

JavaScript

M4103C - Programmation Web – client riche

2ème année - S4, cours - 1/4
2017-2018

Marcel.Bosc@iutv.univ-paris13.fr

Table des matières

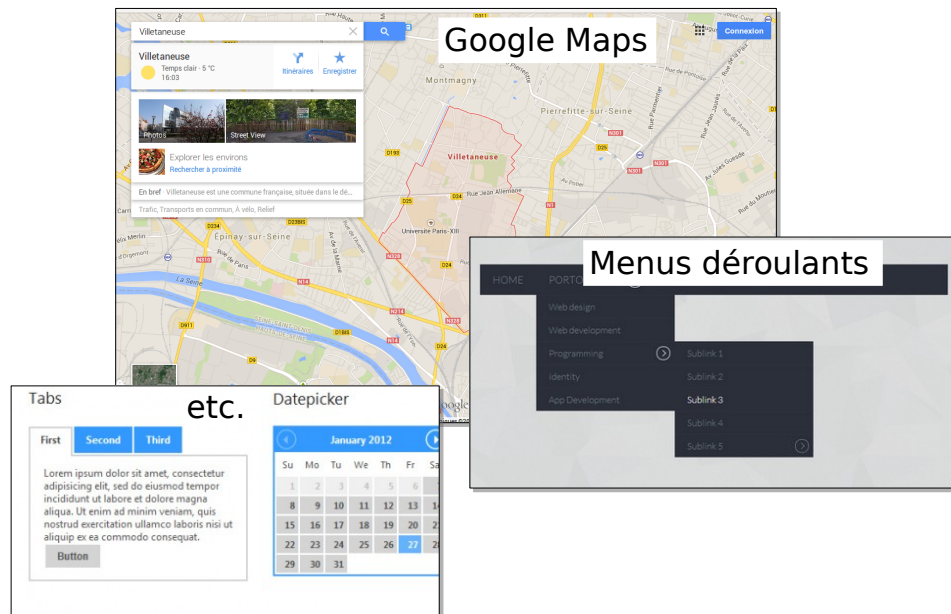
- À quoi ça sert ?
- Présentation
- Exemple 1
- Exemple 2

1ère partie

À quoi ça sert ?

Exemples

Js partout !

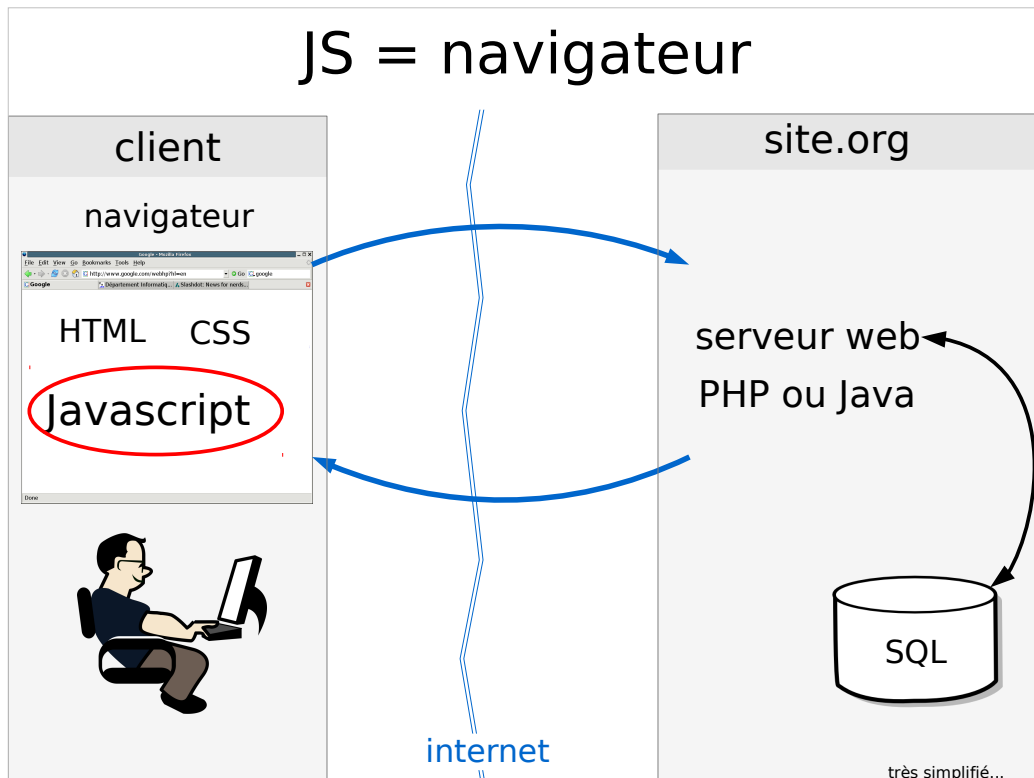


Aujourd'hui le JavaScript est utilisé sur presque toutes les pages du web.

Il permet de gérer les interactions, des plus simples (formulaires, onglets) aux plus complexes (Google Maps, Gmail, ...)

D'autres technologies client (comme le Flash) sont progressivement remplacées par le JavaScript.

[lp519]



Le JavaScript est un langage de programmation qui peut s'exécuter directement dans le navigateur (client).

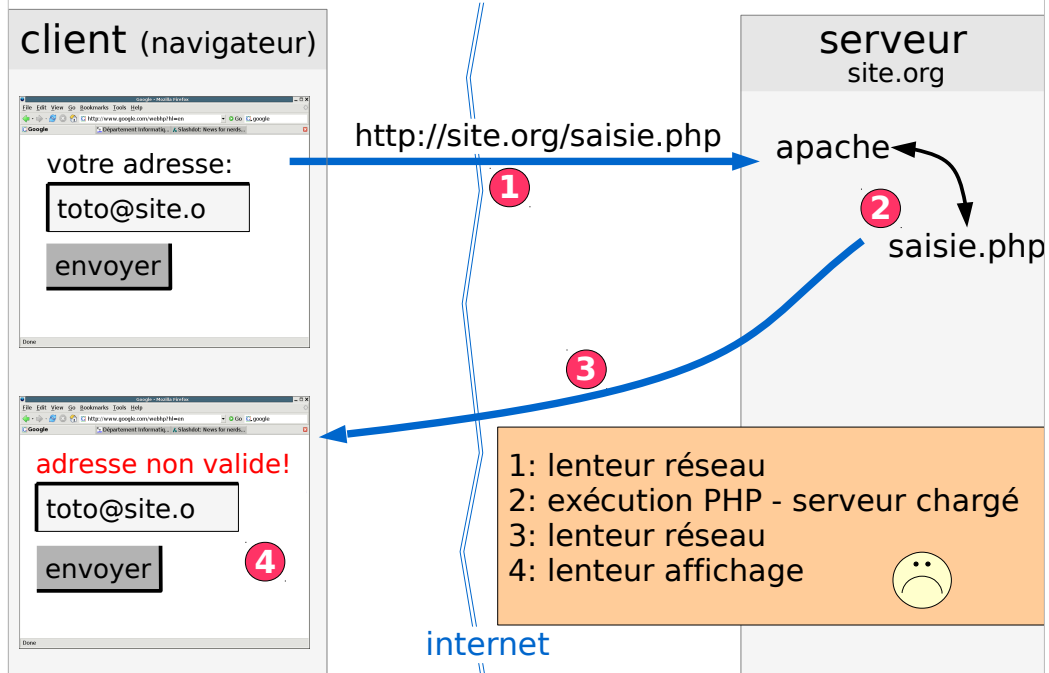
Le HTML et le CSS sont aussi gérés dans le navigateur (client).

Le PHP, Java (Servlets) et SQL sont utilisés coté serveur.

Il est très important de distinguer ce qui se passe sur le client (navigateur) de ce qui se passe sur le serveur. Plus tard, on verra des allers-retours complexes entre les deux.

[lp523]

Serveur = lent



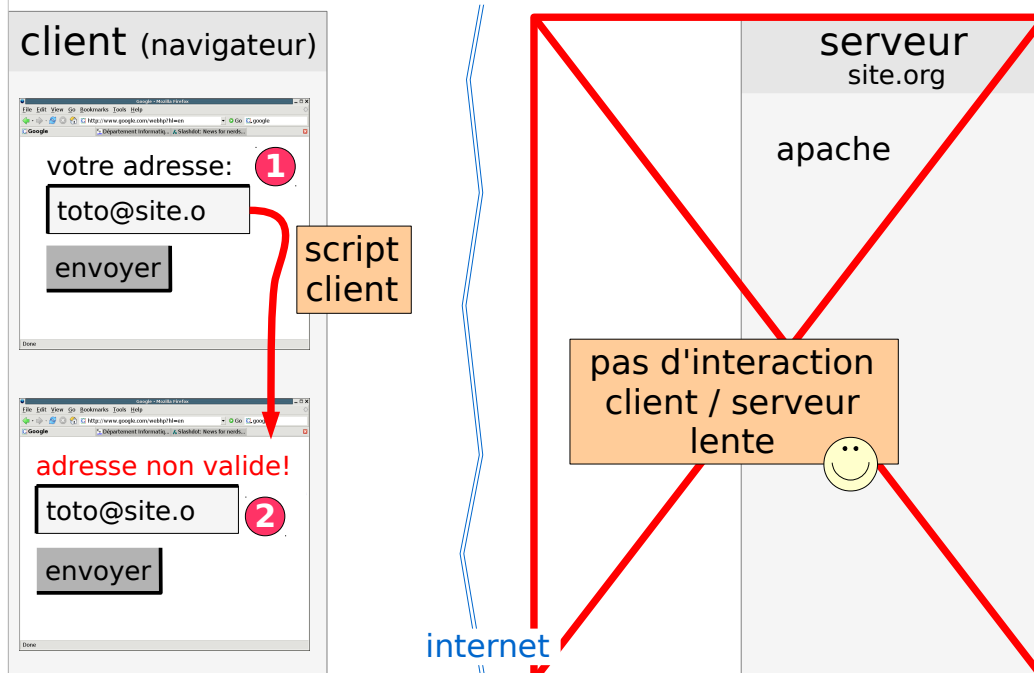
Les interactions client / serveur passent par internet et sont souvent lentes (300ms- 1500ms).

Exemple classique: la validation d'un formulaire.

L'utilisateur tape, appuie sur « envoyer » et doit attendre 1 à 2 secondes avant qu'une nouvelle page s'affiche lui indiquant une faute de frappe.

[lp524]

JavaScript = rapide



Le JavaScript s'exécute directement dans le navigateur.

Il peut modifier l'affichage très rapidement (quelques ms) sans passer par le réseau et sans recharger la page.

Exemple de formulaire:


Avec JavaScript l'erreur peut être affichée immédiatement, après avoir appuyé sur « envoyer », ou même au fur et à mesure que l'utilisateur tape.

[lp525]

Utile et sympa !

Stages :
presque tous Web => JS

Emploi :
très demandé

Sympa ! 

2ème partie

Présentation

Exemple

```
<html>
  <head>
    <title>exemple</title>
    <script src="jquery.js"></script>
    <script src="exemple.js"></script>
  </head>
  <body>
    <h1>Ceci est un titre</h1>
    <p>bla bla bla</p>
  </body>
</html>
```

exemple.html

```
$(document).ready(function()
{
  $('h1').click(function()
  {
    $('p').text('Bonjour!');
    $('p').css('color','red');
  });
});
```

exemple.js

Ceci est un

bla bla bla

Ceci est un

Bonjour!

Le javascript est inclus à partir du HTML avec la balise `<script>`.

Ce programme :

- Attend la fin du chargement du document :

```
$(document).ready(...)
```

- Lorsque l'utilisateur clique sur le titre h1 :

```
$('h1').click(...)
```

- Le texte du paragraphe devient "Bonjour!":

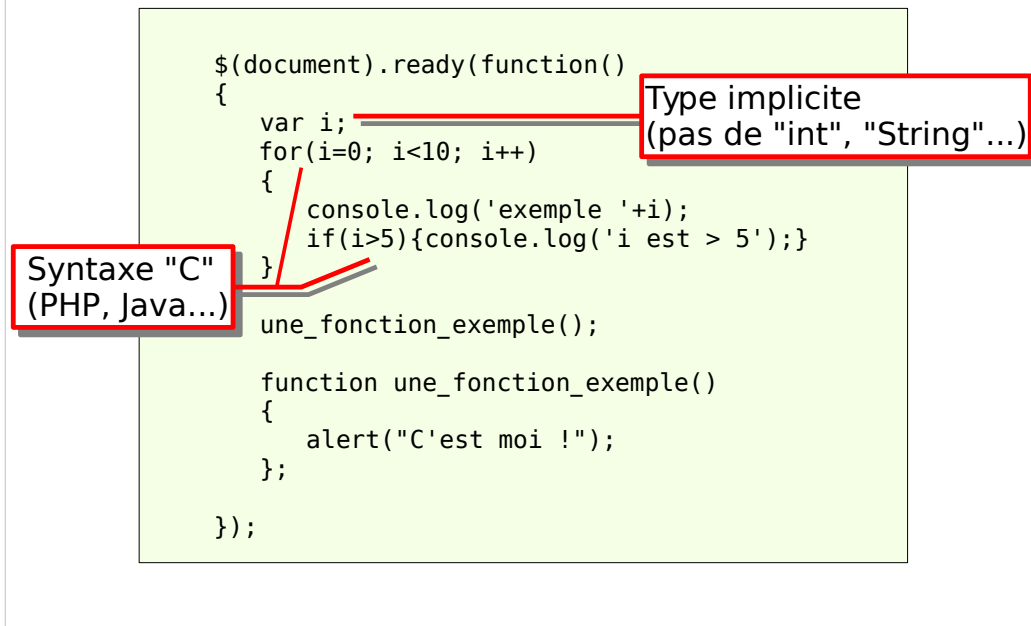
```
$('p').text('Bonjour');
```

- La couleur de ce paragraphe devient rouge

```
$('p').css('color','red');
```

[lp527]

Syntaxe



Le JavaScript est un langage de programmation ayant une « Syntaxe C ». Il ressemble au PHP et au Java.

Il n'y a pas de lien entre JavaScript et Java.

Comme en PHP, le type des variables est géré automatiquement: vous n'avez pas besoin de dire qu'une variable est un "int" ou un "String".

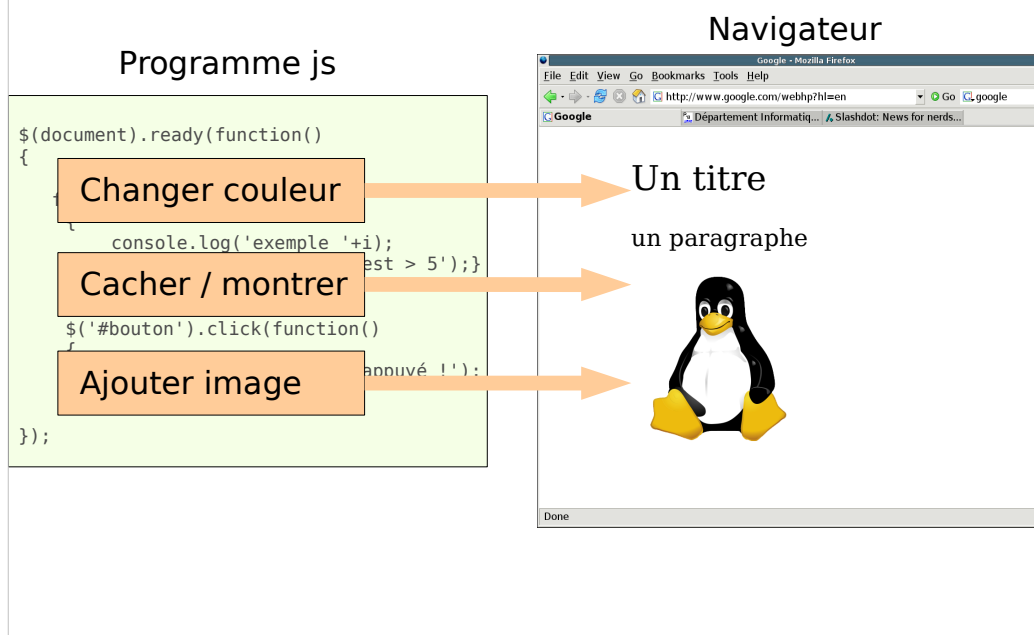
Comme en PHP et Java, la mémoire est gérée automatiquement. Vous n'avez pas besoin de désallouer (malloc=>free en C).

Guillemets simples et doubles comme en PHP.

Déclaration de fonction similaire au PHP.

[lp528]

Changer l'affichage



On va utiliser JavaScript surtout pour interagir avec l'affichage dans le navigateur.

On veut, par exemple, dire au navigateur :
"Affiche ce paragraphe en rouge", "Cache ce titre", "Ajoute une image après ce paragraphe".

On doit donc apprendre deux choses différentes:

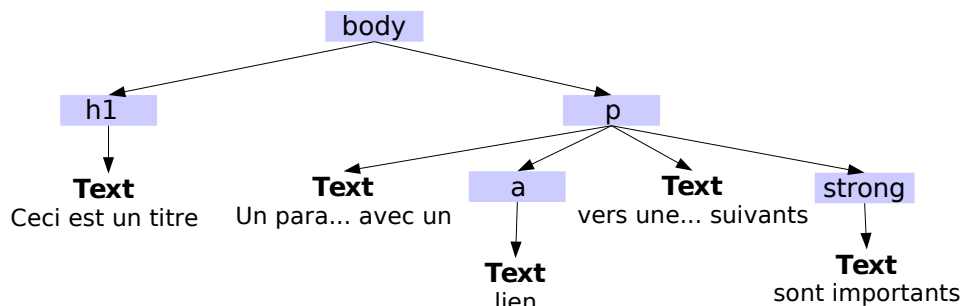
- 1- le langage de programmation JavaScript
- 2 - comment communiquer avec le navigateur

On va donc passer beaucoup de temps à comprendre comment interagir avec le navigateur.

[lp529]

HTML ↔ arbre

```
<body>
  <h1>Ceci est un titre</h1>
  <p>
    Un paragraphe de texte avec un
    <a href="page2.html">lien</a> vers une autre
    page. Les mots suivants
    <strong>sont importants</strong>
  </p>
</body>
```



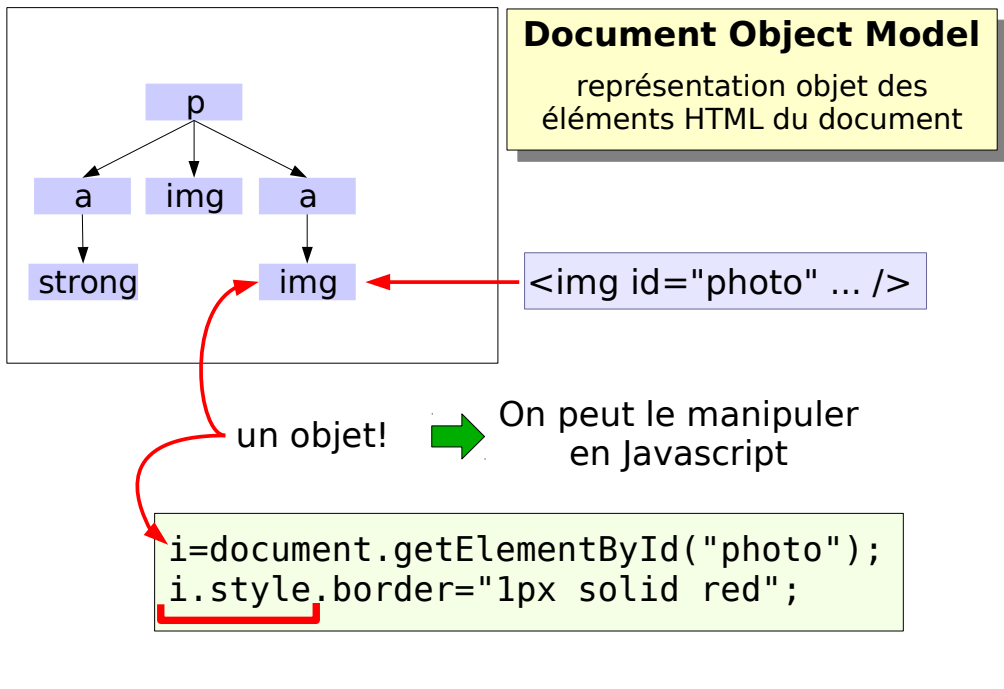
Le HTML peut-être vu comme un arbre.

Chaque noeud de l'arbre correspond à une balise.
L'ordre des balises est important.

Le texte contenu dans chaque balise est représenté par un noeud « Text ». En général, on ne montrera pas sur les schémas les noeuds « Text ».

C'est très important d'avoir toujours en tête la correspondance HTML <=> arbre.

DOM



Le navigateur reçoit et analyse le HTML. Il crée des objets (au sens POO) correspondant à chaque balise, et organisés dans un arbre.

Dans cet exemple « i » est l'objet DOM correspondant à une image. « i » a de nombreuses propriétés, dont certaines correspondent aux attributs (id, style, src...):

i.id => « photo »

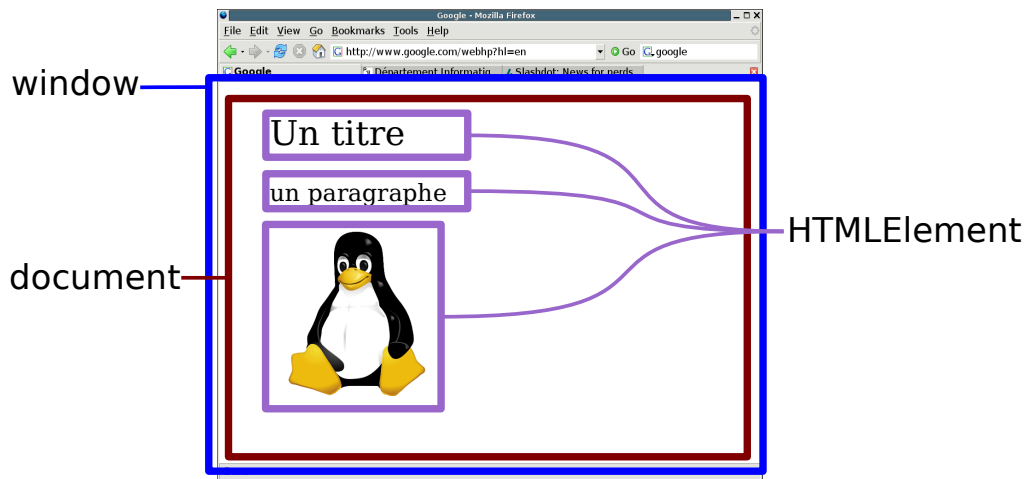
i.style => CSS associé

i.src => url de l'image

...

on peut lire et parfois modifier ces propriétés
[lp531]

Principaux objets DOM



Voici 3 types d'objets DOM importants:

- **window** : la fenêtre d'un document. S'il y a plusieurs onglets, chaque onglet a son window.
 - **document** : à l'intérieur du window, contient l'arbre DOM issu du HTML.
 - **HTMLElement** : la plupart des noeuds de l'arbre que nous manipulerons sont de type HTMLElement
- [lp532]



Facilite l'accès au DOM

DOM brut : ☹️

```
document.getElementById('photo').style.display="none";
```

jQuery : 😊 « \$ » = jQuery

```
$('#photo').hide();
```

```
<img id="photo" ... />
```

```
<img id="photo" ... style="display: none;"/>
```

jQuery est une bibliothèque JavaScript qui simplifie beaucoup la programmation en JavaScript.

Elle est très utilisée.

Elle n'est pas indispensable, mais nous allons l'utiliser dès le début de ces cours.

Nous verrons aussi, plus tard, comment manipuler le DOM sans jQuery.

On accède à jQuery avec « \$ »
[lp533]

3ème partie

Exemple 1

Dans cette partie, on va analyser en détail un exemple simple.

A travers cet exemple on va découvrir les notions essentielles de la programmation JavaScript et jQuery.

Exemple

```
$(document).ready(function()  
{  
  $('h1').click(function()  
  {  
    $('p').text('Bonjour!');  
    $('p').css('color','red');  
  });  
});
```

Ceci est un ti

bla bla bla



Ceci est un ti

Bonjour!

C'est l'exemple vu au début de la partie précédente.

Quand l'utilisateur clique sur le titre "Ceci est un titre", le paragraphe en dessous "bla bla bla" change de texte ("Bonjour!") et de couleur (rouge).

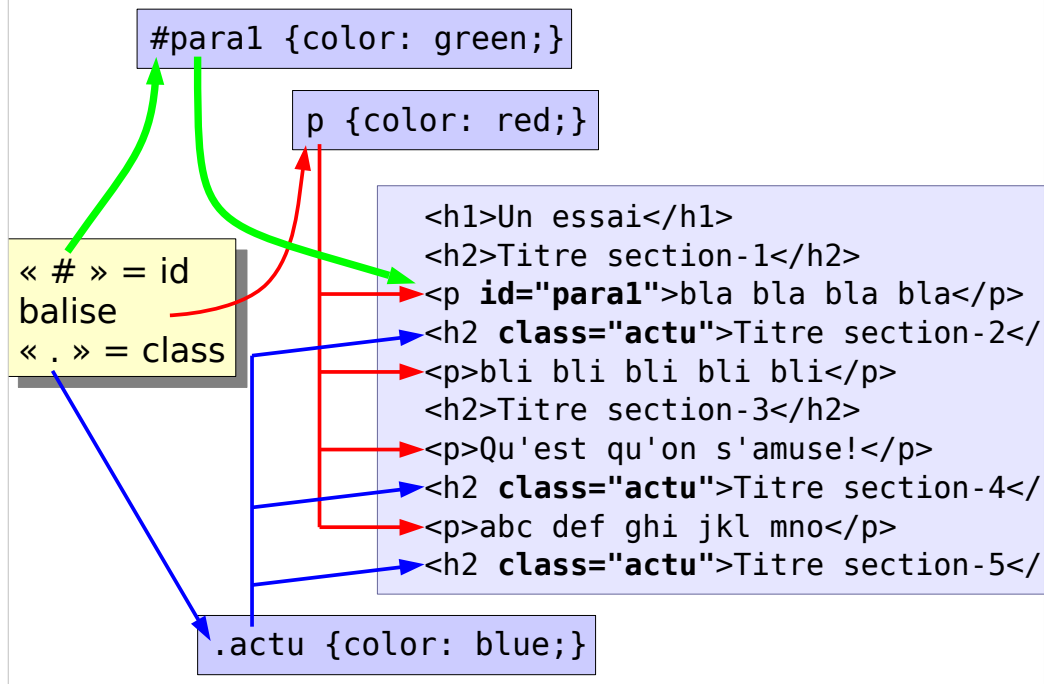
Commençons par la ligne

```
$('p').css('color','red');
```

C'est cette ligne qui change la couleur du paragraphe en rouge.

[lp553]

Rappel : sélecteurs CSS



En JavaScript, on utilise souvent les « sélecteurs ».

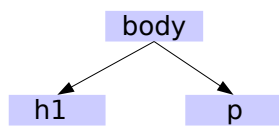
Vous avez appris à utiliser les sélecteurs en CSS. Les sélecteurs permettent de désigner depuis le CSS des éléments HTML (des éléments dans l'arbre DOM). On va aussi les utiliser en JavaScript pour désigner les éléments du DOM.

Le sélecteur le plus simple est le nom de la balise. Par exemple « p » désigne tous les paragraphes.

Le sélecteur « # » permet de désigner un élément (unique) ayant un id donné: `#para1` désigne l'élément qui a `id="para1"`.

Le sélecteur « . » permet de désigner les éléments ayant un « class » donné: `.actu` désigne tous les éléments ayant `class="actu"`

Liste jQuery



```
<body>
  <h1>Ceci est un titre</h1>
  <p>bla bla bla</p>
</body>
```

```
$('p').css('color', 'red');
```

`$('xyz')`

liste de tous les éléments
correspondant au sélecteur xyz

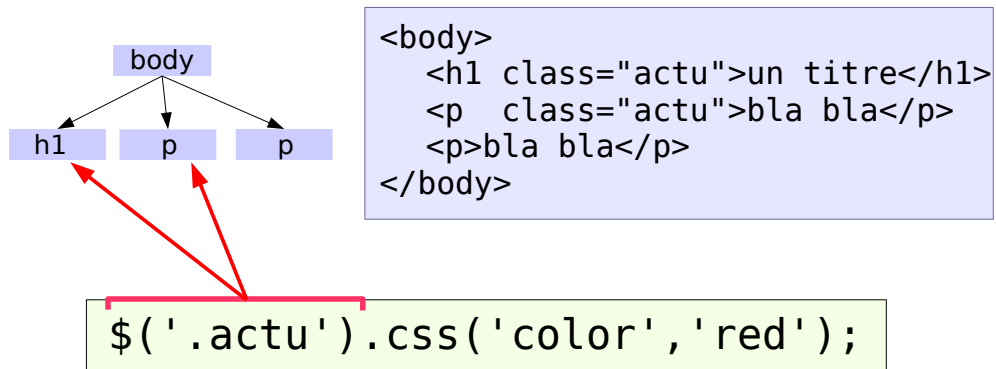
Voyons d'abord la première partie de la ligne:
`$('p')`

'p' est un sélecteur qui désigne tous les paragraphes. Ici, il n'y en a qu'un seul ("bla bla bla").

Le « \$ » correspond à jQuery.

D'une manière générale `$('xyz')` permet de créer une liste jQuery d'éléments correspondant au sélecteur "xyz".

Liste jQuery



Dans l'exemple précédent la liste jQuery `$('p')` ne contenait qu'un seul élément.

Cet exemple est différent. Il y a plus d'éléments et on utilise un sélecteur "class":
`$('.actu')`

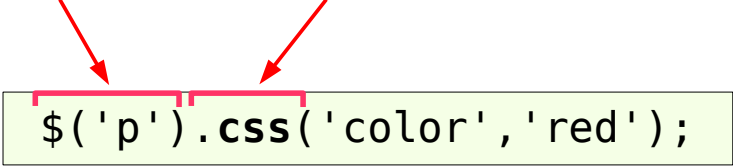
La liste générée contient deux éléments : le h1 et le premier paragraphe.

Cet exemple est simple (`.actu`). En pratique, le sélecteur peut être complexe et la liste peut contenir de très nombreux éléments.

Les listes jQuery sont un moyen très pratique pour faire des opérations sur les éléments DOM. On va les utiliser très souvent.

Fonction jQuery : .css()

Sur la liste appeler la fonction "css"



```
$( 'p' ).css( 'color' , 'red' );
```

".css()" change la propriété « style » des éléments :

<p>



<p style="color: red;">

La deuxième partie ".css('color','red')" est une fonction associée à la liste \$('p')

\$('p') est une liste jQuery ... qui est aussi un objet. .css() est une fonction associée à cet objet (en POO on dit aussi « méthode »).

Les listes jQuery fournissent de très nombreuses (>180) fonctions permettant de faire toutes sortes d'opérations. Ici on s'intéresse à .css()

Quand on écrit du HTML, on écrit en général le CSS dans un fichier séparé. Mais on peut aussi ajouter directement du CSS dans une balise avec l'attribut style:

```
<h1 style="font-size: 20px">exemple</h1>
```

La fonction .css permet de changer « style ».

[lp549]

Fonctions jQuery : class

```
$( 'p' ).addClass( 'actu' );
```

`<p>` → `<p class="actu">`

```
$( 'p' ).removeClass( 'actu' );
```

`<p class="actu">` → `<p>`

Mieux que .css() !
class + fichier CSS



toggleClass(...)

```
$( 'p' ).hasClass( 'actu' )
```

↙ true
↘ false

Rappel:

un élément peut avoir plusieurs valeurs dans class, séparées par des espaces:

`<p class="actu exemple autre-exemple">...</p>`

jQuery fournit les fonctions `.addClass()` et `.removeClass()` pour ajouter et enlever des noms dans "class".

`.hasClass()` permet de voir si un nom se trouve dans les valeurs de class.

Pour des affichages complexes, il est préférable d'utiliser "class" que style/.css(). En utilisant class on écrit uniquement le CSS dans des fichiers CSS. C'est plus propre / lisible.

[lp548]

Fonctions jQuery : .text() .html()

```
$('p').text('bonjour');
```

`<p>bla bla</p>`



`<p>bonjour</p>`

```
$('p').html('bonjour <a href= "... ">joe</a>');
```

`<p>bla bla</p>`



`<p>bonjour joe</p>`

Voici deux autres fonctions, très utilisées, s'appliquant sur une liste jQuery et permettant de modifier le contenu de tous les éléments de cette liste.

.text() permet de modifier le texte contenu dans un élément. Tout ce que contenait cet élément est préalablement effacé.

.html() permet de modifier le HTML contenu dans un élément. Tout ce que contenait cet élément est préalablement effacé.

Exemple

```
$(document).ready(function()  
{  
  $('h1').click(function()  
  {  
    $('p').text('Bonjour!');  
    $('p').css('color','red');  
  });  
});
```

Ceci est un ti

bla bla bla

Ceci est un ti

Bonjour!

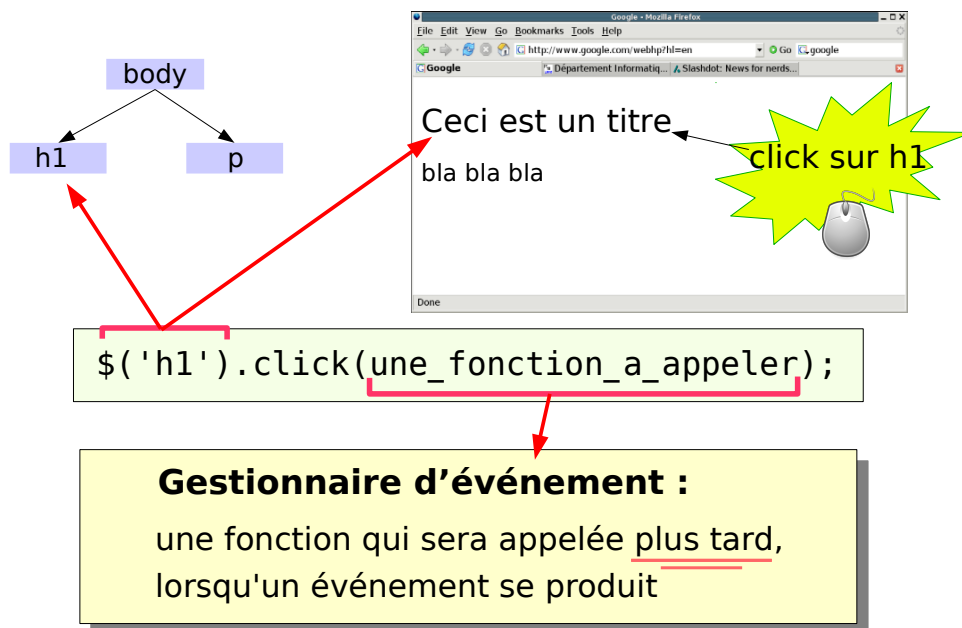
Passons maintenant à une autre ligne de l'exemple:

`$('h1').click(...)`

Rappelez-vous: dans cet exemple le paragraphe change de texte et de couleur quand l'utilisateur clique sur le titre.

[lp544]

Réagir à un événement



Cette ligne se divise en trois:

- \$('h1') : une liste jQuery contenant h1
- .click() : une fonction jQuery appelé sur la liste
- une_fonction_a_appeler : une fonction qu'il faudra appeler plus tard.

Important : remarquez que une_fonction_a_appeler n'est pas appelée toute de suite. Elle n'est appelée que plus tard, lorsque l'utilisateur clique avec la souris. On appelle cette fonction un « gestionnaire d'événement ».

En JS on a souvent besoin de réagir à des événements (comme un click).

Événements

click	click bouton souris
mousedown	bouton souris enfoncé
mouseover	souris entre sur un élément
mousemove	souris bouge sur un élément
keydown	touche enfoncée
keyup	touche relâchée
keypress	touche enfoncée et relâchée
ready / load	élément a fini de charger
change	élément formulaire modifié
submit	formulaire envoyé

...

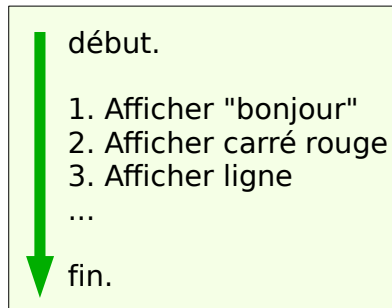


Le navigateur génère de nombreux événements, souvent à la suite d'une action de l'utilisateur.

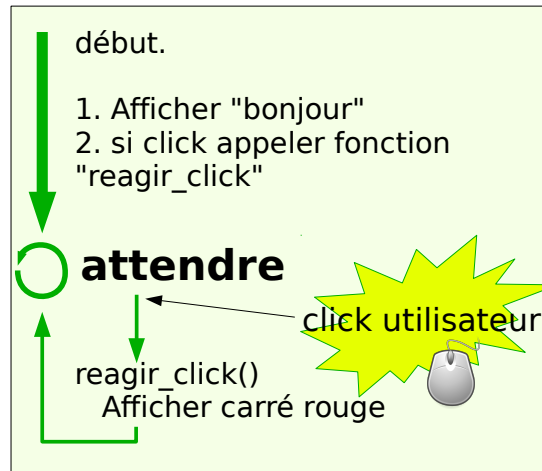
[lp542]

Programmation événementielle

Séquentielle « classique »



Événementielle



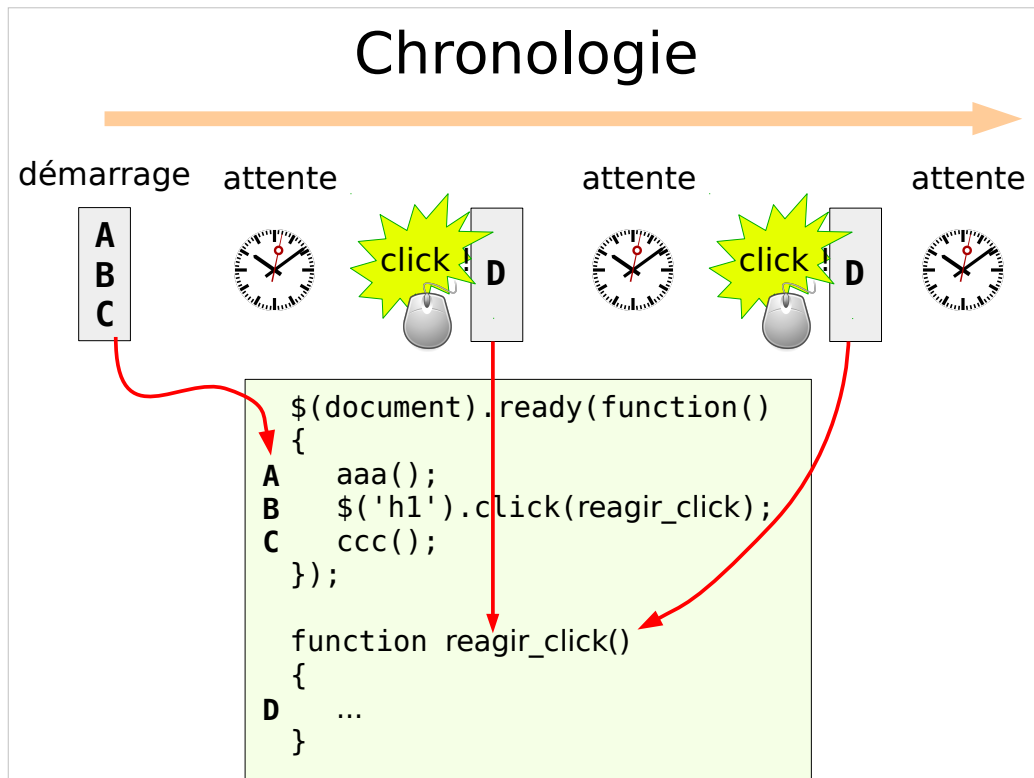
La programmation événementielle n'est pas facile à comprendre quand on a l'habitude de la programmation séquentielle.

En programmation événementielle, on n'a pas la main sur le déroulement du programme. On déclare quelles actions doivent être faites pour certains événements puis on rend la main.

Le système passe l'essentiel de son temps à attendre. Lorsqu'un événement arrive, il appelle une de nos fonctions, puis on rend à nouveau la main.

Cette manière de faire n'est pas toujours intuitive.

[lp541]



Reprenons sur un exemple concret.

Les lignes A,B,C sont exécutées, puis le programme se met en attente.

Ensuite, chaque fois que l'utilisateur clique sur un h1, la ligne D est exécutée.

L'erreur que font beaucoup de débutants, est de penser que l'ordre d'exécution est ABDC.

C'est une confusion sur ce que veut dire la ligne B. Celle-ci indique "Associer la fonction `reagir_click` à l'événement `click` sur les `h1`". La fonction `reagir_click` n'est appelée que plus tard, chaque fois que l'utilisateur clique.

Event / this

```
$('#h1').click(une_fonction_a_appeler);  
...  
function une_fonction_a_appeler(event)  
{  
    if(event.which===1)  
    {  
        $(this).css('font-size','12px');  
    }  
}
```

this : élément DOM cible de l'événement

event : objet décrivant l'événement

Quand un événement survient, notre fonction est appelée. Elle reçoit deux informations importantes:

- "event" (en argument): un objet décrivant l'événement, avec des informations comme la position de la souris, le bouton de souris utilisé, la touche appuyée ... On peut omettre cet argument si on n'en a pas besoin.

- "this" : l'objet associé à cette fonction. C'est l'élément DOM sur lequel est survenu l'événement. Dans cet exemple c'est h1. Attention: "this" est un objet DOM. Pour l'utiliser plus facilement, on peut le transformer en liste jQuery : `$(this)`

[lp540]

Exemple

```
$(document).ready(function()  
{  
  $('h1').click(function()  
  {  
    $('p').text('Bonjour!');  
    $('p').css('color','red');  
  });  
});
```

Ceci est un titre

bla bla bla



Ceci est un titre

Bonjour!

Analysons maintenant « `function(){...}` »
Cette fonction est appelée quand l'utilisateur clique sur h1. Elle change le texte et la couleur du paragraphe.

[lp539]

Fonction anonyme

Approche « habituelle »

```
$('#h1').click(une_fonction_a_appeler);  
...  
function une_fonction_a_appeler()  
{  
    $('#p').css('color','red');  
}
```

lourd ...



Fonction anonyme

```
$('#h1').click(function()  
{  
    $('#p').css('color','red');  
});
```

pratique...

très utilisé !



Dans l'approche « habituelle » on déclare une fonction avec un nom ("une_fonction_a_appeler"). On fournit ce nom en argument à la fonction click.

Très souvent cette fonction ne va être utilisée qu'une seule fois, et elle est assez courte.

Le JS permet de créer une fonction à n'importe quel endroit, sans lui donner de nom (fonction anonyme). Ici, la fonction est créée au même endroit où elle est passée en argument à click.

Cette approche rend le code beaucoup plus lisible en évitant des renvois et des noms superflus. On verra plus tard, qu'elle permet aussi de simplifier le partage de variables entre la fonction appelante et la fonction appelée.

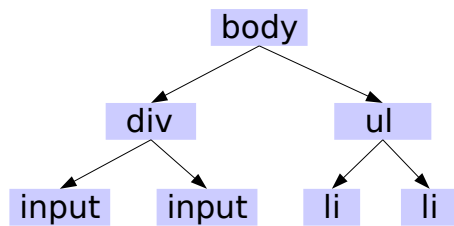
4ème partie

Exemple 2

On va finir par un deuxième exemple, où on verra comment ajouter des choses sur une page.

Exemple 2

```
<body>
  <div>
    <input id="saisie" type="text">
    <input id="ajouter" type="button" value="Ajouter"/>
  </div>
  <ul id="liste">
    <li>Tom</li>
    <li>Joe</li>
  </ul>
</body>
```



A visual representation of the HTML code. It shows a text input field, a button labeled 'Ajouter', and a bulleted list containing 'Tom' and 'Joe'.

Cet HTML simple affiche une zone texte, un bouton, et une liste.

[lp586]

exemple2.js

```
$(document).ready(function()  
{  
  $('#ajouter').click(function()  
  {  
    var ligne=$('<li></li>');  
    var texte=$('#saisie').val();  
    ligne.text(texte);  
    $('#liste').append(ligne);  
  });  
});
```

```
<div>  
  <input id="saisie"  
  <input id="ajouter"  
</div>  
<ul id="liste">  
  <li>Tom</li>  
  <li>Joe</li>  
</ul>
```

Wang

Ajouter

- Tom
- Joe
- Wang

On y associe un programme JS qui permet à l'utilisateur de taper du texte dans le champs texte puis d'appuyer sur le bouton « Ajouter ». Lorsque le bouton est appuyé, une ligne est ajoutée à la liste, contenant le texte tapé.

[lp585]

Rappels

```
$(document).ready(function()  
{  
    $('#ajouter').click(function()  
    {  
        var ligne=$('<li></li>');  
        var texte=$('#saisie').val();  
        ligne.text(texte);  
        $('#liste').append(ligne);  
    });  
});
```

attendre événement
« prêt » sur doc.

attendre événement
« click » sur élém.
id="ajouter"

changer texte
dans él. jQuery
« ligne »

Ces trois lignes correspondent à des notions déjà
vues précédemment
[lp584]

Fonction jQuery : .val()

```
$(document).ready(function()
{
    $('#ajouter').click(function()
    {
        var ligne=$('#<li></li>');
        var texte=$('#saisie').val();
        ligne.text(texte);
        $('#liste').append(ligne);
    });
});
```

<code><select></code>	<code><input type="text"/></code>	<code><textarea></code>
<div>Chocolat : Chocolat Fraise Vanille</div>	<div>essai</div>	<div>un exemple de texte</div>

En HTML, les formulaires sont très utilisés pour permettre à l'utilisateur de saisir des données.

La fonction jQuery .val() permet de lire les données entrées par l'utilisateur dans plusieurs types de champs:

`<select>`

`<input type="text"/>`

`<textarea>`

Comme toujours avec jQuery, il faut d'abord créer une liste en utilisant un sélecteur qui désigne le champ en question. Par exemple `$('#saisie').val()`

.val() permet aussi de modifier le champs:

`$('#saisie').val('nouveau texte')`

[lp583]

Créer un élément

```
$(document).ready(function()
{
    $('#ajouter').click(function()
    {
        var ligne=$('<li></li>');
        var texte=$('#saisie').val();
        ligne.text(texte);
        $('#liste').append(ligne);
    });
});
```

élément pas
encore dans l'arbre
DOM

`$('<xyz ..>...</xyz>')`

Crée des nouveaux éléments
à partir du code HTML.

Une opération courante en JS est de créer de nouveaux éléments pour les ajouter à une page.

On a vu précédemment la fonction jQuery .html() qui permet d'insérer du code HTML dans un élément existant. Cette approche n'est pas toujours pratique.

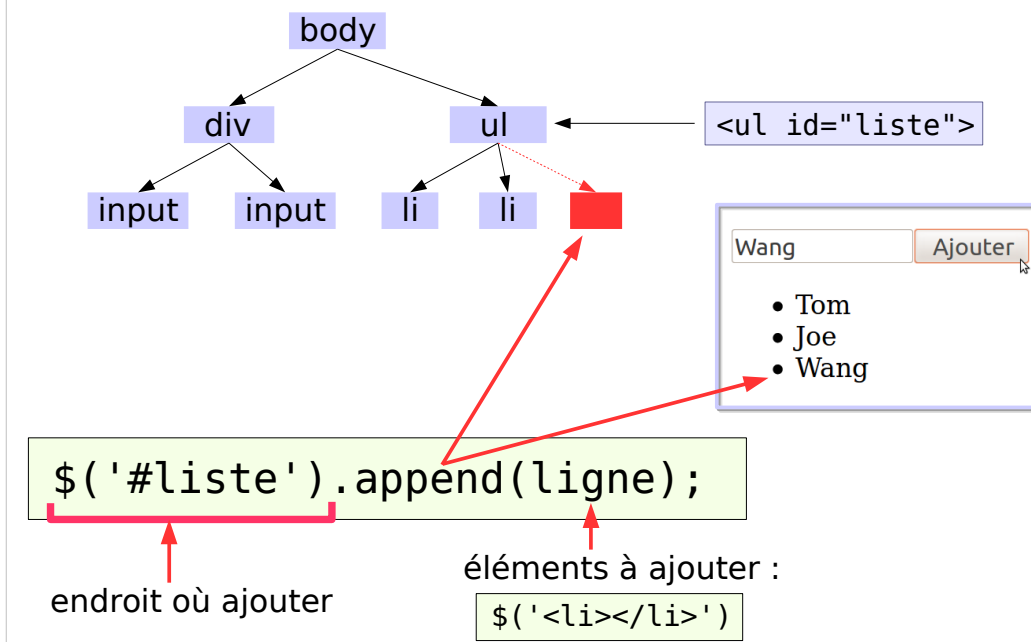
jQuery permet aussi de créer des éléments à partir de code HTML, sans les insérer dans l'arbre DOM.

`$('<p>exemple</p>')`

On peut manipuler ces éléments avec les fonctions jQuery. Ensuite on va les insérer dans l'arbre DOM.

[lp582]

Ajouter dans l'arbre DOM



jQuery fournit des fonctions pour insérer des éléments dans l'arbre DOM.

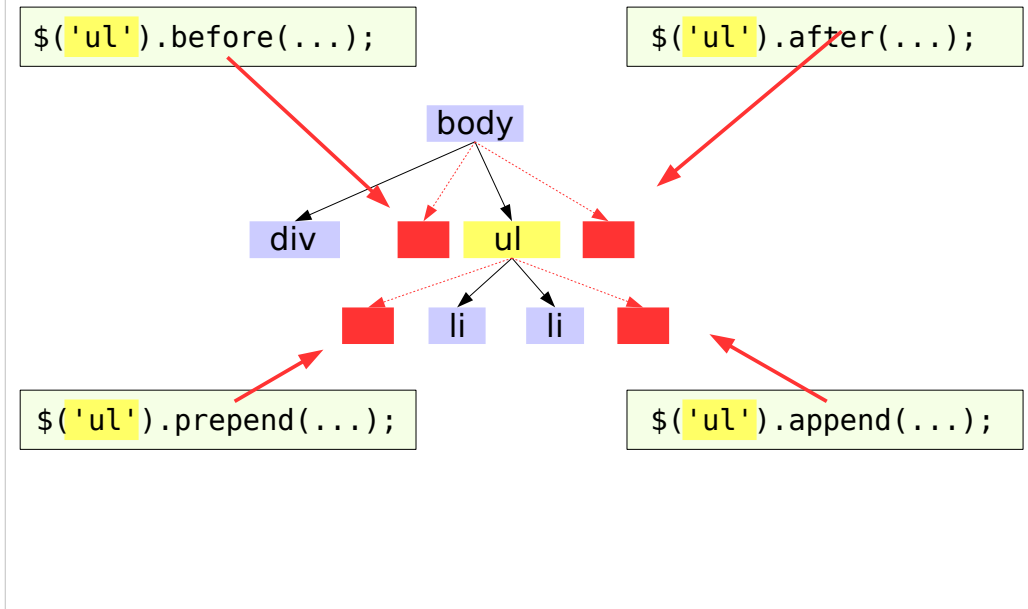
Il est important de comprendre à quel endroit exactement on veut insérer l'élément.

Ici on a créé un élément ``. On veut l'ajouter à la fin de la liste, c'est à dire dans `` et après les autres ``. On va donc utiliser la fonction `.append()` appliquée à la liste ``.

« append » veut dire « ajouter à la fin de ».

[lp581]

Où ajouter dans l'arbre DOM ?



jQuery fournit d'autres fonctions pour insérer des éléments dans l'arbre DOM, en fonction de l'endroit exact que l'on vise.

Dans ces 4 exemples on insère par rapport à `` : `$('ul').xyz(...)`

`before` : avant (frère précédent)

`after` : après (frère suivant)

`prepend` : avant le premier fils

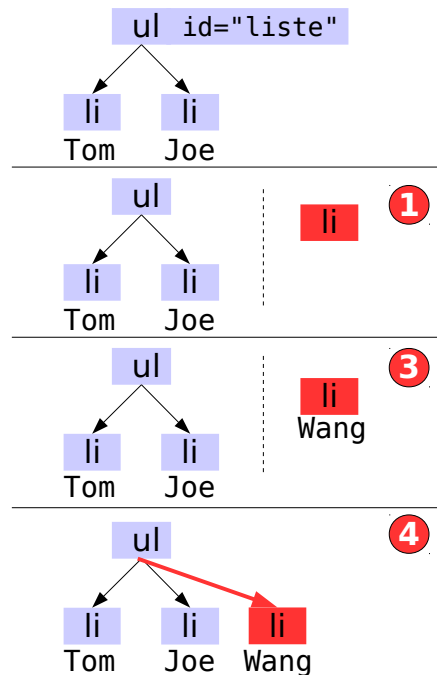
`append` : après le dernier fils

[lp580]

Récapitulatif

```
...  
$('#ajouter').click(function()  
{  
1  var ligne=$('#li></li>');  
2  var texte=$('#saisie').val();  
3  ligne.text(texte);  
4  $('#liste').append(ligne);  
});  
...
```

- Tom
- Joe
- Wang



Récapitulons:

- 1) On crée une liste jQuery contenant un `` vide. Ce `li` est créé par le navigateur, mais n'est pas encore inséré dans l'arbre du document.
- 2) On lit le texte que l'utilisateur a tapé
- 3) On écrit ce texte dans le `` créé.
- 4) On insère le `` dans l'arbre, à la fin du ``

[lp1761]

Ce document est distribué librement.

Sous licence GNU FDL :

<http://www.gnu.org/copyleft/fdl.html>

Les originaux sont disponibles au format LibreOffice

<http://www-info.iutv.univ-paris13.fr/~bosc>

Marcel.Bosc@iutv.univ-paris13.fr