

JavaScript

M4103C - Programmation Web – client riche

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

Marcel.Bosc@iutv.univ-paris13.fr

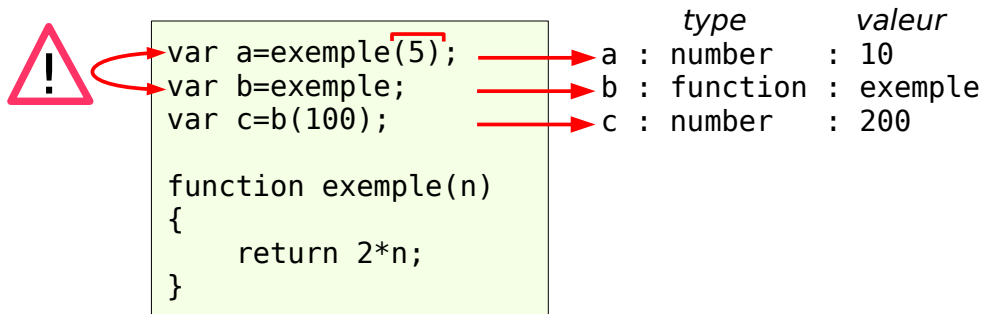
Table des matières

- fonctions JS
- `$(document).ready(...)`
- HTML & texte
- http
- HTML ajouté
- Listes jQuery

1ère partie

Fonction JS

Fonctions : appel / valeur



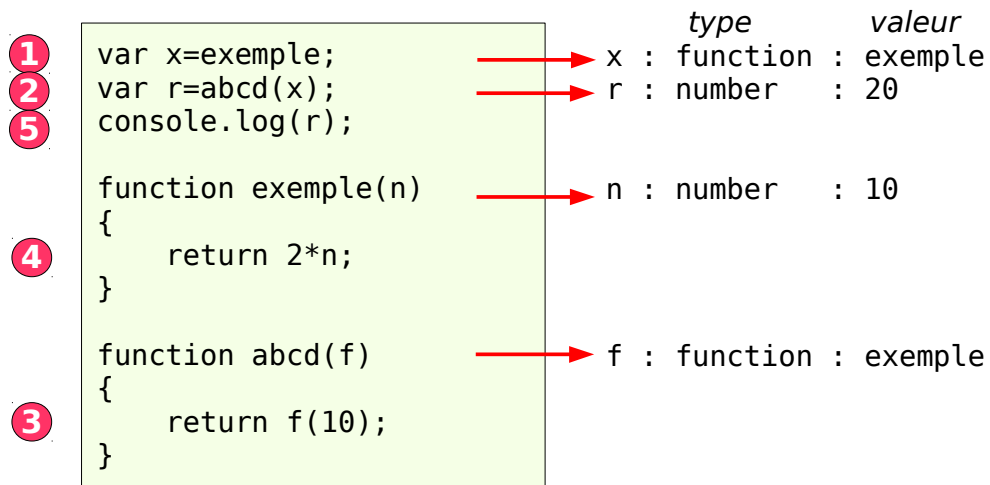
Dans les cours et TP précédents on a surtout utilisé des fonctions anonymes. Un petit rappel sur les fonctions "normales" (non-anonymes) peut-être utile. La déclaration d'une fonction "normale" se fait comme en PHP.

C'est très important de distinguer `exemple` qui est la fonction et `exemple()` qui est l'appel de la fonction.

En JS les fonctions sont un type de valeur comme les autres. On peut affecter ce type à une variable. On peut ensuite appeler la fonction.

[lp1409]

Fonctions : appel / valeur



Voici un autre exemple où on manipule des fonctions dans une variable.

[1] Ici « x » reçoit la valeur « exemple » qui est de type « function ». Remarquez bien que la fonction « exemple » n'est pas appelée.

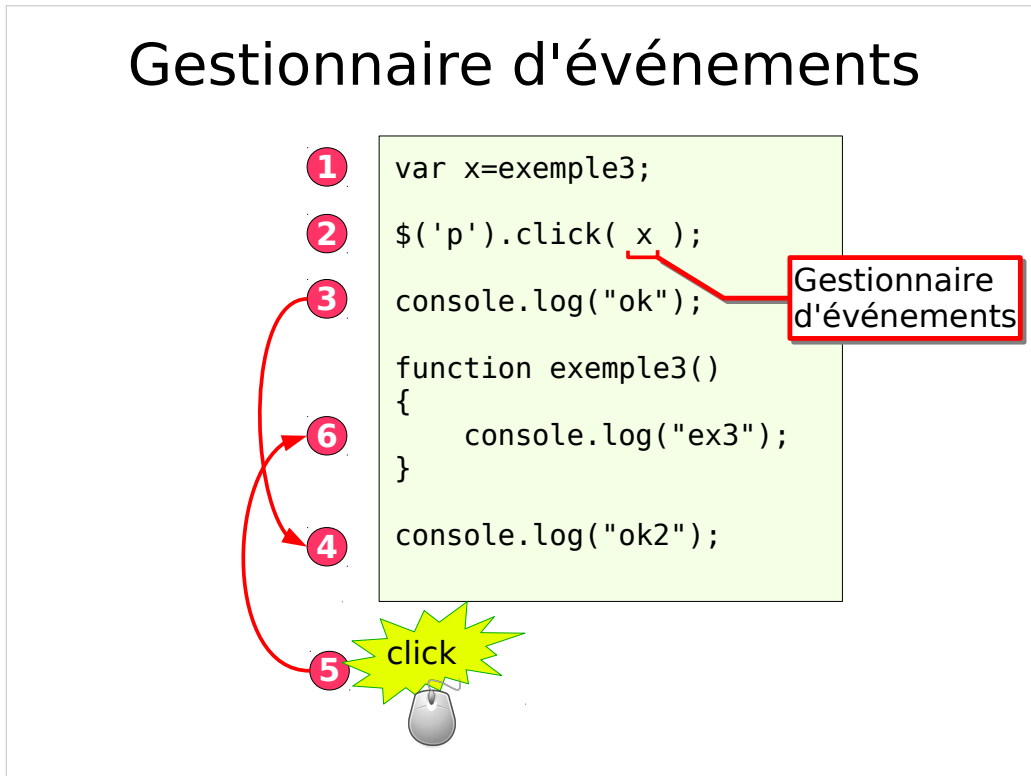
[2] « x » est ensuite passé en paramètre à la fonction « abcd ».

[3] Donc « f » dans « abcd » est bien la fonction « exemple ». Elle est appelée avec l'argument 10.

[4] « exemple » renvoie 20, donc abcd renvoie 20, donc « r » vaut 20.

C'est important de bien suivre l'ordre de l'exécution.

Gestionnaire d'événements



Les exemples précédents pouvaient paraître « artificiels ». Mais en pratique on manipule souvent les fonctions de cette manière.

[1] Ici « `exemple3` » est affecté à « `x` ». Remarquez bien que « `exemple3` » n'est pas appelée.

[2] Ensuite « `x` » est passé en argument à `.click()`. Remarquez bien que « `exemple3` » n'est pas non plus appelée.

[3,4] Le programme continue et rencontre la déclaration de « `exemple3` ». Remarquez bien que « `exemple3` » n'est pas non plus appelée.

Ensuite `console.log("ok2")` est appelé et le programme se termine.

[5] Après une attente l'utilisateur clique sur le paragraphe. Un événement est envoyé à celui-ci

[6] et la fonction « `x` » (donc `exemple3`) est appelée

Fonctions anonymes

```
var b=exemple;  
var c=b(100);  
  
function exemple(n)  
{  
    return 2*n;  
}
```



```
var b=function(n) 1  
{  
    return 2*n; 3  
};  
var c=b(100); 2
```

Depuis le début de ce cours on a souvent utilisé des fonctions anonymes.

C'est très pratique pour la programmation événementielle.

Ici on voit un exemple de fonction anonyme dans cadre non-événementiel.

On voit aussi les ressemblances avec la fonction non-anonyme « exemple ».

Remarquez bien que la fonction anonyme n'est pas appelée à la ligne [1]

[lp1406]

Gestionnaire d'événements

```
$('#p').click(exemple3);  
function exemple3()  
{  
    console.log("ex3");  
}
```



```
$('#p').click(function()  
{  
    console.log("ex3");  
});
```

Un exemple pour montrer qu'on n'est pas obligé d'utiliser des fonctions anonymes comme gestionnaire d'événements.

C'est souvent pratique, mais parfois on peut vouloir séparer le code. Par exemple, si le gestionnaire d'événements devient trop long, il est parfois plus facile de lire le programme séparément.

[lp1405]

Types en JavaScript

Types « primitifs »

- boolean → true , false
- null
- undefined
- number → 1234 , 3.14159
- string → "bonjour"

Type Object

→ { nom: "Leïla", score: 123 }
["orange", "banane"]

Array = Objet

typeof

JavaScript gère automatiquement les types. On n'a pas tout le temps besoin de s'en occuper, mais ... c'est important de bien les comprendre.

En JS, il existe uniquement 5 types primitifs (immuables) et le type Objet. Les tableaux ne sont qu'une forme d'Objet.

Pour connaître le type d'une expression on peut utiliser « typeof ».

[lp1404]

undefined & typeof

```
exemple(10);  
exemple();  
  
function exemple(a)  
{  
  if(typeof a === "undefined")  
  {  
    a=123;  
  }  
}
```

En JavaScript on peut passer moins d'arguments que prévu à une fonction. Les arguments manquants auront pour type undefined.

La seule manière toujours correcte de vérifier si une variable n'est pas définie est d'utiliser :
`typeof a === "undefined"`

Fonctions : paramètres

```
var e=1;  
var c="bonjour";  
var o={ nom: "Leïla", score: 20 };  
  
exemple(e,c,o);  
  
console.log(e,c,o);  
  
function exemple(x,y,z)  
{  
    x=100;  
    y="au revoir";  
    z.nom="Tom";  
    z.score=200;  
}
```

types « primitifs »

e : 1
c : "bonjour"
o : { nom: "Tom",
score: 200 }

primitifs : par valeur

objets : par référence

Ici e et c sont des variables avec des types primitifs (number et string).
o est un objet.

On passe e,c et o en paramètre de la fonction exemple.

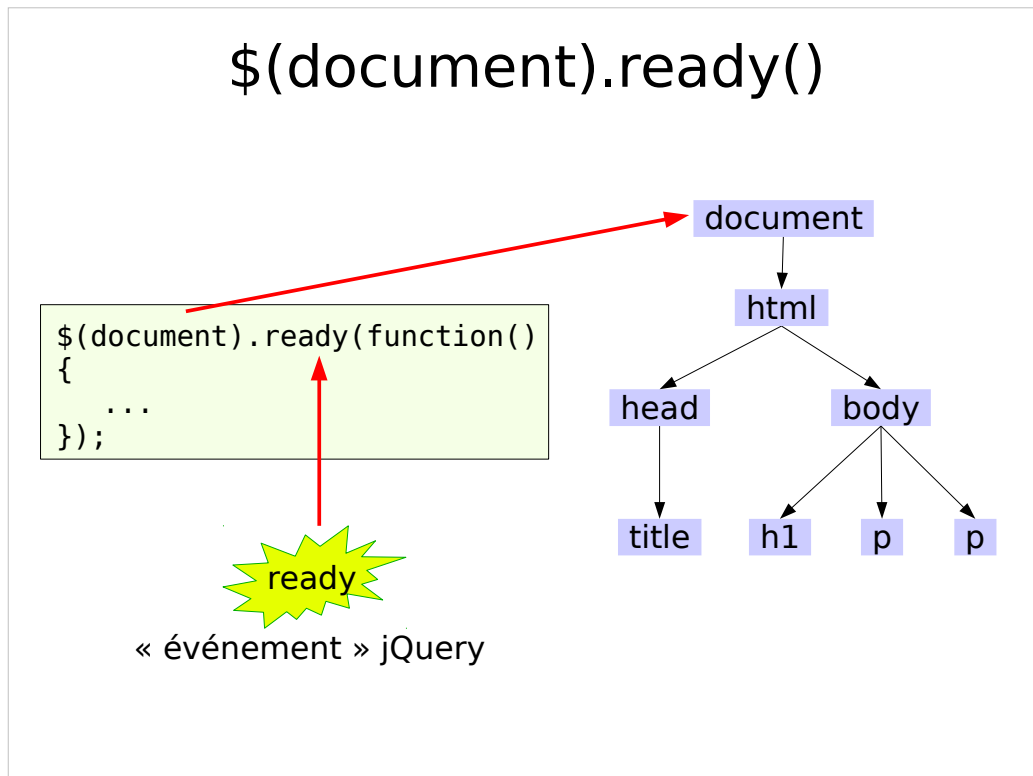
Dans la fonction exemple les paramètres correspondants (x,y et z) sont modifiés.

On remarque alors que e et c n'ont pas été modifiés (passage par valeur).

Par contre o a bien été modifié (passage par référence)

2ème partie

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



`$(document).ready()` figure dans tous nos programmes. On va prendre un moment pour essayer de comprendre.

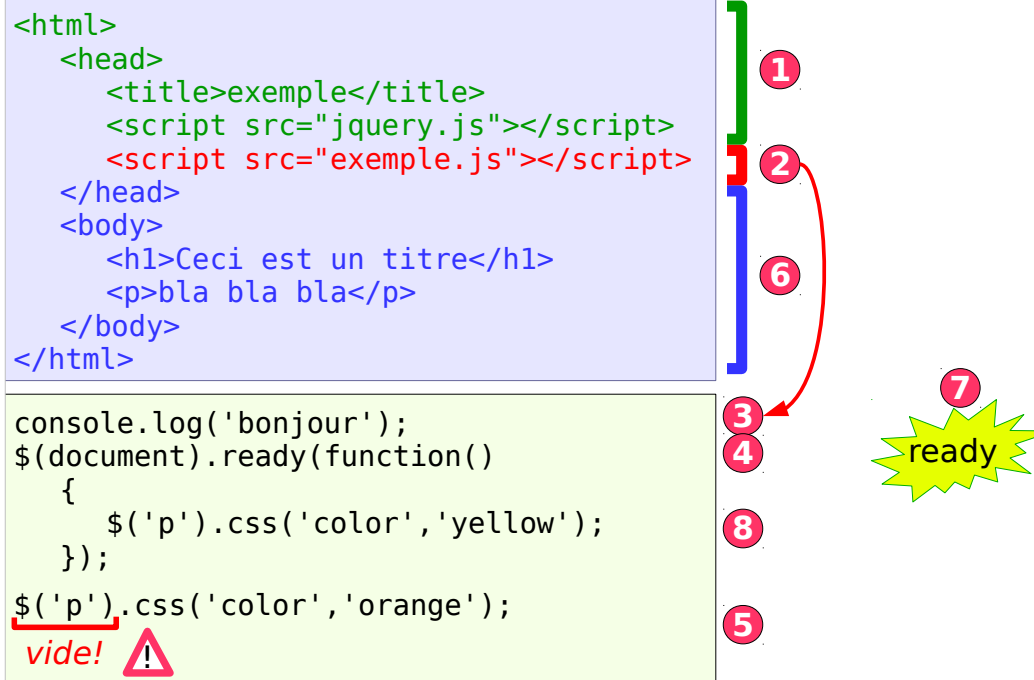
`$(document)` est une liste jQuery contenant un seul élément : `document`.

`document` est l'objet DOM tout en haut de l'arbre DOM.

`ready` est un événement jQuery qui est appelé quand le DOM est prêt.

On est donc dans le même cadre d'utilisation que n'importe que autre gestionnaire d'événements (comme `.click()`). La différence est que `.click()` est un événement lié à une action de l'utilisateur alors que `.ready()` est un événement automatiquement déclenché par le navigateur (et jQuery).

Ordre d'exécution



Pourquoi `$(document).ready()` est-il nécessaire ?

Le navigateur construit l'arbre DOM progressivement.

D'abord la partie verte [1]. Ensuite il arrive au programme exemple.js [2], qu'il exécute.

Important: remarquez que quand exemple.js est exécuté, l'arbre DOM partie bleue [6] n'existe pas encore !

[3] Le programme s'exécute

[4] Le gestionnaire d'événement ready est ajouté (mais pas encore appelé)

[5] Ici on construit la liste jQuery `$('p')`. Comme l'arbre DOM de la partie bleue n'existe pas encore, la liste jQuery est vide et il ne se passe rien !

[6] Le programme a fini. L'arbre DOM restant est construit.

[7] L'arbre DOM est prêt Le gestionnaire d'événements ready est appelée.

[8] `$('p')` contient bien le paragraphe.

3ème partie

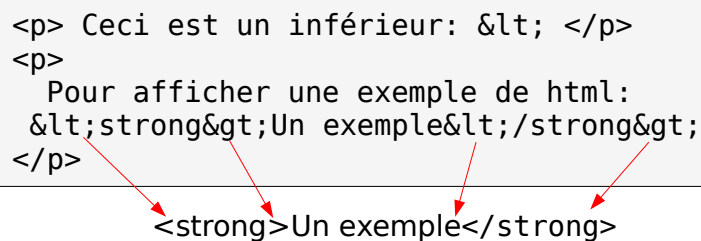
HTML & texte

Entités

>	>	>
<	<	<
"	"	"
'	'	'

HTML:

```
<p> Ceci est un inférieur: &lt; </p>
<p>
  Pour afficher une exemple de html:
  &lt;strong&gt;Un exemple&lt;/strong&gt;
</p>
```



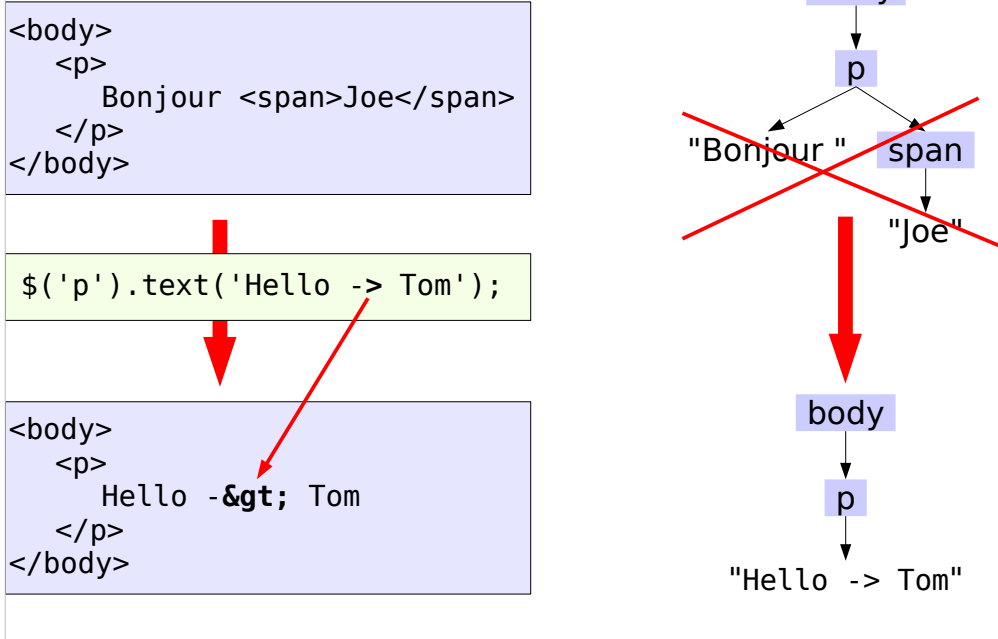
Un exemple

Le HTML et le texte « brut » se ressemblent beaucoup, mais sont en réalité deux formats différents. Les confondre peut conduire à des problèmes d'affichage et, plus grave, à des problèmes de sécurité.

En HTML, certains caractères ont un sens particulier. Par exemple "<" est le début d'une balise. Si on veut représenter ces caractères, on doit utiliser les « entités » correspondantes. Dans cet exemple on doit utiliser "<" au lieu de ">"

[lp1399]

jQuery : .text()



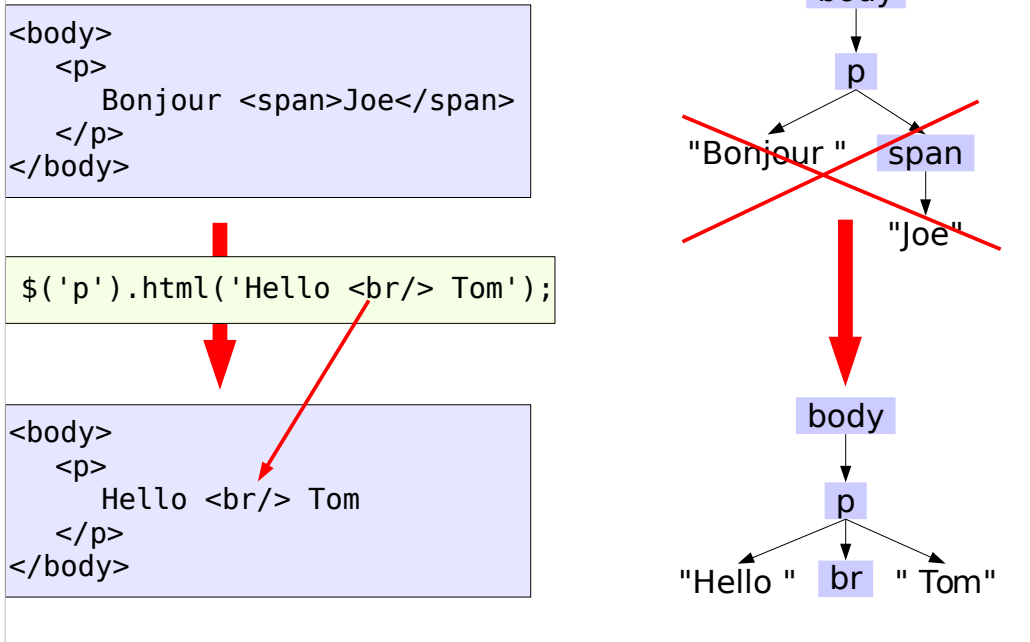
La fonction jQuery `.text()` permet de lire le texte contenu dans un élément. `.text('...')` permet de créer / remplacer du texte.

Si vous utilisez `.text('...')` sur un élément qui contient déjà d'autres éléments, ils sont tous supprimés avant que votre texte soit ajouté.

Dans l'arbre DOM le texte est représenté par des noeuds Text. En général, on ne les dessine pas sur les schémas, mais ils existent bien dans l'arbre.

La fonction `.text('...')` ajoute bien du texte et pas du HTML. Les caractères spéciaux (comme le `">"`) sont correctement gérés.

jQuery : .html()



Il ne faut pas confondre `.text()` et `.html()`.

`.html('...')` permet d'ajouter du html à l'intérieur d'éléments existants dans l'arbre.

D'abord, tout le contenu de l'élément cible est supprimé.

Ensuite, le HTML à ajouter est transformé en un fragment d'arbre DOM qui est ensuite inséré dans l'arbre.

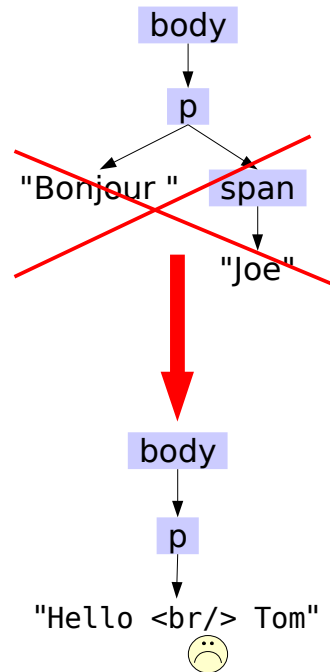
[lp1397]

jQuery : .html()

```
<body>
  <p>
    Bonjour <span>Joe</span>
  </p>
</body>
```

```
$('p').text('Hello <br/> Tom');
```

```
<body>
  <p>
    Hello &lt;br/&gt; Tom
  </p>
</body>
```



Si on utilise `.text()` à la place de `.html()`, les balises ne sont pas transformées en éléments de l'arbre: elles sont considérées comme du texte.

[lp1396]

jQuery : .val()

uniquement formulaire

```
$('#select').val('Chocolat');
```

```
$('#input').val('essai');
```

```
$('#textarea').val('Un exemple...');
```

<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>

Il y a parfois des confusions entre .text() et .val().

Ces deux fonctions ne font pas du tout la même chose.


.val() permet de lire et d'écrire la valeur d'un champs de formulaire.

.text() permet de lire et d'écrire le texte contenu dans un élément de l'arbre.

[lp1395]

Exemple

```
<input id="saisie" type="text">
<input id="ajouter" type="button" value="Ajouter"/>
<p></p>
```



bonjour

Ajouter

bonjour

```
$('#ajouter').click(function()
{
  $('#p').html($('#saisie').val());
});
```

.text()



a <<joe>>

Ajouter

a

DOM XSS



```
<p>a<<joe>></p>
```

Dans cet exemple, l'utilisateur tape du texte dans un champs texte `<input>`, puis appuie sur un bouton. La valeur saisie est ensuite affichée à l'aide de `.html()` dans le paragraphe.

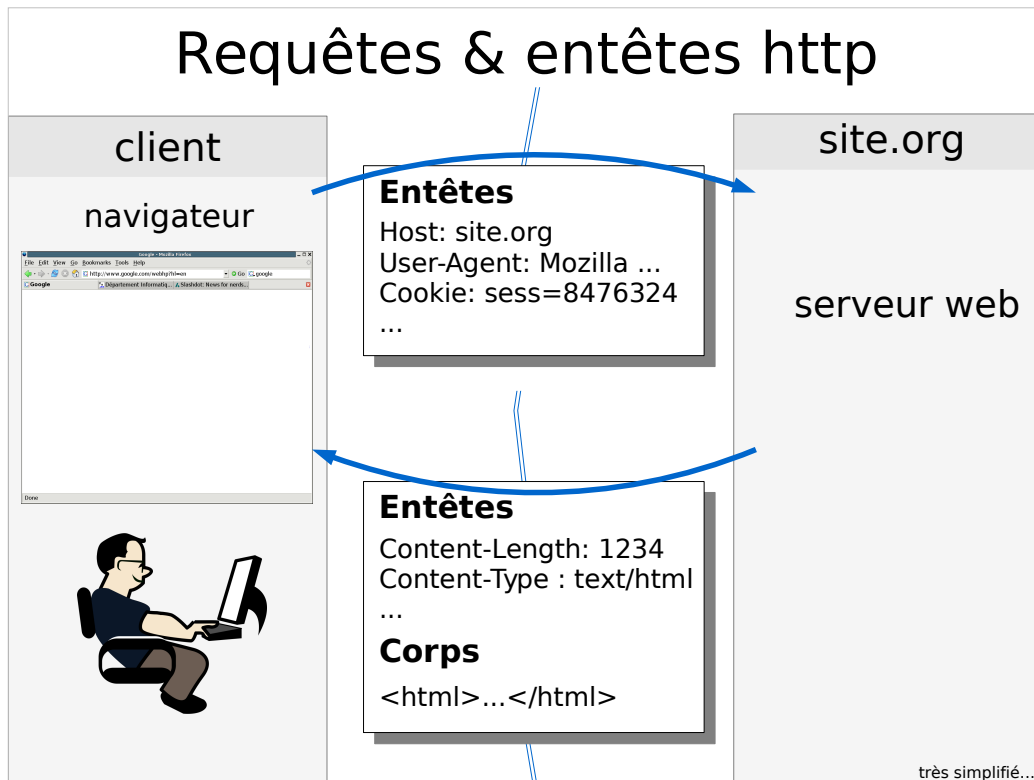
Si l'utilisateur a saisi du texte simple (des lettres ordinaires), le programme fonctionne correctement.

Si l'utilisateur rentre des caractères spéciaux, il peut y avoir des problèmes d'affichage. Par exemple un "<" sera considéré comme un début de balise. C'est embêtant.

On peut imaginer un contexte où un utilisateur malveillant fourni du texte contenant du JS. Plus tard un autre utilisateur (par exemple un admin) lit la page. Si un programme JS copie avec `.html()` le texte fourni par l'utilisateur malveillant, alors celui-ci pourra voler ses cookies...

4ème partie

http



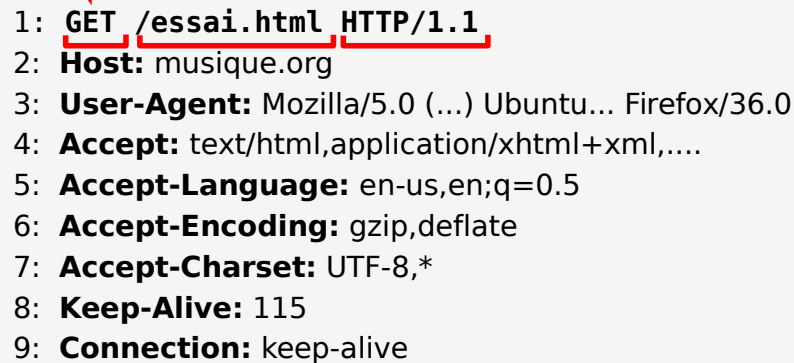
La communication entre le navigateur et le serveur se fait dans un langage (protocole) appelée "http". On peut l'imaginer comme une « discussion ». Dans une requête GET simple, le navigateur envoie des informations au serveur. Ces informations ne sont pas visibles par l'utilisateur. Ce sont des « entêtes » http.

De même, la réponse du serveur contient à la fois des « entêtes » invisibles et le « corps » : le HTML. Les entêtes étant invisibles, on a tendance à les oublier. Elles jouent pourtant un rôle important. (Attention à ne pas confondre les entêtes <head> du HTML et les entêtes du http. Les entêtes <head> du HTML se trouvent dans le corps de la réponse HTTP)

La requête

Bonjour, je voudrais la page `essai.html` qui se trouve sur le site `musique.org`. Je suis un navigateur firefox. J'accepte les pages aux formats

GET, POST...



```
1: GET /essai.html HTTP/1.1
2: Host: musique.org
3: User-Agent: Mozilla/5.0 (...) Ubuntu... Firefox/36.0
4: Accept: text/html,application/xhtml+xml,...
5: Accept-Language: en-us,en;q=0.5
6: Accept-Encoding: gzip,deflate
7: Accept-Charset: UTF-8,*
8: Keep-Alive: 115
9: Connection: keep-alive
```

Console

On peut imaginer l'échange entre le navigateur et le serveur comme une « discussion » dans un langage appelé « http ». Elle est traduite en français ici.

Ici, le navigateur demande une page au serveur. Il envoie de nombreuses informations pour indiquer au serveur qu'il est, ce qu'il veut, et ce qu'il est capable d'accepter comme réponse.

Cette requête GET ne contient que des entêtes (pas de corps).

Toutes ces informations sont visibles à l'aide de Firebug (onglet Réseau).

[lp1392]

La réponse

La page demandée a bien été trouvée. Je suis un serveur Apache version... Voici un cookie pour pouvoir vous re-identifier plus tard. Voici la page demandée au format html encodé en UTF-8.

200,404,304, 500...

entêtes	1: HTTP/1.1 200 OK
	2: Date: Mon, 16 Mar 2015 11:24:21 GMT
	3: Server: Apache/2.2.22 (Debian)
	4: X-Powered-By: PHP/5.4.36-0+deb7u3
	5: Set-Cookie: SESSf8818008...
	6: Last-Modified: Mon, 16 Mar 2015 10:41:34 GMT
	7: Content-Type: text/html; charset=utf-8
corps	8:
	9: 12d79
	10: <!DOCTYPE html ...>
	11: <html ...>
	12: <head>
	13: <title>Le site Musique.org</title>

Le serveur répond. Il envoie un code pour dire au navigateur s'il a bien réussi (200) à fournir la page demandée. Quelques exemples de codes d'erreur : 404, 500,...

Cette réponse contient à la fois des entêtes et un corps (du HTML).

Une information importante est le « Content-Type ». Elle dit au navigateur comment il doit afficher le corps.

text/html : il doit l'afficher en tant que HTML

image/png : il doit l'afficher en tant qu'image

...

En programmation web, on a souvent besoin d'indiquer le Content-Type.

[lp1391]



Une des informations cruciales transmises dans les entêtes est le cookie.

Les cookies sont initialement fournis par le serveur. Ensuite, à chaque requête, le navigateur renvoie le cookie au serveur. De cette manière, le serveur peut identifier un utilisateur précis. C'est ce mécanisme qui permet d'attribuer des identités à des utilisateurs (connexion à un compte, ...).

Un cookie de session doit rester secret. Si un utilisateur malveillant réussit à connaître votre cookie, il peut prendre votre identité.

Ca peut arriver si le développeur (PHP, JS, ...) confond HTML et texte... ouvrant la voie à une attaque appelée XSS

5ème partie

HTML ajouté

jQuery : .is()

```
$('#liste').is('ul')      → true  
$('#l2').is('#liste li') → true  
$('#l2').is('ul')       → false  
$('body').mousedown(function()  
{  
    if($(this).is('#l1')){ ... }  
});
```

```
<ul id="liste">  
  <li id="l1">Tom</li>  
  <li id="l2">Joe</li>  
</ul>
```

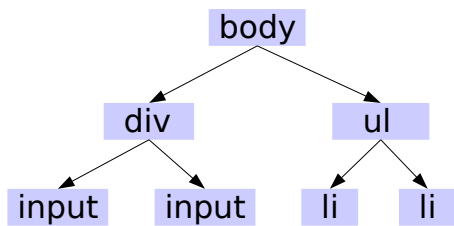
La fonction jQuery .is() permet de vérifier si un élément correspond bien à un sélecteur.

C'est souvent utilisé dans les gestionnaires d'événements où "this" peut prendre plusieurs valeurs.

[lp1389]

Ajout DOM : HTML

```
<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 form. It features a text input field, a button labeled 'Ajouter', and a list containing 'Tom' and 'Joe'.

Dans cet exemple, un utilisateur peut saisir du texte dans un champs texte puis appuyer sur un bouton. Le texte saisi est alors ajouté à une liste.

On veut aussi que l'utilisateur faire une action quand l'utilisateur clique sur les éléments de la liste.

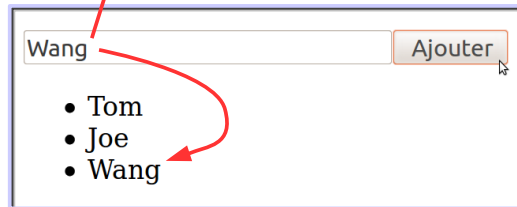
[lp1388]

Ajout DOM : JS

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

$('li').mousedown(function(){
    alert('mousedown!');
});
```

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



Le JS est constitué de deux parties:

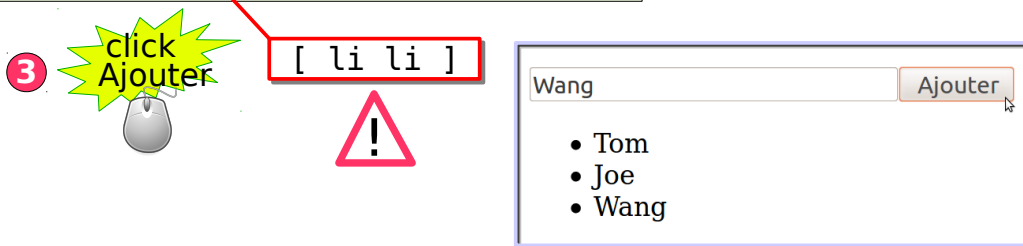
- 1) le gestionnaire d'événements permettant d'ajouter le texte saisi à la liste
- 2) le gestionnaire d'événements pour faire une action quand l'utilisateur clique sur un élément de la liste.

[lp1387]

Ajout DOM : JS

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

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



Ce code contient un bug. Pour le comprendre, il faut suivre, étape par étape, le déroulement du programme.

1) le gestionnaire de click est ajouté. Mais pour l'instant l'utilisateur n'a pas cliqué. Donc la ligne "Wang" n'existe pas encore

2) On ajoute le gestionnaire mousedown sur les lignes existantes. Donc sur les deux premières (Wang n'existe pas encore).

3) l'utilisateur clique et

4) la ligne Wang est ajoutée

Si l'utilisateur fait mousedown sur Wang, rien ne se passe... :-)

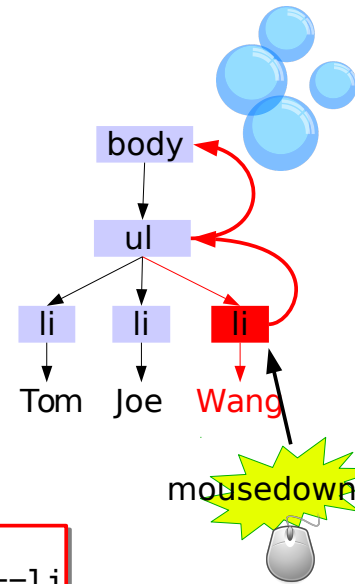
[lp1386]

Bubbling

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

$('li').mousedown(function(){
    alert('mousedown!');
});

$('ul').mousedown(function(e){
    if($(e.target).is('li'))
    {
        alert('mousedown!');
    }
});
```



Pour remédier à ce problème, deux solutions différentes sont possibles.

1) ajouter un gestionnaire juste après le `append()`. Ca demande de réorganiser / dupliquer du code.

2) profiter du "bubbling"

On va préférer la (2)

L'événement `mousedown` est d'abord transmis à `` puis à son père ``, puis à son père `<body>...`

On peut donc installer le gestionnaire `mousedown` sur `` au lieu de ``. `` est bien présent au démarrage du programme.

Il reste alors à vérifier dans le gestionnaire que l'utilisateur a bien cliqué sur un `` (et pas sur le ``).

.on()

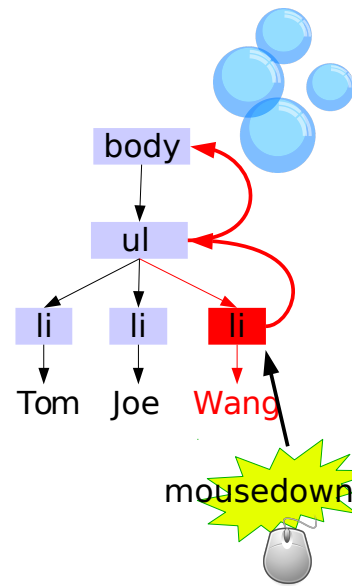
```
$('#ajouter').click(function()
{
    var ligne=$('#<li></li>');
    var texte=$('#saisie').val();
    ligne.text(texte);
    $('#liste').append(ligne);
});

$('#li').mousedown(function(){
    alert('mousedown!');
});

$('#ul').on('mousedown','li',
    function(){
        alert('mousedown!');
    });
```

this==li

this==li



Cette opération est très courante. jQuery fournit donc une fonction appelée `.on()` qui permet de le faire simplement.

Remarquez que dans `.on()` "this" est `` alors que dans `$('#ul').mousedown()` vu à la page précédente, this était ``.

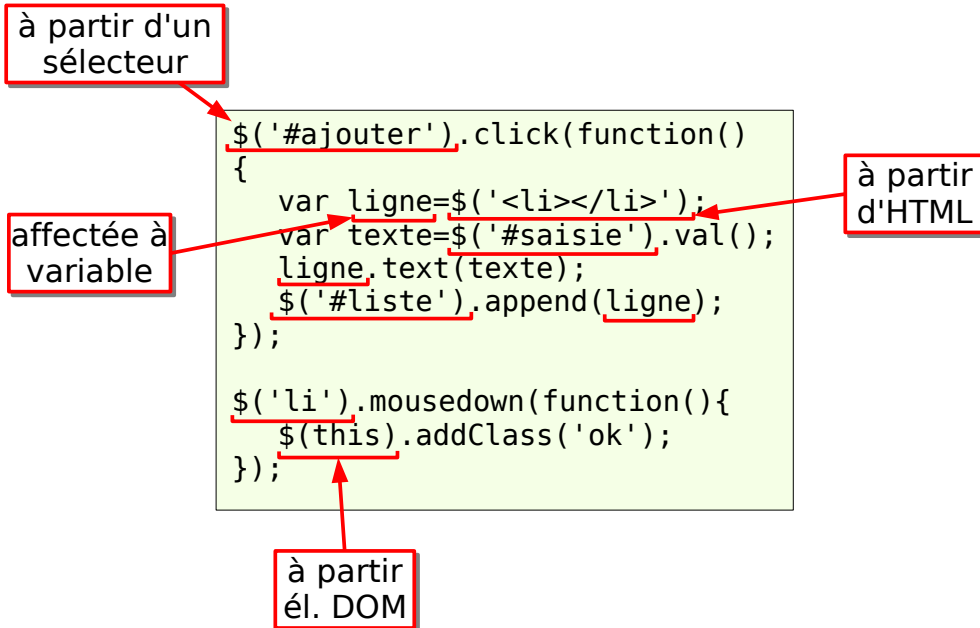
`.on()` est plus pratique.

[lp1384]

6ème partie

Listes jQuery

Listes jQuery



Les listes jQuery sont omniprésentes.

On peut les créer de plusieurs manières :

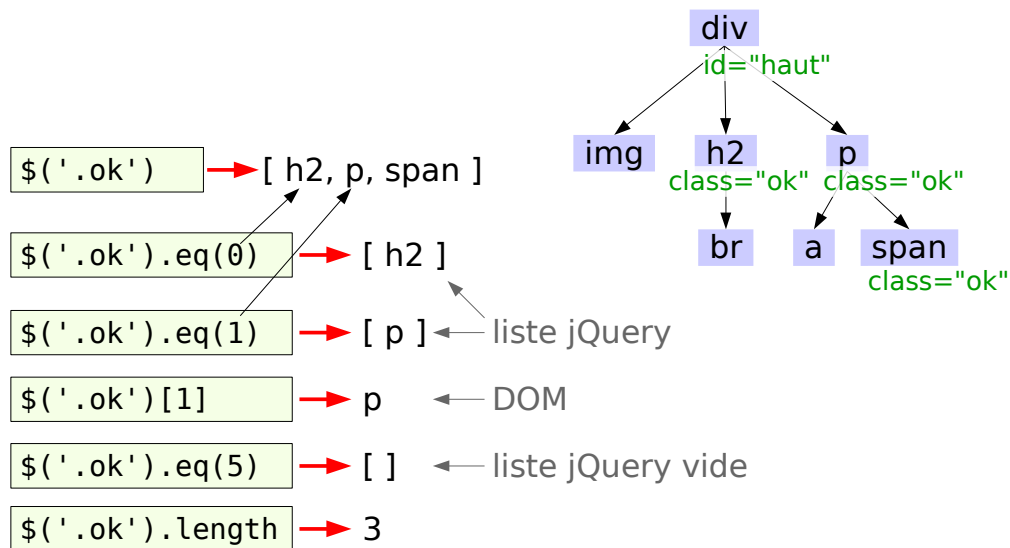
- 1) à partir d'un sélecteur CSS. ex: `$('#ajouter')`
- 2) à partir de HTML. ex: `$('')`
- 3) à partir d'un élément DOM. ex: `$(this)`

Dans tous les cas, il s'agit bien d'une liste jQuery. On peut appliquer les presque 200 fonctions associées aux listes jQuery.

Une liste jQuery peut être vue comme un tableau. On manipule souvent des listes avec un seul élément.

Les listes peuvent, bien sur, être affectées à des variables, passées en argument à des fonctions, etc.

Listes jQuery



Dans tous ces exemples on manipule une même liste, constituée des trois éléments `h2`, `p` et `span`.

Cette liste peut être vue comme un tableau.

La fonction `.eq()` permet d'accéder aux éléments de ce tableau. Elle renvoie une liste jQuery ne contenant qu'un seul élément.

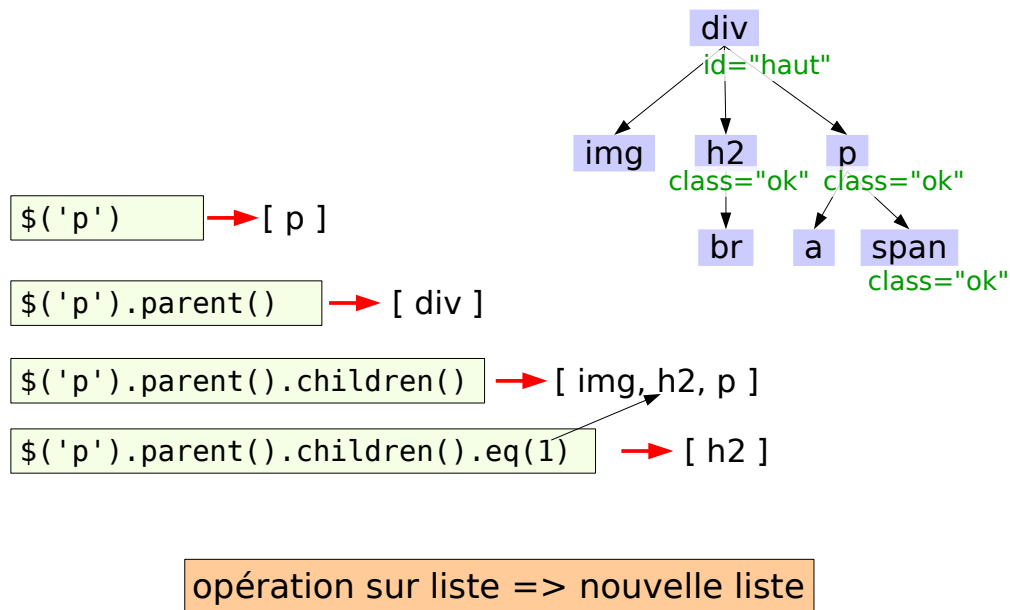
De manière similaire, l'opérateur `[]` (crochets) permet d'accéder à un élément. Il renvoie un élément DOM.

Comme pour un tableau, on peut utiliser `.length`

Dans toutes ces opérations, il n'y a pas de parcours d'arbre.

[lp1382]

Parcours d'arbre

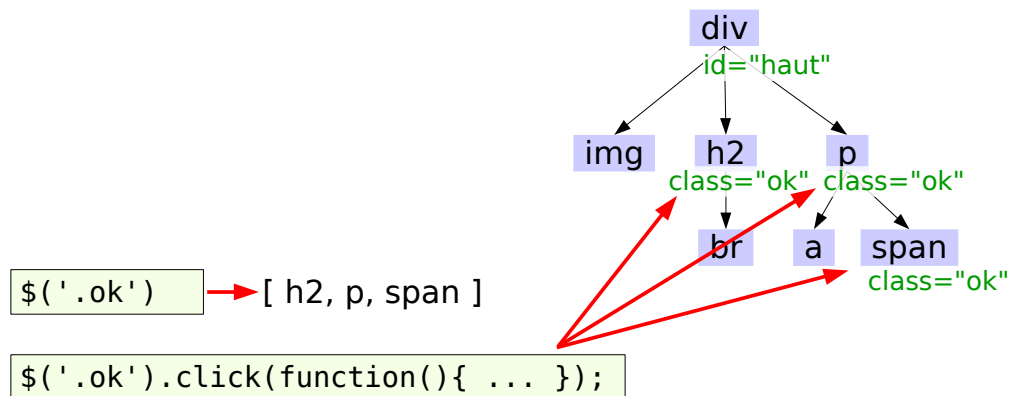


Les fonction jQuery renvoient en général une nouvelle liste jQuery. On peut donc les enchaîner.

Remarquez qu'à chaque fois la fonction jQuery est appliquée sur le résultat de la précédente fonction jQuery.

[lp1381]

.click()



`$('.ok')` est une liste contenant 3 éléments.

Donc `.click()` est appelé sur 3 éléments

Le gestionnaire d'événements est bien ajouté 3 fois.

Remarquez ici un effet probablement non-souhaité :

Si on clique sur ``, le gestionnaire va être appelée un première fois. Ensuite par "bubbling" l'événement remonte à `<p>`. Le gestionnaire est donc appelé une 2e fois.

[lp1380]

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