rendre visible ou invisible,
 - déplacer,
 - jouer sur la taille,
 - jouer sur l'opacité,.....
- Vérifier et informer de la validité des informations saisies
- Envoyer des demandes sur le serveur pour récupérer des informations complémentaires ou mettre à jour des informations.
- Sauvegarder des informations,

QUI - QUAND - COMMENT

Exemples :

quelle action - sur quel objet - résultat attendu

- Un click de l'internaute sur le bouton "voir conditions" déclenche l'affichage des conditions générales de vente
- La soumission du formulaire d'inscription déclenche la vérification que les zones nom , prénom, email sont correctement remplies
- Lorsque le navigateur a fini le chargement de la page il faut cacher les sous-menus de la barre de menu
- Le click sur une des images de la div galerie déclenche l'affichage celle-ci dans sa version "grande"
- La frappe d'une touche non numérique dans la zone numéro stoppe l'affichage de celle-ci
- Lorsqu'il valide une question du blind test on lui affiche la suivante
- Lorsqu'il valide la dernière question du blindtest on lui affiche son score

QUI

La notion objet

Javascript est un langage orienté objet:

Les objets gérés par le navigateur

- on dispose d'objets liés à la page chargée : fenêtre, page, tous les éléments du DOM

- On dispose d'objets standards décrits par le langage : String, Date, RegExp,.....

On peut aussi créer par programmation **ses propres objets**

Le DOM

Nous avons vu dans le cours HTML, que le navigateur, lors du chargement de la page, **crée en mémoire des objets correspondants à chaque élément** de la page. Ces objets ont une structure hiérarchique les uns par rapport aux autres (notion "d'arbre").

Le **DOM** (Document Object Model) est une API (définie par le W3C) permettant d'obtenir que toutes les applications et notamment les navigateurs accèdent de manière identique à ces objets. Il définit donc les propriétés et les méthodes des différents objets manipulés.

L'objet window

Au chargement de la page, le premier objet créé par le navigateur est l'objet **window**

Cet objet dispose d'un certain nombre de propriétés identifiant l'environnement de la fenêtre. Voici les principaux :

- `window.location` : barre d'adresse
- `window.status` : la barre d'état
- `window.navigator` : le navigateur utilisé
- `window.screen` : l'écran dont dispose votre internaute
- `window.document` : la page chargée dans la fenêtre

• L'objet window est toujours sous-entendu, on peut omettre son nom :

`window.screen.width`  `screen.width`

L'objet window

Les principales méthodes dont il dispose, permettent :

- de mettre en place des **timers** qui permettront d'exécuter des actions répétitives toutes les x milli-secondes
 - window.setTimeout (...)
 - window.setInterval(....)
- d'**imprimer** la page
 - window.print(...)
- d'ouvrir des **pops-ups**
 - window.open(...)
- d'afficher des **boîtes de dialogues**
 - window.alert (...)
 - window.confirm(....)
 - window.prompt(....)

L'objet document

- Le deuxième objet instancié par le navigateur est l'objet **document**, cet objet est instancié systématiquement, par le navigateur, dès le démarrage du chargement de la page.
- On y accède via la syntaxe : **window.document** ou bien encore **document** (window étant alors sous entendu)
- Il correspond à la page chargée dans la fenêtre.
- Il dispose d'un certain nombre de propriétés et de méthodes permettant **d'accéder aux différents éléments de la page**
- Il permet aussi de **créer de manière dynamique** (par code) **de nouveaux éléments dans la page**

Les objets noeuds éléments et noeuds textes

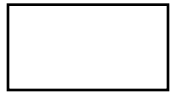
Le navigateur va ensuite analyser le code HTML qu'il a reçu du serveur web et va instancier (créer)

- un **noeud élément** à chaque fois qu'il rencontre une balise
- un **noeud texte** à chaque fois qu'il rencontre un texte

Les noeuds éléments ont pour propriété tous les attributs html qui ont été définis par le W3C et **sont capables d'accéder à tous les éléments ayant un lien avec eux** (parent, frère, enfants, descendants). Ils sont aussi aptes à **réagir à des actions utilisateurs** (click, frappe, passage de souris, etc,...)

L'arbre DOM

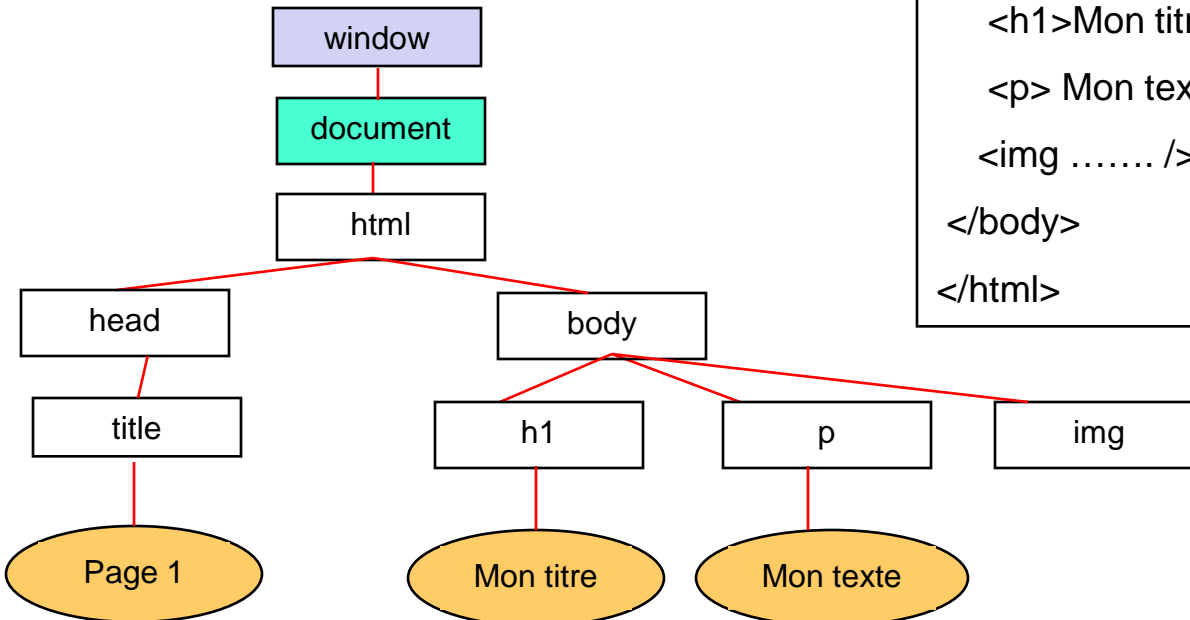
L'organisation hiérarchique de ces différents objets



nœud élément



nœud texte



```
<html>
<head>
  <title> Page 1 </title>
</head>
<body>
  <h1>Mon titre </h1>
  <p> Mon texte </p>
  <img ..... />
</body>
</html>
```

Combien d'objets ont été instanciés par le navigateurs ?

L'API DOM : Accéder à des nœuds éléments

Tout code javascript aura pour problématique de « récupérer » dans le DOM, des objets de type nœud élément afin de les modifier et donc de modifier le visuel de la page.

2 types d'objets ont la capacité de retrouver des nœuds éléments dans le DOM :

- L'objet **document** : sa recherche s'effectue dans tout l'arbre DOM
- Les **nœuds éléments** : la recherche s'effectue parmi ses descendants (rappel : les enfants font partie des descendants)

L'API DOM : Accéder à des nœuds éléments

- Pour accéder à **UN** nœud-élément précis on peut utiliser **l'id** de ce nœud : **document.getElementById(id recherché)**

Ex : `document.getElementById("lien1");`

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à un **tag** (balise) donné :

document.getElementsByTagName(nom du tag) : recherche dans le DOM

Ex : `document.getElementsByTagName("div");`

ou **noeudElement.getElementsByTagName(nom du tag)** : La recherche s'effectue parmi les descendants du nœud élément

Ex : `document.getElementById("div1").getElementsByTagName("div");`

L'API DOM : Accéder à des nœuds éléments

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à un **nom** (attribut name) donné :

document.getElementsByTagName(nom recherché)

Ex : document.getElementsByTagName("btnTransport");

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à une classe donnée (attribut class) :

document.getElementsByTagName(classe recherchée): recherche dans tout le dom

nœudElement.getElementsByTagName(classe recherchée): recherche dans les descendants du nœud élément

L'API DOM : Accéder à des nœuds éléments

- Pour accéder à **un ENSEMBLE de nœuds-éléments** correspondant à un **sélecteur css**:

document.querySelectorAll("sélecteur css"): recherche dans tout le dom

Ex : `document.querySelectorAll("#div1>:first-child");` ramènera le premier enfant de div1

nœudElement.querySelectorAll("sélecteur css"): recherche dans les descendants du nœud élément

- Pour accéder à **UN nœud élément** correspondant au premier élément vérifiant le sélecteur css.

document.querySelector("sélecteur css"): recherche dans tout le dom

Ex : `document.querySelector("div:first-child");` ramènera la première div de rang 1 dans sa fratrie

nœudElement.querySelector("sélecteur css"): recherche dans les descendants du nœud élément

L'API DOM: accéder à des nœuds éléments



- Les méthodes `getElementsByTagName`, `getElementsByName`, `getElementsByClassName` et `querySelectorAll` ramènent toujours leur résultat sous forme de collection (tableau) même s'il n'y a aucun nœud dans celle-ci.
- Seul l'objet document dispose des méthodes `getElementById` et `getElementsByName`
- Attention à ne pas confondre un nœud élément avec une collection de nœuds éléments.

Ex :

`document.getElementsByTagName("div").getElementsByTagName("p")` génère une erreur d'exécution.

Alors que

`document.getElementsByTagName("div")[0].getElementsByTagName("p")` recherchera tous les descendants de type p de la première div

L'API DOM: accéder à des nœuds éléments

- Lorsqu'une méthode ramène au plus un élément (getElementById et querySelector) , elle ramène **null** si aucun élément ne correspond à la recherche.



- Lorsqu'une méthode peuvent ramener plusieurs éléments, elle ramène un tableau vide lorsqu'aucun élément ne correspond à la recherche

- **Null = aucun objet**

- **Tableau vide = 1 objet (le tableau) mais avec 0 case**

L'API DOM: accéder à des nœuds éléments

.....

```
<body> ①
  <div id="div1"> ②
    <img src=img1.jpg" id="img1" class="c1" /> ③
    <img src=img2.jpg" id="img2" class="c2" /> ④
  </div>
  <div id="div2"> ⑤
    <img src=img3.jpg" id="img3" /> ⑥
    <img src=img4.jpg" id="img4" class="cl1" /> ⑦
  </div>
</body>
</html>
```

Donnez le n° des éléments sélectionnés

(1) document.getElementById("div1")

(2) document.getElementById("IMG1")

(3) document.getElementsByTagName("img")

(4) document.querySelector("img:last-child")

(5) document.querySelectorAll("img:last-child")

L'API DOM: accéder à des nœuds éléments

.....

```
<body> ①
  <div id="div1"> ②
    <img src=img1.jpg" id="img1" class="c1" /> ③
    <img src=img2.jpg" id="img2" class="c2" /> ④
  </div>
  <div id="div2"> ⑤
    <img src=img3.jpg" id="img3" /> ⑥
    <img src=img4.jpg" id="img4" class="cl1" /> ⑦
  </div>
</body>
</html>
```

Donnez le n° des éléments sélectionnés

(1) `document.body.querySelector(":first-child")`

(2) `document.getElementById("div2").querySelector(":first-child")`

(3) `document.querySelector("div").getElementsByTagName("img")`

Les attributs communs aux noeuds

Le type du nœud

nœud.nodeType : type du nœud

1: nœud élément

3: nœud texte

8: nœud commentaire

Le nom du nœud

nœud.nodeName : #text pour les nœuds textes, le nom de la balise pour les nœuds éléments (en majuscule. Ex DIV,P,TITLE)

La valeur du nœud:

nœud.nodeValue: la valeur du texte pour un nœud texte, null pour les nœuds éléments

Les autres attributs d'un nœud sont les attributs html (ex src pour une image, href pour un lien , title ou id pour tous)

Les attributs communs aux noeuds

Le contenu HTML d'un nœud élément

nœud.innerHTML: le code HTML ou le texte qui se trouve entre la balise ouvrante et la balise fermante

Ex :

```
<div id="div1">
```

```
    <p id="p1"> paragraphe1 </p>
```

```
</div>
```

document.getElementById("div1").innerHTML a pour valeur :

```
<p> paragraphe 1 </p>
```

document.getElementById("p1").innerHTML a pour valeur :

```
paragraphe 1
```

QUAND

La notion d'évènement

Comment gérer les actions utilisateurs ?

Lorsqu'une action utilisateur détectable par le navigateur se produit (manipulation de la souris, du clavier, ...), le navigateur va envoyé aux éléments html concernés un **message** indiquant que l'évènement (par ex **click**) vient de se produire.

A charge pour l'élément HTML de réagir ou non à cet évènement.

Notre travail dans cette phase va donc consister à indiquer les éléments qui doivent réagir à un évènement donné.

Certains évènements ne pourront être interceptés que par certain type d'éléments :

Ex : l'évènement submit ne peut être géré que par les éléments `<form>`

Les évènements utilisateurs

| Evènement | Description |
|------------------|--|
| click | Se produit lorsque l'utilisateur clique sur l'élément associé à l'évènement |
| change | Survient lorsque l'élément perd le focus et que la valeur de l'élément a changé par rapport à la valeur initiale |
| blur | Survient lorsque l'élément perd le focus |
| focus | Survient lorsque l'élément prend le focus (pour les champs saisissables , le curseur se trouve dans l'élément) |
| keypress | Survient lorsque l'utilisateur appuie sur une touche du clavier |
| mousedown | survient lorsque l'utilisateur appuie sur un bouton de la souris |
| mousemove | survient lorsque l'utilisateur bouge la souris |
| mouseout | survient lorsque le pointeur de souris quitte une zone sensible |
| mouseover | Se produit lorsque le pointeur de souris entre dans une zone sensible (sans cliquer) |
| reset | Survient lorsque l'utilisateur clique sur un bouton reset |
| select | Survient lorsque l'utilisateur sélectionne tout ou partie d'un texte |
| submit | Survient lorsque l'utilisateur clique sur un bouton submit |
| dblclick | survient lorsque l'utilisateur double-clique sur l'élément associé à l'évènement |
| resize | Se produit lorsque l'utilisateur redimensionne la fenêtre du navigateur |

Les évènements utilisateurs

| Evènement | Supporté par quel élément html |
|---|--|
| click,dblclick | <a>, <address>, <area>, , <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <dfn>, <div>, <dl>, <dt>, , <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, , <input>, <kbd>, <label>, <legend>, , <map>, <object>, , <p>, <pre>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, , <var> |
| change | <input type="text">, <select>, <textarea> |
| blur, focus | <a>, <acronym>, <address>, <area>, , <bdo>, <big>, <blockquote>, <button>, <caption>, <cite>, <dd>, , <dfn>, <div>, <dl>, <dt>, , <fieldset>, <form>, <frame>, <frameset>, <h1> to <h6>, <hr>, <i>, <iframe>, , <input>, <ins>, <kbd>, <label>, <legend>, , <object>, , <p>, <pre>, <q>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, , <var> |
| keypress | <a>, <acronym>, <address>, <area>, , <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, , <dfn>, <div>, <dt>, , <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, <input>, <kbd>, <label>, <legend>, , <map>, <object>, , <p>, <pre>, <q>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, , <var> |
| mousedown, mouse move, mouseout, mouseover | <a>, <address>, <area>, , <bdo>, <big>, <blockquote>, <body>, <button>, <caption>, <cite>, <code>, <dd>, <dfn>, <div>, <dl>, <dt>, , <fieldset>, <form>, <h1> to <h6>, <hr>, <i>, , <input>, <kbd>, <label>, <legend>, , <map>, , <p>, <pre>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <tbody>, <td>, <textarea>, <tfoot>, <th>, <thead>, <tr>, <tt>, , <var> |
| reset, submit | <form> |
| select | <input type="text">, <textarea> |
| resize | <a>, <address>, , <big>, <blockquote>, <body>, <button>, <cite>, <code>, <dd>, <dfn>, <div>, <dl>, <dt>, , <fieldset>, <form>, <frame>, <h1> to <h6>, <hr>, <i>, , <input>, <kbd>, <label>, <legend>, , <object>, , <p>, <pre>, <samp>, <select>, <small>, , , <sub>, <sup>, <table>, <textarea>, <tt>, , <var> |

Evenement "Système"

Ces événements sont générés par le navigateur lorsqu'il a terminé une action de chargement ou une action de déchargement

| Evènement | Description |
|---------------|--|
| load | Survient lorsque le navigateur a fini un chargement (page, image ,...) |
| unload | Survient lorsque le navigateur décharge la page |

| Evènement | Supporté par quel élément |
|---------------|--|
| load | window, <frame>, <frameset>, <iframe>, , <link>, <script> |
| unload | window, <frameset> |

Mettre en place la gestion d'évènement

Comment définir que les éléments réagissent aux évènements?

On a longtemps intégré le code directement dans la page HTML :

<balise `on`event="fonction javascript">

`event` faisant référence à un des évènements définis précédemment.

ex

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

```
<form id="form" action="page.php" onsubmit="verifier()">
```

```

```

Mettre en place la gestion d'évènement

De plus en plus de sites, dissocie l'interactivité (donc le javascript) de la page html (même principe que pour la mise en forme)

La mise en place de la gestion des évènements se fera donc entièrement dans le fichier javascript

Dans ce cours, seule cette solution sera autorisée.

2 possibilités pour gérer cette interactivité :

- en utilisant les attributs onevent des éléments
- en utilisant l' API DOM

Mettre en place la gestion d'évènement

1) En utilisant l'API DOM 0 et les attributs onevent (onclick, onmouseover ,.....)

objet.onevent=nomFonctionAExecuter;

Ex :pour mettre une bordure rouge sur l'image 1 lorsqu'on clique dessus

```
document.getElementById("img1").onclick=clickImage1;
```

```
function clickImage1() {  
    document.getElementById("img1"). style.border="1px solid red";  
}
```

Mettre en place la gestion d'évènement

2)En utilisant les spécifications du DOM 2

objet.addEventListener("event",nomFonction)

Ex

```
document.getElementById("img1").addEventListener("click", borderRouge);
```

Avantage:

- C'est la norme donc moins de risque que cela ne fonctionne plus un jour .
- Dans certains cas, on veut mettre en place plusieurs écouteurs pour un même événement. Seule cette solution nous permet cela.

Ex : on veut à la fois gérer 2 écouteurs de sur l'événement click : un qui permet de modifier la bordure et l'autre qui permet de changer l'image

```
document.getElementById("img1"). addEventListener("click", borderRouge);
```

```
document.getElementById("img1"). addEventListener("click", changerImage);
```


Mettre en place la gestion d'évènement

- On peut facilement supprimer une gestion d'évènements

`nœud.removeEventListener("event",nomFonction)`

Ex : `document.getElementById("img1").removeEventListener("click", borderRouge);`

Inconvénient:

Il n'y en a qu'un mais suffisamment gênant pour empêcher sa mise en pratique pour certains sites : Ce n'est pas **compatible Internet Explorer <9**

Cas d'internet explorer

Internet explorer, jusqu'à la version 8 comprise, ne gèrait pas `addEventListener`. A la place, on dispose de :

`attachEvent("onevent",nomFonction)`

et

`detachEvent("onevent",nomFonction)`

Ex

```
window.attachEvent("load",gererInteractivite);
```

```
window.attachEvent("load",fonction1);
```

L'explication par l'exemple

```
<html>
..
<body>
  <script src="code.js"></script>
  <div id="etu1"></div>
  <div id="etu2"></div>
  <div id="etu3"></div>
</body>
</html>
```

L'utilisateur :

Il clique sur etu1

Puis il clique sur etu2

```
window.addEventListener("load" , distribuerAction);

function distribuerAction(evt) {
  document.getElementById("etu1").addEventListener("click", doA1);
  document.getElementById("etu2").addEventListener("click", doA2);
  document.getElementById("etu2").addEventListener("click", doA3);
  document.getElementById("etu3").addEventListener("click", doA3);
}

function doA1(evt) {
  lever le bras droit
}

function doA2(evt) {
  lever le bras gauche
}

function doA3(evt) {
  lever le pied gauche
}
```

A retenir

Toute instruction se trouvant en dehors de toute fonction va s'exécuter au moment où le navigateur charge le fichier javascript (c'est à dire lorsqu'il analyse le fichier script suite à son chargement à l'aide de la balise script.)

Toute instruction se trouvant dans une fonction n'est exécutée que lors de l'appel de la fonction. Elle peut donc ne jamais être exécutée.

On ne peut accéder aux éléments du DOM que lorsqu'ils sont créés i.e lorsque le dom est chargé i.e après la diffusion de l'événement load auprès de l'objet window. Il n'y aura donc généralement qu'une instruction hors des fonctions :

```
window.addEventListener("load",nomFonction)
```

COMMENT

Actions = fonction

Toutes les transformations à apporter à la page lors de la survenue d'un événement utilisateur ou d'un événement système seront programmés dans une **fonction**

Pour déclarer une fonction en javascript :

```
function nomFonction(parametres) {  
    .....  
}
```

L'API DOM: Modifier le contenu du DOM

Modifier le contenu de la page

L'api dom nous permet d'accéder et de modifier la valeur des attributs d'un nœud élément

Pour obtenir la valeur d'un attribut

`noeud.getAttribute("nom attribut")`

Ex : `document.getElementById("img1").getAttribute("alt")` : permet d'obtenir la valeur de l'attribut alt de l'élément html ayant pour id img1

Tout attribut HTML correspond à **une propriété de l'objet**. On peut donc aussi accéder directement à la propriété

`noeud.attribut`

ex : `document.getElementById("img1").alt`

L'API DOM : Modifier le DOM



Attention pour certains d'attributs, la valeur obtenue n'est pas la même si on utilise la méthode `getAttribute()` ou la propriété de l'objet

- **Pour les attributs dont la valeur est un chemin**

Ex: ``

`document.getElementById("img1").src` retournera le **chemin absolu** du fichier image soit `http://www.monsite.com/images/toto.jpg`

`document.getElementById("img1").getAttribute("src")` retournera le **chemin indiqué dans le code html** soit `"images/toto.jpg"`

L'API DOM : Modifier le DOM



Pour les attributs d'état (ex checked, selected, ..)

Ex: `<input id="i1" type="checkbox" checked="checked" />`

`document.getElementById("i1").getAttribute("checked")` retournera toujours la valeur de l'attribut soit "checked" et cela que la case soit décochée ou non par l'internaute

`document.getElementById("i1").checked` retournera une valeur booléenne indiquant si la case est cochée ou non. Si l'internaute coche la case, la propriété `checked` a pour valeur `true`, si il décoche la case elle a pour valeur `false`.

L'API DOM: modifier le DOM

Pour modifier la valeur d'un attribut:

`element.setAttribute("nom attribut", "valeur attribut")`

Ex : `document.getElementById("img1").setAttribute("src", "img2.png")` :
modifie la valeur de l'attribut src de l'élément html ayant pour id img1 ainsi que la valeur de sa propriété src.

On peut aussi utiliser la notation objet:

`element.attribut=valeur`

ex : `document.getElementById("img1").src="img2.png"`

L'API DOM : Modifier le DOM

Ajouter des nœuds dans le DOM

L'API DOM dispose de méthodes permettant d'ajouter des nœuds dans le dom.

document.createElement('nom balise') : création d' un nœud élément

document.createTextNode('texte') : crée un nœud texte

noeudParent.appendChild(nœud à ajouter) : permet de le ranger dans l'arbre comme dernier nœud enfant du nœud parent

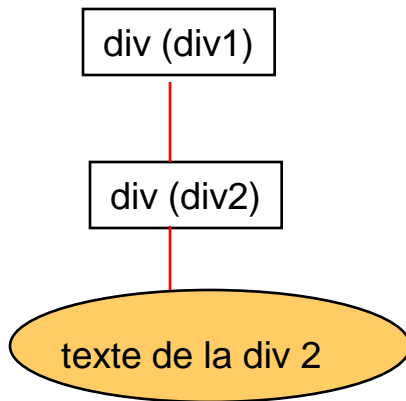
noeudParent.insertBefore(nœud à ajouter, noeud1) : permet de le ranger comme noeud enfant du nœud parent, juste avant le noeud1

L'API DOM : Modifier le DOM

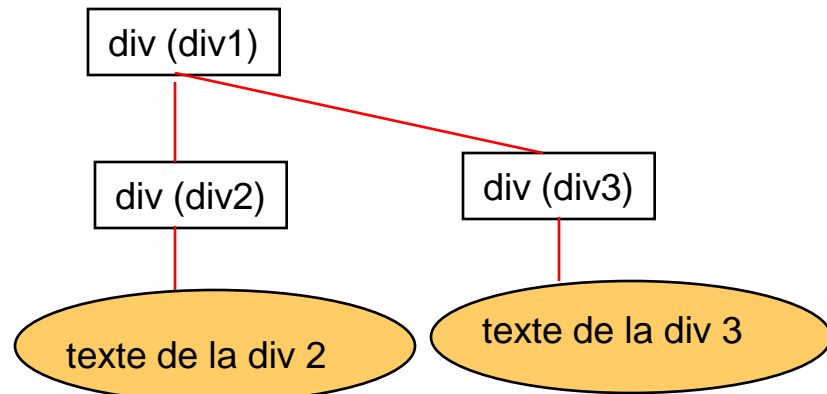
Exemple :

```
var nouvDiv= document.createElement("div");  
nouvDiv.setAttribute("id","div3");  
var texte= document.createTextNode("texte de la div 3");  
nouvDiv.appendChild(texte)  
document.getElementById("div1").appendChild(nouvDiv)
```

Avant



Après



L'API DOM : Modifier le DOM

Ajouter du code HTML : la navigateur transforme ce code html en éléments du DOM

noeudParent.innerHTML=code HTML : on remplace le contenu d'un élément par le code HTML fourni.

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

```
div.innerHTML="<p> Voici un nouveau  paragraphe </p>";
```

noeud.insertAdjacentHTML("où l'ajouter", code HTML) : permet d'insérer le code HTML à l'intérieur ou autour du nœud choisi.

"où l'ajouter" peut prendre 4 valeurs :

"beforeEnd" : le code HTML est ajouté après le dernier enfant

"afterBegin" : le code HTML est ajouté avant le premier enfant

"afterEnd" : le code HTML est ajouté après l'élément choisi

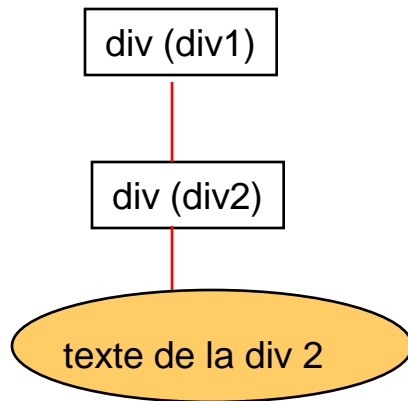
"beforeBegin" : le code HTML est ajouté avant l'élément choisi

L'API DOM : Modifier le DOM

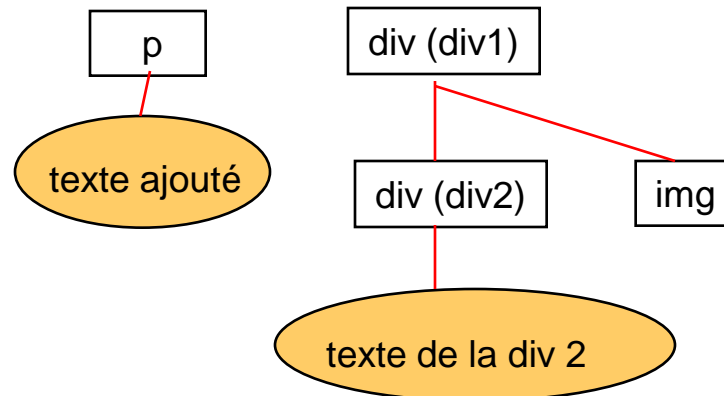
Exemple :

```
var div1= document.getElementById("div1");  
div1.insertAdjacentHTML("beforeEnd","<img src='img1' />");  
div1.insertAdjacentHTML("beforeBegin","<p>Texte ajouté </p>");
```

Avant



Après



L'API DOM : Modifier le DOM

On peut aussi **cloner** un nœud élément (avec ou non ses descendants).

`noeudACloner.cloneNode(avecDescendant)` : retourne un nœud identique au nœud cloné (avec tous ses descendants si avecDescendant = true, juste le nœud sinon).

On peut **supprimer** un nœud

`noeudParent.removeChild(noeudASupprimer)`

Ou dans les dernières versions:

`nœud.remove()`

On peut le **déplacer** dans la hiérarchie:

`noeudParent.appendChild(noeudADéplacer)`

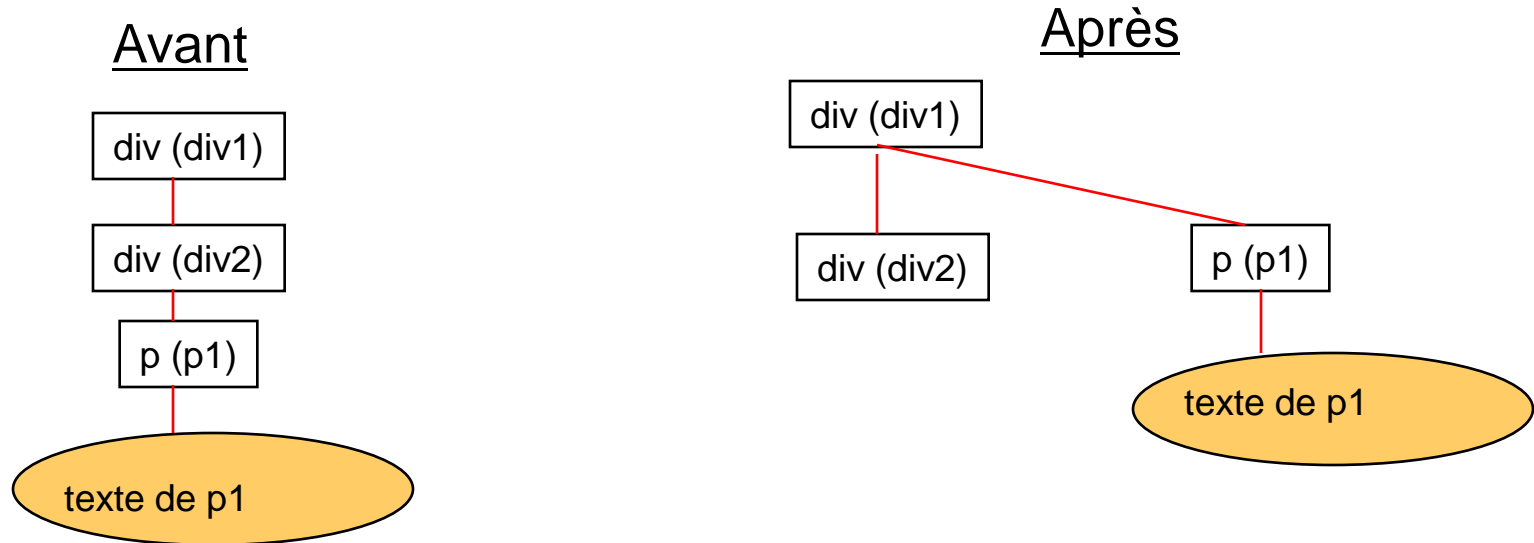
ou `noeudParent.insertBefore(noeudADeplacer,noeud1,)`

Un nœud ne pouvant avoir qu'un seul parent, le fait de le rattacher à un nouveau nœud parent, élimine sa relation à l'ancien nœud parent

L'API DOM : Modifier le DOM

Exemple :

```
var eltP1= document.getElementById("p1");  
document.getElementById("div1").appendChild(eltP1) ;
```



On déplace le nœud avec toute sa descendance. De même si on supprime un nœud , on supprime le nœud et toute sa descendance.

L'API DOM: Modifier la mise en forme

Modifier la mise en forme d'éléments dans la page

Lorsqu'on veut récupérer ou modifier la mise en forme d'un élément ,
Pour mettre en forme

- On peut utiliser l'attribut class

`nœud.getAttribute("class") - nœud.setAttribute("class","valeur")`

ou

`nœud.className - nœud.className=valeur`

Avec HTML5 :

`var liste =noeud.classList;`

`liste.add('classe');`

`liste.remove('classe');`

`liste.toggle('classe');`

`liste.contains('classe');`

L'API DOM: les propriétés de style (css)

On peut obtenir les propriétés de style de l'élément

`nœud.getAttribute("style").nomProprieteCss`

ou `nœud.style. nomProprieteCss`



le nom de la propriété css n'est pas identique à celui de la feuille de style : **le tiret est remplacé par une majuscule.**

Ex background-color devient backgroundColor

font-size devient fontSize

L'API DOM: les propriétés de style (css)

On peut modifier les propriétés de style de l'élément

`nœud.setAttribute("style","propriété-css:valeur; propriete-css:valeur;....")`

ou `nœud.style.nomProprieteCss=valeur;`

Ex : Pour modifier la couleur d'arrière-plan et la bordure de p1

`var elt1=document.getElementById("p1");`

Solution 1

`elt1.style.backgroundColor="red";`

`elt1.style.border="2px solid black";`

Solution 2

`elt1.setAttribute("style","background-color:red;border:2px solid black;");`

L'API DOM: les propriétés de style (css)



L'attribut style fait référence à la valeur de l'attribut style défini dans l'élément HTML (ou via javascript) , il ne tient pas compte des propriétés de style héritées par la feuille de style.

Si l'on veut pouvoir obtenir la propriété de style "globale" (html +css) il faut utiliser :

`element.currentStyle //IE`
ou `window.getComputedStyle(element, null) //FFx et Opera`

Le mot clé this

Dans un certain nombre de cas, notamment lorsqu'on souhaite associer un comportement identique à plusieurs éléments html, il est souvent nécessaire de pouvoir faire référence à l'élément interceptant l'événement. En javascript, la fonction s'exécute dans le contexte de l'élément source. C'est le but du mot clé **this**.

Ex :

```
document.getElementById("para1").addEventListener("click",quiEstClique);  
document.getElementById("para2").addEventListener("click",quiEstClique);  
;
```

```
function quiEstClique () {  
  alert (this.id);  
  alert(this.innerHTML);  
}
```

```
page html    <p id="para1"> paragraphe 1 </p>  
               <p id="para2">paragraphe 2</p>
```

Si on clique sur

- le premier élément p , il s'affichera para1 puis paragraphe1
- sur le 2eme il s'affichera para2 puis paragraphe2

L'appel de fonction

Ecrire :

`window.addEventListener("load",initialiser)`

et `window.addEventListener("load",initialiser())` ne veut pas du tout dire la même chose.

cas 1: `window.addEventListener("load",initialiser)`

On donne le nom (et donc le code) de la fonction à exécuter lorsque l'événement load sera diffusé à l'objet window. C'est donc le moteur javascript du navigateur qui se chargera de l'exécution **effective** de la fonction (de manière schématique il ajoute les parenthèses pour demander l'exécution)

L'appel de fonction

cas 2 : `window.addEventListener("load",initialiser())`

On demande l'affectation du résultat de la fonction `initialiser` associé à l'écouteur `"load"`. C'est possible, mais dans ce cas, le résultat doit être de type `Function`. La fonction `initialiser` est donc **exécutée immédiatement**

Concrètement, vous ne serez pas amené à utiliser cette syntaxe cette année , cela fait partie des techniques de javascript avancé.



En résumé : il ne faut **jamais mettre de parenthèses** après le nom de la fonction écouteur

L'appel de fonction

Quels sont les paramètres transmis aux fonctions écouteur ?

Il y a un seul paramètre transmis aux fonctions "ecouteurs" , c'est un objet de type **Event**.

Cet objet contient un certain nombre d'informations sur l'événement. Par exemple, la position de la souris lorsque le click a eu lieu, l'élément html cible de l'événement, etc.

L'appel de fonction

Puis-je transmettre des paramètres à une fonction écouteur?

C'est possible mais cela nécessite l'utilisation de techniques avancées de javascript (**closure**)

Ce n'est pas au programme de la première année.

Donc pour cette année, le seul paramètre reçu par les fonctions écouteurs est le paramètre evt. Ce paramètre est transmis par la navigateur lorsque l'action a lieu.

L'objet Event

| Nom de la propriété | description | I.E<9 |
|---------------------|--|-------------------|
| bubbles | indique si l'événement bouillonne (est diffusé à toute l'arborescence) | inexistant |
| cancelable | indique si l'événement est annulable | inexistant |
| target | indique le nœud cible de l'évènement | srcElement |
| currentTarget | indique le nœud actuel réagissant à l'évènement | inexistant |
| type | indique le type d'événement (click, mouseover,...) | inexistant |
| timestamp | indique la date et l'heure de l'évènement | timestamp |
| which | Code de la touche frappée | keyCode |
| clientX, clientY | Position de la souris | ClientX,clientY |
| Nom de la méthode | | |
| stopPropagation() | arrête la propagation | cancelBubble=true |
| preventDefault() | annule l'événement. Cela empêche le navigateur de procéder à l'action par défaut pour l'événement (par ex, activer l'url d'un lien, soumettre un formulaire) | returnValue=false |

Ordre d'exécution

Exemple :

```
<html>
  <head>
    <script type="text/javascript" src="ex.js"></script>
  </head>
  <body>
    <div>
      
    </div>
  </body>
</html>
```

```
window.addEventListener("load",initialiser);
```

1

```
function initialiser() {
```

2

```
  document.getElementById("img1").addEventListener("mouseover",afficherBord);
```

3

```
  document.getElementById("img1").addEventListener("mouseout", cacherBord);
}
```

```
function afficherBord(evt) {
```

4

```
  document.getElementById("img1").style.border="1px solid red";
}
```

```
function cacherBord(evt) {
```

5

```
  document.getElementById("img1").style.border="none";
}
```

Ordre d'exécution

Déroulement :

Au temps t0 : l'internaute tape l'url de la page, le navigateur reçoit la page et commence à l'analyser

au temps t1 : il analyse `<script....src="ex.js">` , il charge le fichier et exécute les instructions hors fonctions donc ici l'instruction **1** . Il enregistre un écouteur permettant d' exécuter la fonction *initialiser* lorsque l'événement load sera diffusé

au temps t2 : la page est entièrement analysée et chargée, le DOM est créé. Le navigateur diffuse l'événement load qu'intercepte la fenêtre et donc elle exécute la fonction *initialiser* (**2** et **3**). Dans cette fonction, on met en place l'interactivité sur l'image 1 : on enregistre un ecouteur permettant d'exécuter la fonction *afficherBord* lorsqu'un événement mouseover sera diffusé sur l'image1 (lorsque l'internaute passe sa souris). Un autre écouteur est défini sur l'émission de l'événement qui permettra l'exécution de la fonction *cacherBord*.

au temps t3 : L'internaute passe la souris sur l'image1 , l'événement mouseover est envoyé à l'image 1 et la fonction *afficherBord* (**4**) est exécutée. On modifie la propriété de style border de l'image 1. Les ascendants de image1 sont aussi avertis de l'événement mais ils ne l'écoutent pas donc rien ne se produit.

Ordre d'exécution

au temps t4 : L'internaute enlève la souris de l'image1, l'événement mouseout est diffusé à l'image 1 et , la fonction *cacheBord* (5) est exécutée. On modifie la propriété de style border de l'image 1. Les ascendants de image1 sont aussi avertis de l'événement mais ils ne l'écoutent pas donc rien ne se produit.

au temps t5 : L'internaute passe la souris sur l'image1 , l'événement mouseover est diffusé à l'image 1 et la fonction *afficherBord* (4) est exécutée. On modifie la propriété de style border de l'image 1. Les ascendants de image1 sont aussi avertis de l'événement mais ils ne l'écoutent pas donc rien ne se produit.

au temps t6 : L'internaute tape une autre url, la page est déchargée. L'événement "unload" est diffusée à la fenêtre mais elle ne l'écoute pas donc rien ne se produit.

En programmation événementielle, le code s'exécute à des moments différents.

Lorsqu'on associe une fonction à la diffusion possible d'un événement pour un élément donné, la fonction n'est pas exécutée lors de l'association mais lorsque l'événement est réellement diffusé.

La notion de propagation d'événement

La gestion des événements est un phénomène plus compliqué qu'il en a l'air au premier abord

Soit le code suivant associé à la css suivante

...

```
<div id="div1">
```

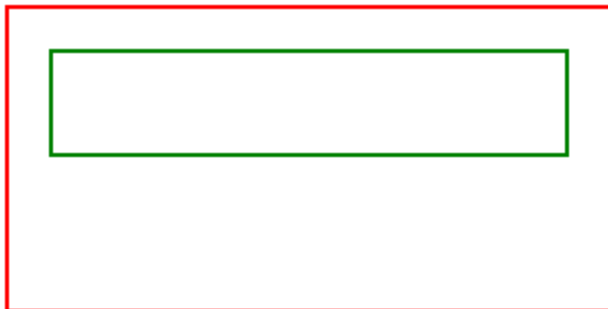
```
#div1 { height:150px; border:2px solid red;}
```

```
  <div id="div2">
```

```
#div2 { height:50px; border:2px solid green; margin:20px;}
```

```
  </div>
```

```
</div>
```



Lorsque je clique à l'intérieur de la boîte verte est-ce que je clique aussi à l'intérieur de la boîte rouge ?

La notion de propagation

```
<html>
```

```
.....
```

```
<body>
```

```
  <div id="div1">
```

```
    <div id="div2">
```

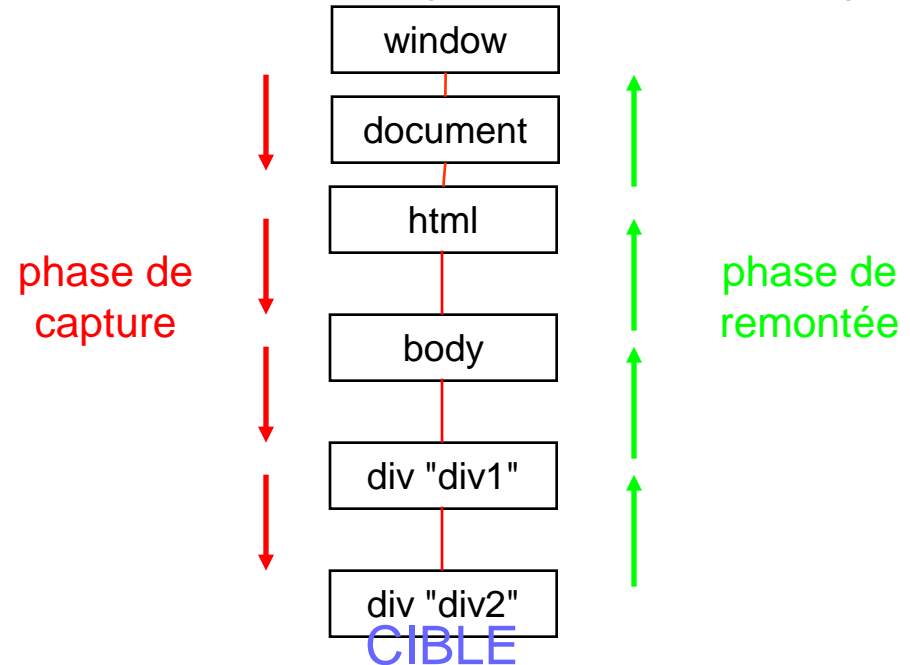
```
      </div>
```

```
    </div>
```

```
  </body>
```

```
</html>
```

Si on clique sur l'image, l'événement se propage ainsi



La **cible** est toujours l'élément en avant-plan (donc ici div 2)

Il va donc y avoir 2 phases distinctes lorsqu'un événement se produit :

phase de capture : tous les éléments en lien avec la cible (de window jusqu'à la cible) sont prévenus qu'un événement de type click vient d'avoir lieu

phase de remontée : Dans cette phase on repart de la cible et on remonte les ancêtres ayant un lien avec la cible. A nouveau chacun est averti qu'un événement a eu lieu

La notion de propagation

Cette propagation existe systématiquement, malgré tout :

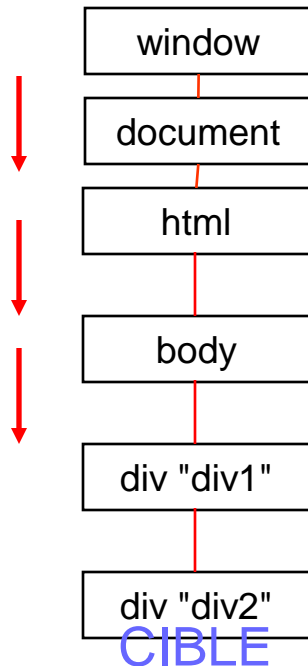
- Généralement, on intervient pendant une seule de ces phases
- Bien souvent, seule la cible réagira à l'événement

Par défaut, lorsqu'on utilise `addEventListener` ou les attributs `"onaction"`, on se place dans la phase de remontée

Pour intercepter un événement pendant la phase de capture, il faut fournir **un troisième paramètre** à `addEventListener`
`objet.addEventListener("action",fonction,true)`

La notion de propagation

Par l'intermédiaire de l'objet Event, on peut par contre empêcher cette propagation :



On arrête la propagation lors de la phase de capture sur l'objet div1 :

```
document.getElementById("div1").addEventListener ("click",stop,true)

function stop(evt) {
    evt.stopPropagation();
}
```

La notion de propagation

Il existe un autre type de propagation, ce que j'appelle la propagation "native"

Certains éléments sont naturellement interactifs :

les liens, les boutons radios, les cases à cocher , l'élément submit des formulaires, etc.

Lorsqu'on met en place une écoute de cet évènement (ex click) , notre code s'exécute **avant** le code exécuté par le navigateur.

Il est alors possible d'empêcher son exécution en faisant appel à la méthode **preventDefault()**

evt.preventDefault();

Les bases du langage

Les commentaires :

// ou /* */

Les variables

- En javascript sont reconnus 5 types de données :

- **nombre** (entier ou réel) (number)
- **chaîne de caractère** (string)
- **booléen** (boolean)
- **objet** (object)
- **undefined**: lorsque la variable n'est pas initialisée

- **Pas de déclaration de type en javascript.**

Le type est directement attribué par le langage en fonction du contenu de la variable. La variable n'a pas de type fixe, il évolue en fonction du contenu.

Rque : Dans la version 3 de ECMAScript le typage des variables apparaît.

- Les variables doivent malgré tout être déclarées à l'aide de **var nomVariable;**

Les bases du langage

Même règle de nommage qu'en java (par contre pas de caractère accentué).

JavaScript est aussi **sensible à la casse** : var1 et Var1 sont 2 variables différentes.



La **déclaration de variable** en javascript peut être **implicite**, mais dans ce cas vous prenez des risques : si vous orthographiez mal les variables, à chaque nouvelle orthographe, une nouvelle variable sera créée.

Les bases du langage

Rque : Le type de la variable étant défini de manière automatique au cours de l'exécution il peut parfois être intéressant de le connaître :

Pour cela vous pouvez utiliser la fonction `typeof` :

Ex

- `var chaine='moi' ;`
`typeof(chaine)` a pour résultat String
- `chaine=0 ;`
`typeof(chaine)` a pour résultat Number
- `chaine=0.0;`
`typeof(chaine)` a pour résultat Number
- `chaine=false;`
`typeof(chaine)` a pour résultat Boolean

Les bases du langage

JavaScript se charge lui-même des conversions de type.

Malgré tout, 2 fonctions permettent de lui indiquer explicitement cette conversion :

`parseInt(String,base)` convertit une chaîne en entier (système hexadécimal).

base indique dans quelle base est exprimée la chaîne à transformer (si elle est omise, il considère que la valeur est en base 10)

ex `parseInt('12abc',10)` a pour résultat 12

```
var chaine1 = "123" ;
```

```
var chaine2 = "456" ;
```

```
var chaine3 = chaine1 + chaine2;
```



```
chaine3="123456" →
```

```
var chaine4 = parseInt(chaine1,10) + parseInt(chaine2,10)
```

```
chaine4=579
```

```
var chaine5= parseInt("111",2)
```

```
chaine5=7
```

`parseFloat(String,base)` converti une chaîne en float dans une base.

ex `parseFloat('12.8abc',10)` a pour résultat 12.8

`parseFloat("abc",10)` a pour résultat NaN (not a number)

Les bases du langage

A l'inverse, on peut aussi transformer un **nombre en String** à l'aide de la méthode **toString()**:

Ex:

```
var nb1 =123 ;
```

```
var nb2 = 456 ;
```

```
var chaine3 = nb1 + nb2;           ↔           chaine3=579; (typeof Number)
```

```
var chaine4 = nb1.toString() +nb2 ↔ chaine4="123456"
```

Les bases du langage

Les opérateurs : idem java

- arithmétique : + , - , * , / , %
- affectation : = , += , -= , *= , /= , % =
- incrémentation : ++ , -- (idem java : ++nb1 ou nb1++ : se reporter au cours java pour voir la différence)
- comparaison : == , != , === , !== , > , < , <= , >=
 - == : compare les 2 variables en essayant de convertir le type
Ex 0=="0" a pour résultat vrai
 - === : compare la valeur **et** le type
Ex 0=== "0" a pour résultat faux
- logiques:
 - non : ! ,
 - et : && ,
 - ou : ||
- ternaire : condition ? val1 : val2

Les bases du langage

Les instructions conditionnelles :

- **si.. sinon :**

```
if (condition) {  
    bloc instructions;  
}  
else {  
    bloc instructions;  
}
```

- **instruction selon :**

```
switch (variable) {  
    case valeur1 :  
        bloc 1;  
        break;  
    case valeur2 : case valeur3:  
        bloc2;  
        break;  
    default : bloc default;  
}
```

Les bases du langage

Les instructions répétitives :

- **tant que fin tant que:**

```
while (condition) {  
    bloc instructions;  
}
```

- **répéter.... tant que**

```
do {  
    bloc;  
} while (condition);
```

- **pour compteur allant de debut à fin par pas de 1**

```
for (var compteur=debut; compteur<= fin; compteur++) {  
    instructions;  
}
```

Comme en java :

- break pour arrêter la boucle
- continue pour passer un tour

Les bases du langage

Déclarations de fonctions :

```
function nomDeLaFonction (param1,param2,...) {  
    instructions;  
}
```

Si la fonction calcule un résultat :

```
function nomFonction (param1,param2) {  
    instructions;  
    return valeur ou variable  
}
```

Appel d'une fonction

```
nomFonction(valeur param1,valeur param2);  
ou  
variable=nomFonction(valeur param1,valeur param2);
```

Les bases du langage



Un paramètre ne doit jamais être déclaré à l'aide de var dans l'entête de la fonction.

```
function fonction1(var param1) {  
}
```

Les bases du langage

Exemple

```
function calculerPuissance (nombre,puissance) {  
    var resultat=1;  
  
    for (var i=1;i<=puissance;i++) {  
        resultat=resultat*nombre;  
    }  
    return resultat;  
}
```

Pour l'appeler :

```
var nb1=calculerPuissance (2,3); // calcule 23
```

↑
↓
nb1 = 8;

Les bases du langage

La notion de paramètres optionnels

En javascript , on peut , lors de l'appel d'une fonction, donner moins de valeurs que le nombre de paramètres demandé par la signature de la fonction. C'est généralement ce qu'on appelle des paramètres optionnels.

Ex :

```
function calculerPuissance(nombre,puissance) {  
    if (arguments.length==1) { // permet de savoir combien il y a de valeurs en paramètres  
        puissance=1;  
    }  
    var resultat=1;  
  
    for (var i=1;i<=puissance;i++) {  
        resultat=resultat*nombre;  
    }  
    return resultat;  
}
```

calculerPuissance(3) aura pour résultat 3 , calculerPuissance(3,2) aura pour résultat 9



arguments est un tableau où sont stockées toutes les valeurs passées en paramètre

Les bases du langage

Les tableaux:

on utilise un objet intégré du langage : Array.

3 possibilités pour l'initialiser :

- créer un tableau sans spécifier de taille (s'agrandira au fur et à mesure)
`var tableau = new Array();`
- créer un tableau en spécifiant sa taille i.e le nombre de cases(s'agrandira si besoin)
`var tableau = new Array(taille);`
- créer un tableau en indiquant le contenu de chacune des cases
`var tableau = new Array(valeur1,valeur2,valeur3,..);`
ou `var tableau = [valeur1,valeur2,valeur3];`

Une fois défini, vous pouvez mettre n'importe quel type de données dans les cases et vous pouvez même mélanger les types de données.

Ex

```
var tableau = new Array(true,10,"a","cd");
```

Les bases du langage

Pour obtenir la taille du tableau

nomDuTableau.length

ex: var tableau=new Array(1,2);

tableau.length vaut 2

Pour accéder à une case :

nomDuTableau[noCase] avec noCase variant de 0 à taille-1;

ex :

var tableau=new Array(1,2);

tableau[0] = 3;

var nb1=tableau[1];



Pour parcourir toutes les cases du tableau :

var taille=tableau.length;

```
for (var i=0;i<taille; i++ ) {  
    traitement sur tableau[i];  
}
```


Les bases du langage

Raccourci pour passer en revue tous les n° de case d'un tableau

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

Ex :

```
var nb;  
var chaine="";  
var tableau = new Array("a","b","c",);  
for (nb in tableau) {  
    chaine=chaine+tableau[nb];  
}
```

nb

2

chaine

abc

tableau

| | | |
|---|---|---|
| a | b | c |
|---|---|---|

Les bases du langage

La portée des variables

- Une variable déclarée dans une fonction est utilisable uniquement dans la fonction et inconnue hors de la fonction.
- Une variable déclarée en-dehors d'une fonction est accessible par toutes les fonctions. Elle existe lors du chargement de la page et ne disparaît que lors de son déchargement.



une variable non déclarée est systématiquement considérée comme étant une variable globale.

Les bases du langage

Exemple

```
var nb1;  
function f1 () {  
  var nb2=3;  
}
```

f1 a accès aux variables nb1 et nb2

```
function f2(nb3) {  
  var nb4;  
}
```

f2 a accès aux variables nb1,nb3 et nb4

Les objets prédéfinis de javascript

Il existe un certain nombre d'objets prédéfinis dans le langage disposant de propriétés et de méthodes facilitant la tâche du programmeur :

En voici quelques uns:

Screen : représente l'écran de l'internaute et nous permet d'avoir des informations sur les caractéristiques de l'écran de l'internaute

Date : permet de manipuler plus facilement les dates (notamment pour ajouter ou soustraire 2 dates)

RegExp : permet notamment de vérifier le format d'une saisie

Navigator : correspond au navigateur qui tourne

String : permet de manipuler les chaînes de caractère

Array : permet de manipuler les tableaux

Les objets prédéfinis de javascript

Image : permet de manipuler des images notamment pour les pré-chargés.

Math : contient les fonctions mathématiques (cos, sin, random)...

Global : contient des fonctions générales (parseInt, eval,...)

Vous retrouverez à la fin de ce cours , un récapitulatif des propriétés et méthodes de chacun de ces objets.

Les objets prédéfinis : Exemple Date

Cet objet permet de travailler avec les dates et les heures dans vos scripts.

Les objets Date ont une précision de l'ordre de la milliseconde et sont calculés à partir du 1er Janvier 1970.

Cet objet dispose d'un grand nombre de méthode permettant

- de lire
- d'écrire
- de manipuler des dates.

Les objets prédéfinis : Exemple Date

Pour créer un objet Date : 4 façons

- `var uneDate=new Date();` // initialise la variable uneDate avec la date et l'heure de //votre ordinateur (au moment de la création)
- `var uneDate=new Date(nombre);` // nombre est numérique et représente le nb de // millisecondes s'étant écoulé depuis le 1er Janvier 1970.

Ex `var date1= new Date(30000510255);`

date1 correspond au 14/12/1970 6h28mn30s

- `var uneDate= new Date(année,mois,jour [,heures [,minutes [,seconde [,millisecondes]]]]);` **ATTENTION: mois varie de 0 à 11 et non de 1 à 12**

Ex `date1=new Date(2008,3,3)` correspond au 03 Avril 2008

Rque : les variables écrites entre [] sont des paramètres optionnels

Les objets prédéfinis : Exemple Date

Et enfin

- `var uneDate=new Date(aaaa/mm/jj);` // initialise la variable uneDate avec la date passée en paramètre
- Ex : `var uneDate= new Date("2004/11/02");` correspond au 02 Novembre 2004



Il que soit la manière de créer la date, la date créée est toujours valide :

Ex : `uneDate=new Date("2004/12/53")` donne la date du 22 Janvier 2005

`uneDate = new Date(2004,25,3)` donne la date du 3 Février 2006

`uneDate = new Date (2004,12,1)` donne la date du 1 Janvier 2005

[voir date.htm](#)

Les objets prédéfinis : l'objet Date

```
var uneDate= new Date();
```

| Méthode | Description |
|--|--|
| uneDate.getFullYear() | retourne l'année stockée dans uneDate sous la forme d'un nombre sur 4 chiffres |
| uneDate.getMonth() | retourne un entier compris entre 0 (janvier) et 11 (décembre) |
| uneDate.getDate() | retourne un entier compris entre 1 et 31 correspondant au jour |
| uneDate.getDay() | retourne un entier correspondant au jour de la semaine : 0 pour Dimanche, 1 pour Lundi, 6 pour Samedi. |
| uneDate.getHours() | retourne un entier compris entre 0 et 23 correspondant aux heures |
| uneDate.getMinutes() | retourne un entier compris entre 0 et 59 correspondant aux minutes |
| uneDate.getSeconds() | retourne un entier compris entre 0 et 59 correspondant aux secondes |
| uneDate.setFullYear(an[,mois [,jour]]) | permet de mettre à jour la variable uneDate |
| uneDate.setMonth(mois[,jour]) | permet de mettre à jour le mois et éventuellement le jour de uneDate |
| uneDate.setDate(jour) | permet de mettre à jour le jour de la variable uneDate |
| idem pour les heures,minutes,secondes | |
| uneDate.toLocaleString() | retourne une chaîne de caractère contenant la date formatée selon la valeur par défaut des paramètres régionaux de la machine. |
| uneDate.toUTCString() | retourne une chaîne de caractère contenant la date formatée selon le format UTC (temps universel) |

Comment afficher une boîte de dialogue

Qu'est-ce qu'une boîte de dialogue?

Une boîte de dialogue est une fenêtre qui s'affiche au premier plan et qui permet

- d'avertir l'utilisateur
- de le confronter à un choix (oui, non)
- de lui poser une question et de récupérer sa réponse

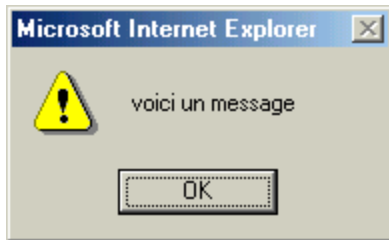
Ces 3 possibilités correspondent à 3 boîtes différentes en JavaScript.

Comment afficher une boîte de dialogue

Avertir l'utilisateur

La fonction: `window.alert(texte)` texte étant de type String

ex : `window.alert("voici un message");`



Pour fermer la boîte, l'utilisateur doit cliquer sur OK

Comment afficher une boîte de dialogue

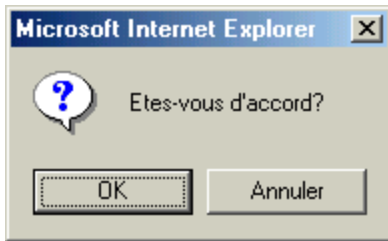
Poser une question à l'utilisateur

Ne peut répondre que par oui ou non

La fonction `window.confirm(texte)` avec texte de type String

Ex :

```
var choix;  
choix=window.confirm("Etes-vous d'accord ?");
```



Ok correspond à oui, Annuler correspond à non

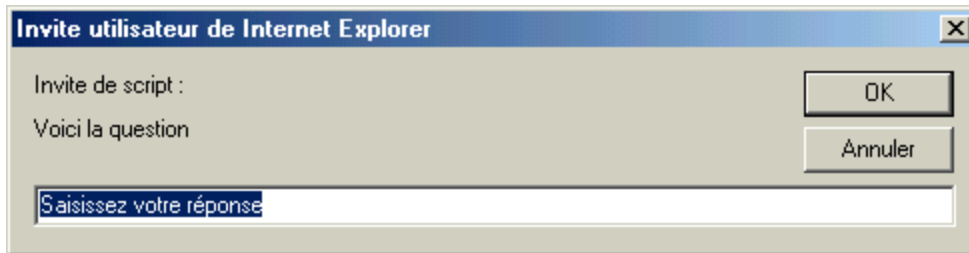
Lorsque l'utilisateur appuie sur "OK" la fonction a pour résultat la valeur *true*, *false* si l'utilisateur appuie sur *annuler*.

Comment afficher une boîte de dialogue

Avec une réponse libre

La fonction `window.prompt(question, preRenseignerReponse)` avec question et preRenseignerReponse de type String

Ex :
var reponse;
reponse=window.prompt ("Voici la question" , "Saisissez votre réponse");



La variable réponse contiendra ce qu'a saisi l'utilisateur.

Si l'utilisateur a cliqué sur `Annuler`, reponse aura pour valeur `null`

Comment ouvrir une pop-up

window.open ("url de la fenetre", "nom de la fenetre", "paramètres optionnels")

paramètres optionnels :

width=valeur largeur de la nouvelle fenêtre

height =valeur hauteur de la nouvelle fenêtre

left =valeur permet de positionner la fenêtre :abscisse du point supérieur gauche

top =valeur ordonné du coin supérieur gauche de la fenêtre

toolbar=yes/no spécifie si la fenêtre doit contenir la barre d'outils

location=yes/no spécifie si la fenêtre doit contenir la barre d'adresse

status=yes/no spécifie si la fenêtre doit contenir la barre d'état

menubar=yes/no spécifie si la fenêtre doit contenir la barre de menu

scrollbar =yes/no spécifie si la fenêtre doit contenir les barres de défilement

resizable=yes/no spécifie si la fenêtre peut être redimensionnée

fullscreen=yes/no spécifie si la fenêtre est affiché en plein écran

Comment ouvrir une pop-up

Exemple :

```
window.addEventListener("load",init,false);
```

```
function init(evt) {
```

```
    // la fonction ouvrirPopUp se déclenchera lorsque l'internaute cliquera sur le bouton  
    // ayant pour id btnOuvrir
```

```
        document.getElementById("btnOuvrir").addEventListener ("click",ouvrirPopUp,false);  
    }
```

```
function ouvrirPopUp() {
```

```
    // on ouvre dans une nouvelle fenêtre la page page2.htm
```

```
    var maPopUp=window.open("page2.htm","page 2",{width:300px,height:200px});  
}
```

Comment ouvrir une pop-up

La fenêtre principale et la pop up sont ensuite capable de dialoguer :

`window.opener` retourne la référence de la fenêtre ayant ouvert celle-ci.

voir exemple [creerPopup.htm](#)

Attention: Lorsqu'on clique la 1ere fois sur un des boutons , il se peut que la fenêtre ne s'affiche pas ou sans la couleur de fond. Cela est dû au fait que lorsqu'on donne le focus ou que l'on change la couleur, la fenêtre n'est pas encore finie de charger.

Comment mettre en place un timer

Il arrive régulièrement que l'on soit amené à exécuter une action de manière répétitive toutes les x millisecondes (par exemple mettre à jour l'heure) ou à effectuer une action au bout d'un certain temps (par exemple dans un quizz, on lui laisse 1 minute pour répondre puis on passe à la question suivante).

Dans ces 2 cas là, on va utiliser un timer.

Pour exécuter UNE fonction au bout de x millisecondes :

identifiant= window.setTimeout(nomFonction, x millisecondes)

L'identifiant est unique pour chaque timer et permet de l'identifier de manière unique

ex timer1=window.setTimeout(passerQuestion,60000);

Pour arrêter le timer avant l'exécution de la fonction

window.clearTimeout(identifiant);

ex : window.clearTimeout(timer1);

Comment vérifier un formulaire

La saisie d'un formulaire s'effectue via la création d'une balise form qui elle-même contient un certain nombre de champs de saisie ET un bouton submit.

```
<form action="script.php" method="post">
```

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

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

...

```
<input type="submit" value="enregistrer" />
```

```
</form>
```

Comment vérifier un formulaire

Les éléments d'un formulaire sont déjà nativement interactif (cette interactivité a été mise en place par le navigateur)

- Les champs de saisies réagissent aux frappes du clavier (le caractère tapé apparaît dans la zone)
- Les boutons radios ou les case à cocher réagissent au click (ou sélection via le clavier) une coche ou un rond noir apparaît dans l'élément sélectionné et l'élément dispose alors d'un attribut checked à true
- Les input de type submit ou image réagissent au click (ou idem sélection clavier). Il y a automatiquement diffusion d'un événement submit lorsqu'on clique sur ces boutons
- Le formulaire (balise form) réagit à l'événement submit . Automatiquement ce que vous avez indiqué dans l'attribut action (généralement un script serveur) est exécuté et les données saisies sont transmises.

Comment vérifier un formulaire

La données saisies doivent toujours être vérifiées avant d'être traitées. Cette vérification doit toujours avoir lieu sur le serveur. On va généralement la doubler d'une vérification en javascript :

- permet d'avertir immédiatement l'internaute des erreurs qu'il a commises
- évite qu'il perde sa saisie
- évite de trop surcharger le serveur :

Etape1 : Bien écrire votre formulaire html : balise form et input submit ou image obligatoires.

Etape 2 : l'objet **formulaire** doit écouter l'événement submit.

```
addEventListener(document.getElementById("form1"), "submit", verifier);
```

Rque : Vous ne **devez surtout pas** mettre en place la vérification du formulaire sur l'événement click du bouton submit.

Comment vérifier un formulaire

Etape 3 : Programmer la fonction verifier. Si la saisie contient des erreurs vous devez **obligatoirement arrêter l'interaction native** (l'exécution du script serveur et l'envoi des données) en appelant la methode preventDefault() de l'objet Event.

```
function verifier(objetEvent) {  
  if (document.getElementById("nom").value=="") {  
    alert("veuillez remplir le nom");  
    objetEvent.preventDefault();  
  }  
}
```

exemple : [verifierFormulaire.htm](#)

l'objet RegExp

On se sert de RegExp pour effectuer des recherches dans des chaînes de caractères ou pour remplacer les occurrences par d'autres. On s'en sert aussi principalement pour valider le format d'une saisie dans un formulaire. Par exemple, pour vérifier qu'une date est bien sous la forme jj/mm/aaaa, qu'un email est valide, qu'un numéro de téléphone est valide, etc...

Pour créer une instance de cet objet :

```
var exp=new RegExp("modèle",[option]) ;
```

[option](#) permet de préciser le type de recherche à effectuer :

- **g**: Recherche globale, c'est à dire ne s'arrête pas à la première occurrence trouvée, fait une recherche sur toute la chaîne.
- **i**: Ignore la casse.
- **m**: Fait une recherche sur plusieurs lignes.

[modèle](#) est composé d'une position(optionnelle) et d'un ensemble de classes de caractères

L'objet RegExp

Classe de caractère peut être construit à l'aide des caractéristiques suivantes

| | | |
|-------|--|---|
| [xyz] | caractères présents dans les crochets | [abc] correspond au caractère a ou b ou c |
| [^xy] | interdit les caractères présents dans les crochets | [^cp] correspond à un caractère différent de c et de p |
| \d | Correspond à un caractère représentant un chiffre | |
| \D | Correspond à un caractère ne représentant pas un chiffre | |
| \w | Correspond à un caractère représentant une lettre, un chiffre ou un souligné | |
| \W | tout ce qui n'est pas lettre, chiffre, souligné | |
| \s | espace, retour à la ligne, retour chariot, tabulation | |
| \S | tout ce qui n'est pas espace, retour à la ligne, retour chariot, tabulation | |
| * | le caractère peut apparaître 0 ou n fois | ex : m* : m peut être présent 0 ou n fois |
| + | le caractère doit apparaître au moins une fois | ex : m+ : m doit au moins être présent une fois |
| ? | le caractère doit apparaître 0 ou 1 fois | ex : m? : m doit être présent 0 ou 1 fois |
| {n} | le car. doit apparaître n fois | ex m{2} m doit être présent 2 fois |
| {n,} | il doit apparaître au moins n fois | ex: m {2,} m doit au minimum être présent 2 fois |
| {n,m} | au moins n fois et au plus m fois. | ex m {2,3} m doit être présent 2 fois minimum et 3 fois maximum |
| a b | a ou b | |

L'objet RegExp

Position

| | |
|----|-----------------------------------|
| ^ | correspondance en début de chaîne |
| \$ | correspondance en fin de chaîne |
| \b | correspondance en début de mot |
| \B | correspondance en fin de mot |

Méthode de RegExp

| Méthode | Description |
|--------------|--|
| exec(chaîne) | Exécute la recherche de l'expression régulière dans chaîne. Si aucune valeur n'est trouvée retourne null, sinon retourne un tableau contenant la valeur de la première occurrence trouvée. si on rappelle une deuxième fois exec retourne la deuxième, etc |
| test(chaîne) | Retourne un booléen indiquant si la chaîne respecte le modèle. retourne vrai si c'est le cas, faux sinon. |

Rque : les caractères spéciaux doivent être précédés de \\

- ♦ \\ correspond au caractère '\'
- ♦ \n correspond à un saut de ligne
- ♦ \f correspond à un saut de page
- ♦ \r correspond à un retour chariot
- ♦ \t correspond à une tabulation
- ♦ \. correspond à tout caractère

Exemple : [verifierAvecRegExp.htm](#)

Comment stocker une information :cookie

Javascript ne peut en aucun cas manipuler des fichiers présents sur le poste de l'utilisateur, que ce soit en lecture ou en écriture.

Il peut, malgré tout, être utile de conserver des informations sur le poste de l'utilisateur pendant une durée déterminée, et ce, même après la fermeture du navigateur.

Javascript met à notre disposition ce que l'on appelle des cookies pour effectuer cela.

Qu'est-ce qu'un cookie :

C'est une information qui est stockée sur le disque dur de l'utilisateur et qui donc persiste d'une session à une autre, y compris lorsque l'ordinateur est éteint.

Ils sont disponibles dans le fichier cookies.txt pour netscape et mozilla et dans le répertoire Cookies pour internet Explorer. Si vous en ouvrez un, vous trouverez une longue liste de site chacun accompagné d'une chaîne de texte généralement incompréhensible.

Comment stocker une information :cookie

- La taille d'un cookie ne peut dépasser 4 kilo-octet (environ 4000 caractères)
 - Un site web ne peut définir et lire que ses propres cookies.
- Vous n'avez droit qu'à 20 cookies par domaine (netscape) (adresse URL principale)
 - Tous les navigateurs n'acceptent pas les cookies
 - Tous les utilisateurs n'acceptent pas les cookies.

Comment stocker une information :cookie

Un cookie est formé de :


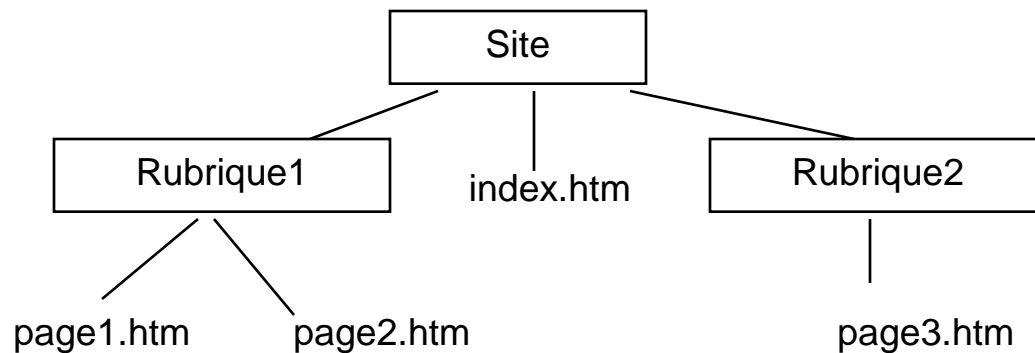
- **Un nom** : comme vous pouvez en avoir plusieurs, c'est ce qui les différencient. Même règle de nommage que les autres variables.
- **Une valeur** : Elle composée d'une chaîne de caractère dont les conventions sont celles d'une URL, c'est à dire que les caractères spéciaux sont remplacés leur correspondance Hexadécimale (espace = %20...). Pour cela on utilise escape et unescape (voir objet Global)
- **Une date d'expiration** : Les cookies peuvent exister uniquement le temps de la session (dans ce cas ils sont supprimés lorsque l'utilisateur quitte le navigateur) ou bien ils servent à stocker des informations devant être disponible hors session et on indique combien de temps le cookie doit exister sur le poste client.

En JavaScript, la date doit être au format **UTC** (pour cela utiliser la fonction `toUTCString()` de `Date`)

Si vous n'indiquez pas de date d'expiration, il n'est valable que le temps de la session.

Comment stocker une information :cookie

• **Un chemin** : Par défaut **le chemin d'un cookie est le répertoire courant de la page ayant créé le cookie**. Cela a une importance pour les autorisations d'accès à ce cookie. En effet seuls les pages appartenant à ce même répertoire ou bien à des sous-répertoires pourront consulter ou modifier le cookie.



page1.htm crée le cookie "cookie1". Seule y aura accès : page2.htm. index.htm et page3.htm n'auront ni le droit de lecture, ni le droit d'écriture. Si c'est index.htm qui crée ce cookie toutes les pages y auront accès

Si on modifie le chemin, c'est à partir de celui-ci que seront définies les autorisations d'accès. La valeur / correspond à la racine du site et dans ce cas toutes les pages du site ont accès au cookie

Comment stocker une information :cookie

Le domaine : Afin que vous ne puissiez pas utiliser les cookies d'autres domaines, celui-ci est rattaché explicitement au domaine l'ayant créé. Cela est fait par défaut par javascript.

secure : C'est un booléen qui permet de demander au cookie de n'être accessible que par le SSL (cryptage des données).

En javascript un cookie est donc construit selon le modèle suivant

nom=**valeur**;expires=**date**;path=**chemin**;domain=**domaine**;**[secure]** .

Pour créer un cookie , il faut écrire cette chaîne de caractère dans la propriété cookie de document.

```
window.document.cookie="nom= escape( valeur) ;expires=date;  
path=chemin;domain=domaine;[secure]";
```

Comment stocker une information :cookie

2eme exemple avec enregistrement de plusieurs valeurs

Comment modifier un cookie ?

Il suffit de recréer le cookie en lui donnant le même nom, cela efface la précédente valeur.

Si on veut ajouter des informations au cookie :

- il faut récupérer la valeur stockée
- y ajouter la nouvelle valeur
- recréer le cookie

Comment stocker une information:localStorage

Avec l'arrivée de html5, une autre manière de sauvegarder des données sur le poste client : l'utilisation d'un objet localStorage (ou sessionStorage suivant la durée de vie demandée) propriété de l'objet window.

Comme pour les cookies les données stockées sont obligatoirement de type string et on stocke des couples clé/valeur

localStorage : les données sont stockées sans limitation de durée

sessionStorage : les données sont stockées aussi longtemps que la fenêtre n'est pas fermée

Exemple de sauvegarde : `window.localStorage.setItem("nom","Smith");`

Exemple de récupération :

`var nomSauvegarde = window.localStorage.getItem("nom");`

Comment stocker une information:localStorage

API de localStorage ou sessionStorage :localStorage[cle]=valeur permet de stocker

localStorage.setItem(clé,valeur) ou localStorage[cle]=valeur permet de stocker une information

localStorage.getItem(clé) ou localStorage[cle] permet de lire une information stockée

localStorage.removeItem(clé) permet de supprimer une information

localStorage.length permet de connaître le nombre de valeurs stockées

localStorage.key(indice) permet de lire la ieme valeur stockée

localStorage.clear() permet de vider les valeurs stockées

Vous disposez aussi d'un évènement:

storage : cet événement est diffusé lorsqu'on dépose, supprime ou efface l'espace. L'object event associé dispose des propriétés suivantes:

- key : la clé
- oldValue :l'ancienne valeur
- newValue :la nouvelle valeur (ou null si il est supprimé)
- url : la page a l'origine du changement

Comment stocker une information:localStorage

| | cookie | localStorage | sessionStorage |
|---------------------------|---|--|----------------------------|
| Type des données stockées | String | String | String |
| Autorisation d'accès | Celles indiquées via le domaine et le chemin. Un site peut donc laisser des informations à destination d'un autre domaine | Le domaine ayant déposé | Le domaine ayant déposé |
| Navigateur y ayant accès | Celui ayant déposé | Celui ayant déposé | Celui ayant déposé |
| Restriction de stockage | 4 KO – 20 cookies par domaine | 5 MB par défaut. Eventuellement certains navigateurs permettent à l'utilisateur de le gérer | Idem localStorage |
| Durée de vie | Paramétrable par l'internaute: date d'expiration du cookie, session ou cookie non autorisé | Aussi longtemps qu'il n'est pas détruit | Lorsqu'on ferme la fenêtre |
| Compatibilité | Tous navigateurs | Tous navigateurs récents (IE8, firefox 3.5, Chrome 4.0, opera 10.5) | Idem localStorage |
| Suppression | Suppression facile pour les internautes | Suppression possible mais pas simple à trouver à l'heure actuelle | Idem |
| Transmission serveur | Le contenu de tous les cookies du domaine est envoyé automatiquement lors de toute connexion au serveur (http) | Jamais envoyé | Idem |

L'objet Array

Comme nous l'avons vu précédemment, cet objet permet de manipuler des variables de type tableau.

```
var unTableau=new Array()
```

| Méthode ou propriété | Description |
|---|---|
| <code>unTableau.concat(tab1, tab2[,tab3, ...])</code> | permet de concaténer plusieurs tableaux i.e crée un tableau à partir des différents tableaux passés en paramètre |
| <code>unTableau.join(separateur)</code> | renvoie une chaîne de caractère constituée de la concaténation de tous les éléments du tableau et séparé par le caractère que vous avez indiqué en tant que séparateur Voir 1 Tableau.htm |
| <code>unTableau.reverse()</code> | inverse l'ordre des éléments du tableau |
| <code>unTableau.slice(indicedebut , indicefin)</code> | retourne un tableau constitué de la plage d'éléments compris entre indexedebut et indicefin |
| <code>unTableau.sort()</code> | trie les éléments du tableau selon l'ordre ASCII Voir 1 tableauTri.htm |

L'objet Math

C'est l'objet qui contient un certain nombre de fonctions mathématiques. il ne peut pas s'instancier (var math=new Math() est impossible) mais le moteur de javascript en crée automatiquement une instance lors de son initialisation.

| Méthode ou propriété | Description |
|--|---|
| Math.abs(nombre) | valeur absolue |
| Math.acos (nombre),Math.asin(nombre),Math.atan(nombre) | arc cosinus et arc sinus , arc tangente |
| Math.ceil(nombre) | entier supérieur ou égal à nombre |
| Math.round(nombre) | arrondit à l'entier le plus proche (inférieur si partie décimale de nombre <0.5, supérieur sinon) |
| Math. cos (nombre),Math. sin(nombre),Math. tan(nombre) | cosinus , sinus , tangente |
| Math.floor(nombre) | entier inférieur ou égal à nombre |
| Math.log(nombre) | logarithme naturel |
| Math.max(valeur1,valeur2) Math.min(valeur1,valeur2) | calcul maximum et minimum de 2 valeurs |
| Math.pow(nombre1,nombre2) | calcul nombre1 puissance nombre2 |
| Math.random() | retourne un nombre compris entre 0 et 1 |
| Math.sqrt(nombre) | calcule la racine carrée |

L'objet String

JavaScript crée automatiquement des objets String lorsque vous déclarez des variables avec de valeurs de type chaîne de caractère . Pour connaître la taille d'une chaîne de caractère vous avez la propriété **length** (maChaine.length)

| Méthode | Description |
|---|--|
| <code>maChaine.anchor("nom")</code> | Entoure la chaine avec les balises d'ancrage. <code> maChaine </code> |
| <code>maChaine.charAt(indice)</code> | retourne le caractère présent dans la chaîne à l'indice indiqué. 1er caractère correspond à l'indice 0. |
| <code>maChaine.charCodeAt(indice)</code> | ramène le code unicode du caractère présent à l'indice indiqué |
| <code>maChaine.concat("chaine2")</code> | concatène 2 chaînes |
| <code>maChaine.indexOf("sous-chaine" [,indice départ])</code> | ramène l'indice de la première occurrence trouvée,-1 si elle n'existe pas. <code>indicedepart</code> permet d'indiquer à partir de quel indice s'effectue la recherche. |
| <code>maChaine.substring(pos1, pos2)</code> | retourne la sous-chaîne comprise entre pos1 et pos2 |
| <code>maChaine.toUpperCase()</code> | retourne la chaîne en majuscule |
| <code>maChaine.toLowerCase()</code> | retourne la chaîne en minuscule |
| <code>maChaine.split("caractère de séparation")</code> | La méthode split utilise un caractère ou un groupe de caractères pour diviser une chaîne en ensemble de sous-chaînes. Retourne un tableau contenant chacune des sous-chaînes. voir 1_split.htm |

L'objet Global

C'est un objet un peu particulier car on ne peut pas créer d'instance de cet objet, il est automatiquement créé par le moteur javascript. On accède à ces méthodes directement (sans préciser global)

| Méthode | Description |
|--------------------------------------|---|
| <code>parseInt(String,base)</code> | Transforme une chaîne en nombre entier dans la base indiquée. Si la chaîne ne contient pas de nombre retourne NaN |
| <code>parseFloat(String,base)</code> | Transforme une chaîne en nombre réel. Si la chaîne ne contient pas de nombre retourne NaN |
| <code>eval(String)</code> | évalue une expression et la calcule : Ex: <code>eval("2+3")</code> a pour résultat 5 ; <code>eval ("2" + "+" + "3")</code> a pour résultat 5 |
| <code>isNaN(valeur)</code> | Retourne true si valeur est égal à NaN(not a number), false sinon |
| <code>escape(String)</code> | Encode les chaînes pour les rendre lisibles sur tous les ordinateurs. Elle convertit les caractères ASCII en caractères UNICODE. Tous les espaces, la ponctuation, les caractères accentués, les caractères non ASCII sont remplacés par un pourcentage (%) suivi de la valeur hexadécimale du caractère. |
| <code>unescape(String)</code> | Décode les chaînes encodées à l'aide de la fonction <code>escape(String)</code> . |

L'objet window

Cet objet référence la fenêtre (celle-ci pouvant être une fenêtre classique ou une frame).

S'utilise pour accéder aux informations concernant l'état de la fenêtre, le navigateur utilisé pour l'afficher, le document html,....

Quelques propriétés (liste non exhaustive)

| propriété | Description |
|-----------------------------------|---|
| <code>window.closed</code> | retourne un booléen indiquant si la fenêtre est fermée ou non |
| <code>window.status</code> | permet d'accéder au message affiché dans la barre d'état |
| <code>window.defaultStatus</code> | valeur par défaut de la barre d'état |
| <code>window.event</code> | retourne une référence vers l'objet Event qui permet d'accéder aux paramètres de l'évènement qui représente l'objet HTML |
| <code>window.frames</code> | retourne un HTMLCollections des frames correspondant aux frames créés à l'aide de la balise frame dans un <frameset> |
| <code>window.name</code> | définit ou retourne le nom de la fenêtre |
| <code>window.navigator</code> | retourne une référence vers l'objet Navigator qui permet de collecter des informations sur le navigateur de l'utilisateur |
| <code>window.screen</code> | retourne une référence vers l'objet Screen qui permet de collecter des informations sur l'écran de l'utilisateur |

L'objet window

Quelques méthodes (liste non exhaustive)

| Méthodes | Description |
|---|--|
| <code>window.alert("message")</code> | affiche une boîte de message |
| <code>window.blur()</code> | fait perdre le focus à la fenêtre et déclenche son événement onblur |
| <code>window.focus()</code> | donne le focus à la fenêtre et déclenche son événement onfocus |
| <code>window.setTimeout(fonction à exécuter, millisecondes)</code> | Indique une fonction à exécuter au bout de x millisecondes. Retourne un identifiant. |
| <code>window.clearTimeout(identifiant)</code> | détruit le décompte de temps créé par la méthode au-dessus, la fonction ne sera donc pas exécutée |
| <code>window.setInterval(fonction à exécuter, millisecondes)</code> | Indique une fonction à exécuter toutes les x millisecondes. Retourne un identifiant. Voir 2_majHeure.htm |
| <code>window.clearInterval(identifiant)</code> | détruit le décompte de temps créé par la méthode au-dessus, la fonction ne sera donc pas exécutée |
| <code>window.navigate(url)</code> | affiche le document spécifié par l'url dans la fenêtre courante |
| <code>window.close()</code> | ferme la fenêtre du navigateur (ne s'utilise que pour les fenêtres ouvertes par script) |
| <code>window.prompt("message", valeur par défaut)</code> | affiche une boîte de dialogue et retourne la réponse |
| <code>window.confirm("message")</code> | affiche une boîte de dialogue avec le bouton Ok et annuler |
| <code>window.print()</code> | permet d'imprimer une fenêtre |
| <code>window.resizeTo(largeur, hauteur)</code> | permet de spécifier la taille d'une fenêtre |

L'objet Navigator

Il permet de collecter des informations sur le navigateur utilisé

| propriété | Description |
|-----------------|--|
| appCodeName | Retourne le nom de code du navigateur |
| appName | retourne le nom du navigateur |
| appVersion | retourne le numéro de version |
| browserLanguage | retourne la langue du browser |
| cookieEnabled | retourne un booléen indiquant si l'utilisateur accepte les cookies |

L'objet Screen

Il permet de collecter des informations sur l'écran de l' utilisateur

| propriété | Description |
|-------------|---|
| height | Hauteur en pixel de l'écran |
| width | Largeur en pixel de l'écran |
| availHeight | Retourne la hauteur en pixel de l'espace écran disponible |
| availWidth | Retourne la largeur en pixel de l'espace écran disponible |

L'objet location

Il permet de collecter des informations sur l'URL de la page (window.location)

| propriété | Description |
|-----------|---------------------------------------|
| hash | Nom de l'ancre à l'intérieur de l'URL |
| host | Nom de domaine à l'intérieur de l'url |
| href | URL/ lien à une URL |
| pathname | Nom du chemin à l'intérieur de l'url |

| méthodes | Description |
|-----------|--|
| reload() | Permet de recharger la page actuelle (idem à actualiser) |
| replace() | Charge une autre adresse URI sur l'élément actuel dans la liste des pages visitées (historique). L'élément actuel n'apparaît plus alors dans les pages visitées. |