

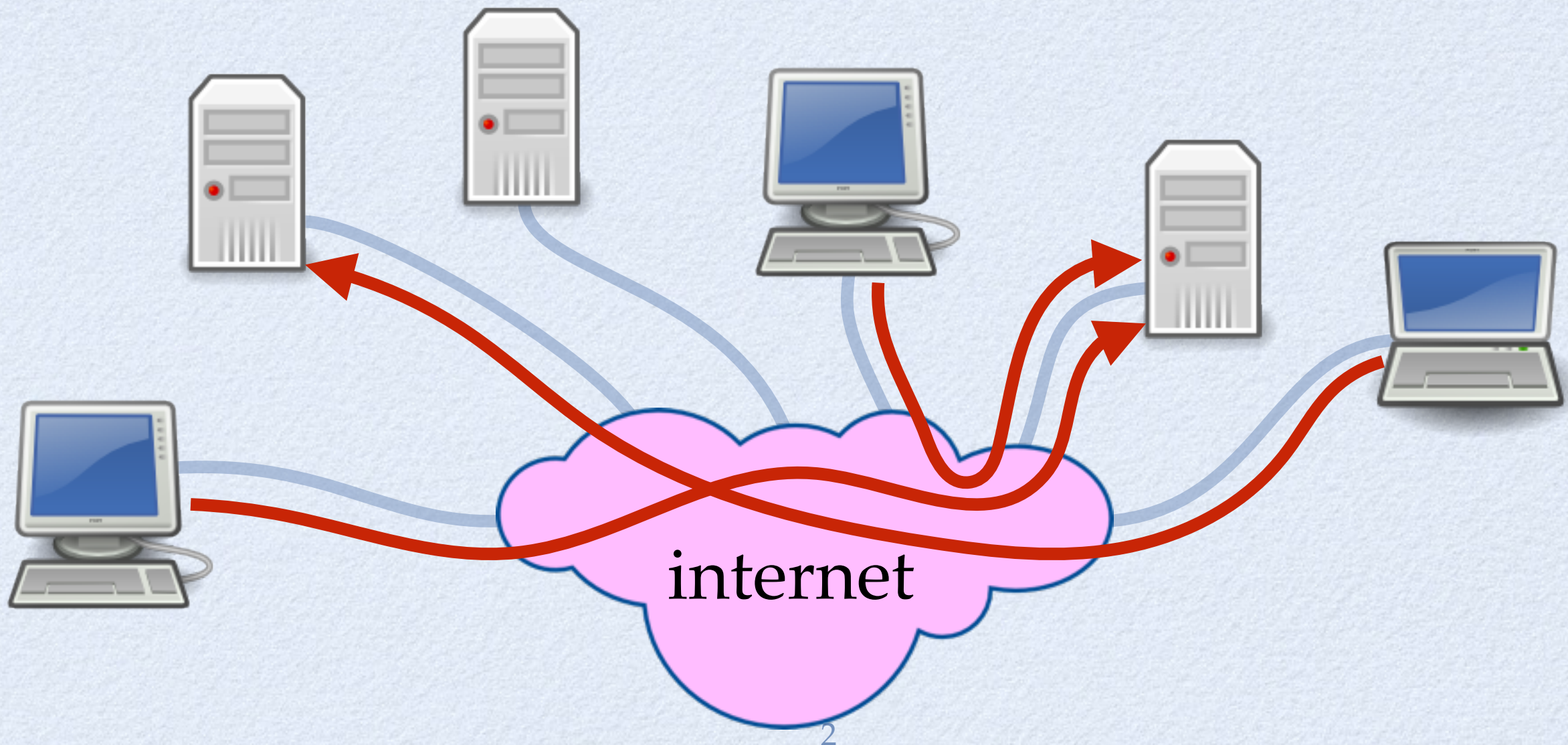
IFT1015 Programmation 1

Programmation web

Marc Feeley

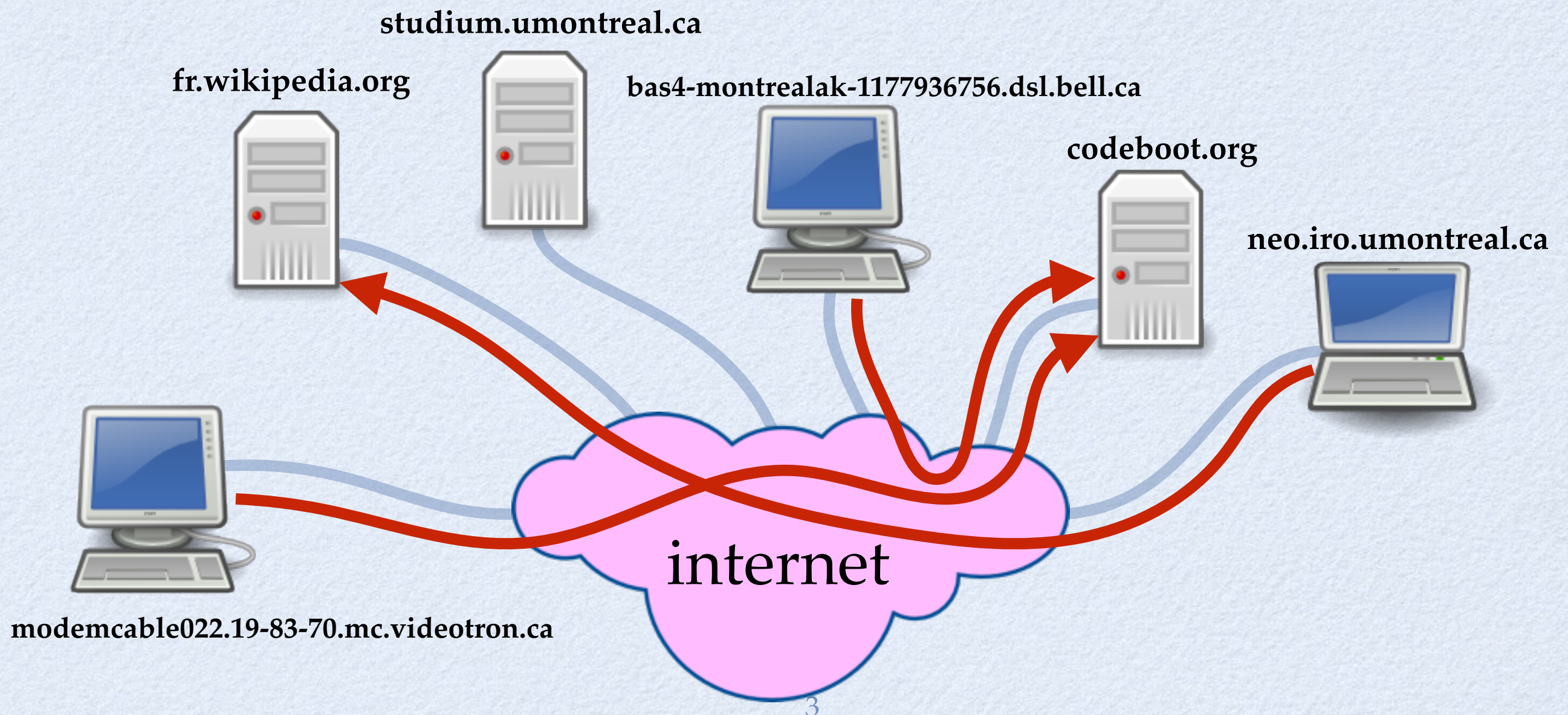
Internet

- L'internet est un réseau informatique qui relie un grand nombre d'ordinateurs et qui permet l'échange de données entre ces ordinateurs



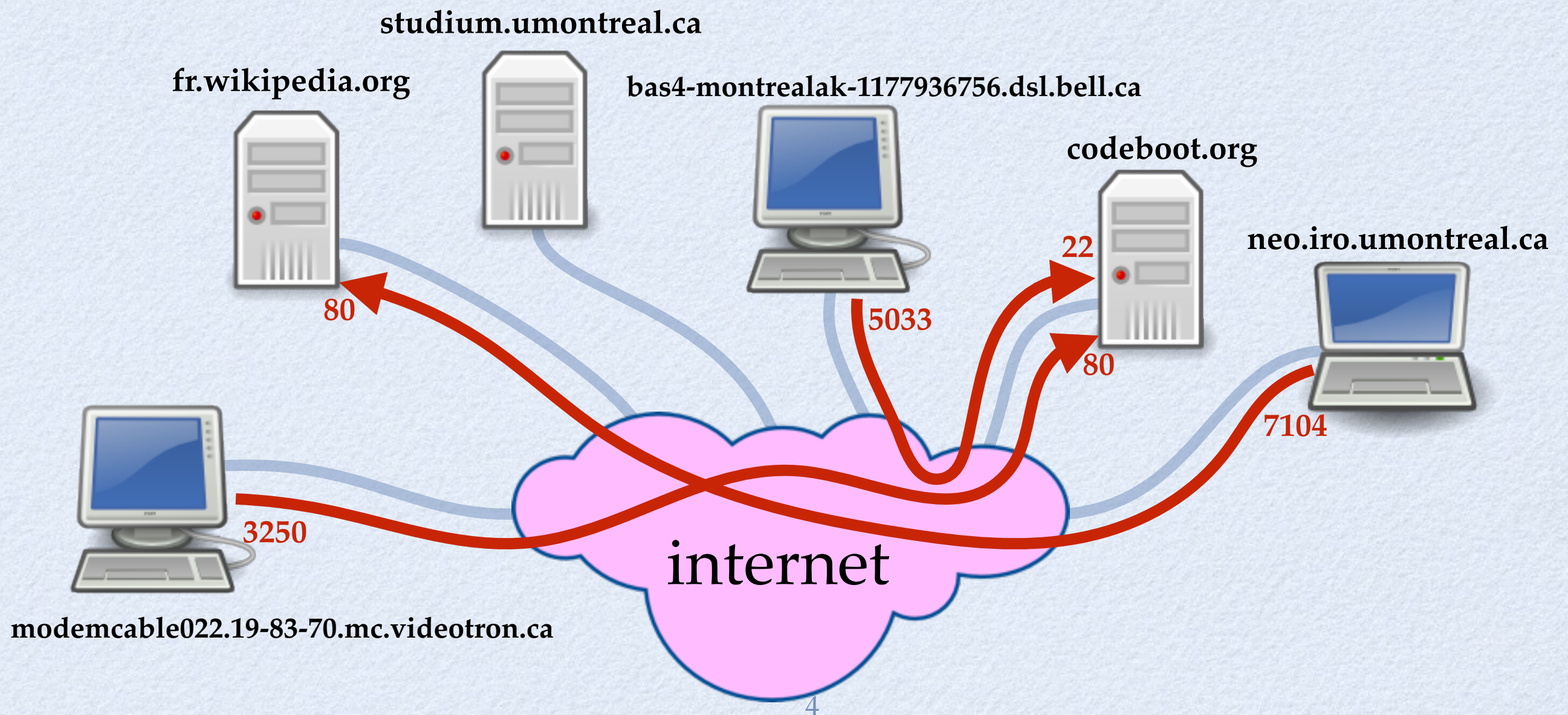
Internet

- Chaque ordinateur a une **interface réseau** (ou plusieurs) pour le brancher au réseau et chaque interface est identifiée par une **adresse IP** unique (si on ne tient pas compte des réseaux locaux)



Internet

- Chaque interface possède 65535 **ports** de communication
- Plusieurs numéros de port (< 1024) sont assignés à des **services** précis, par exemple 22 = shell sécurisé (SSH) et 80 = serveur web



Web

- Le web est basé sur une architecture client-serveur :
 - **serveur** = programme qui fournit des services, typiquement des documents à visionner
 - **client** = fureteur qui offre une interface graphique à l'utilisateur pour visionner des documents et naviguer d'un document à un autre (par exemple, Firefox et Google Chrome)
- Les documents sont identifiés par un **URL** (un genre de chemin d'accès mais au niveau web) :

http://fr.wikipedia.org:80/wiki/Lune

The diagram shows the URL **http://fr.wikipedia.org:80/wiki/Lune** with red brackets underneath it. Below the brackets are labels: 'protocole' under 'http', 'adresse IP du serveur' under 'fr.wikipedia.org', 'port (optionnel)' under ':80', and 'document sur ce serveur' under '/wiki/Lune'. The label 'port' is also underlined.

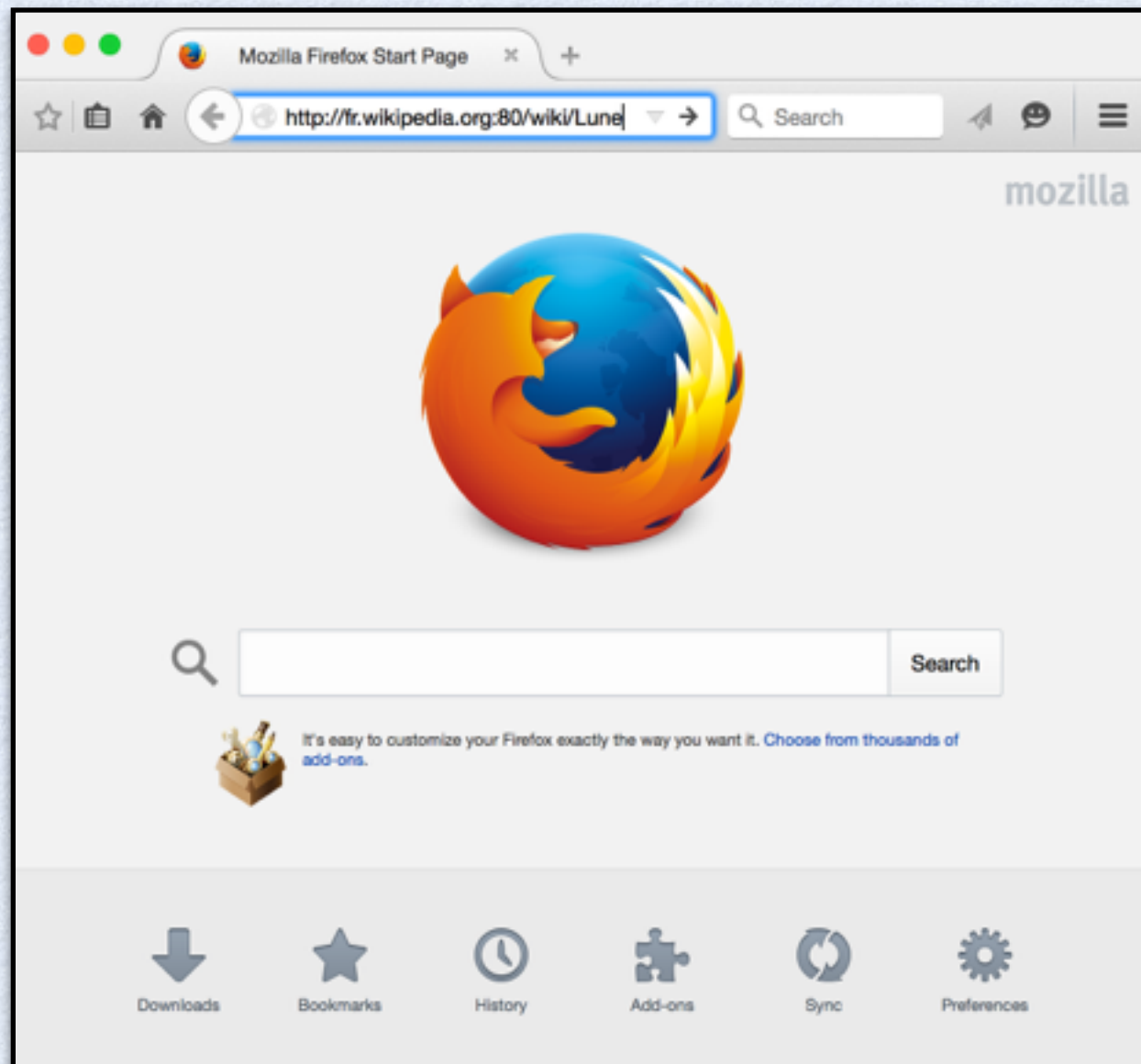
protocole *adresse IP du serveur* *port* *document sur ce serveur*

5 (optionnel)

Web

- Le **protocole HTTP** définit comment un **client** et un **serveur web** échangent l'information

client



serveur

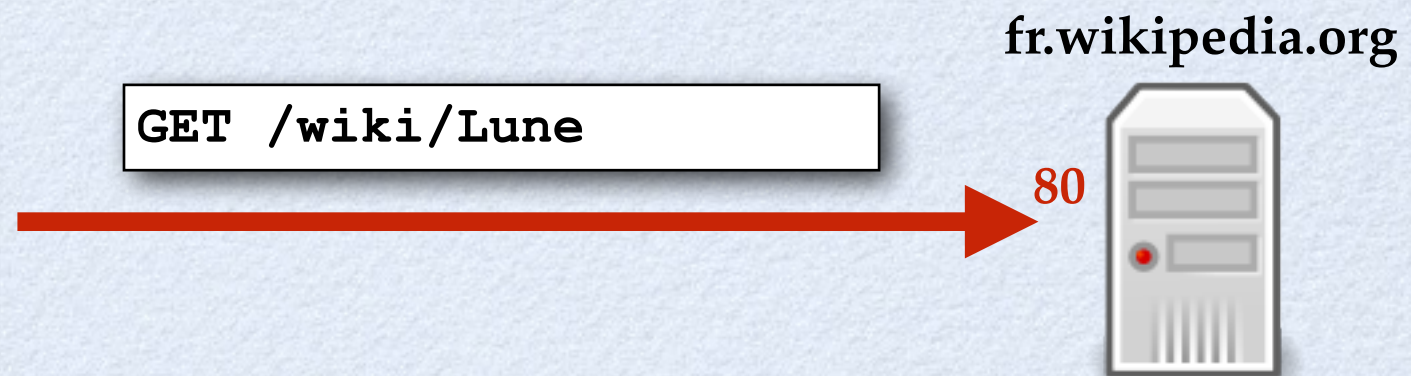
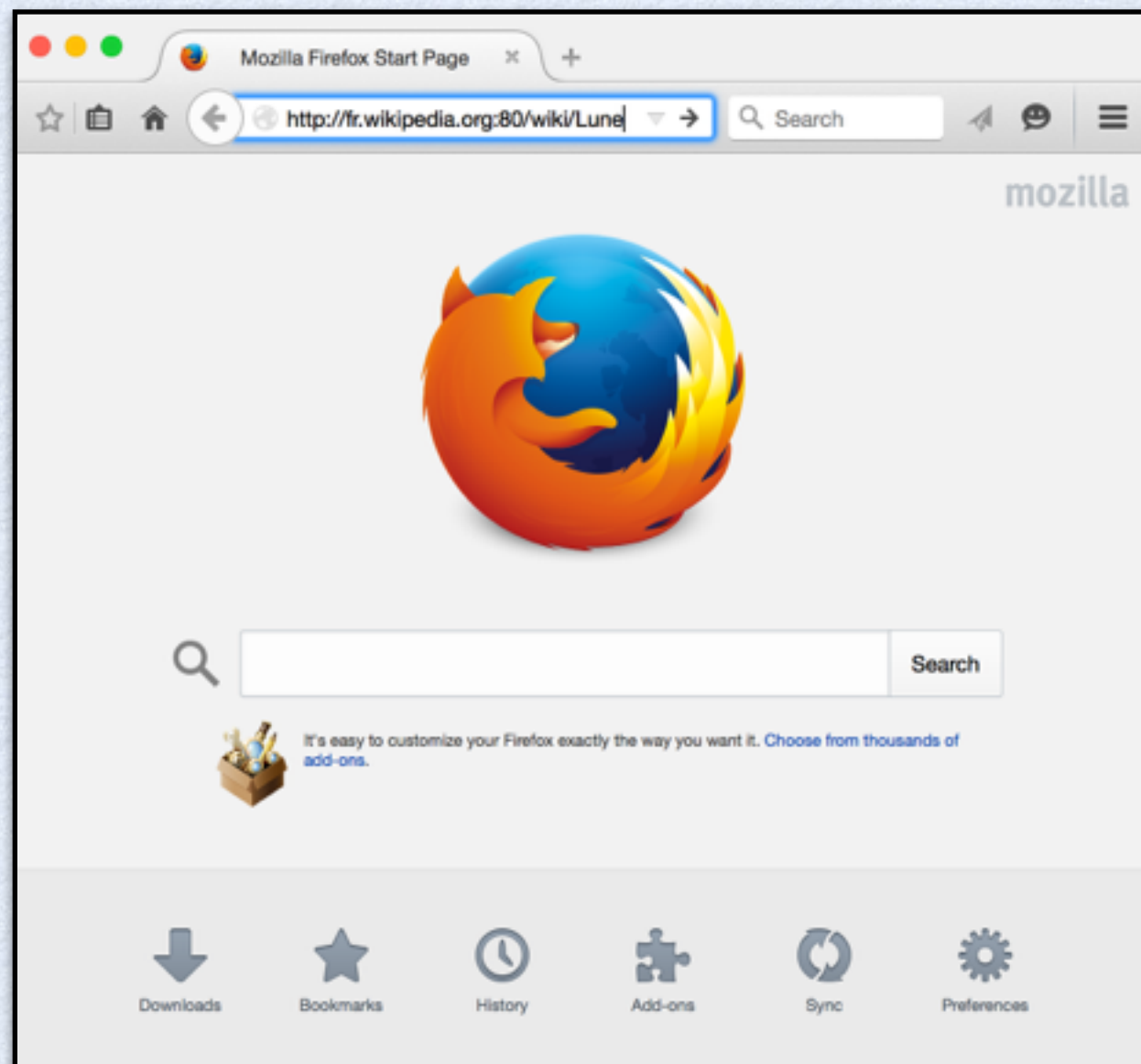
fr.wikipedia.org



Web

- Pour visionner `http://fr.wikipedia.org:80/wiki/Lune`

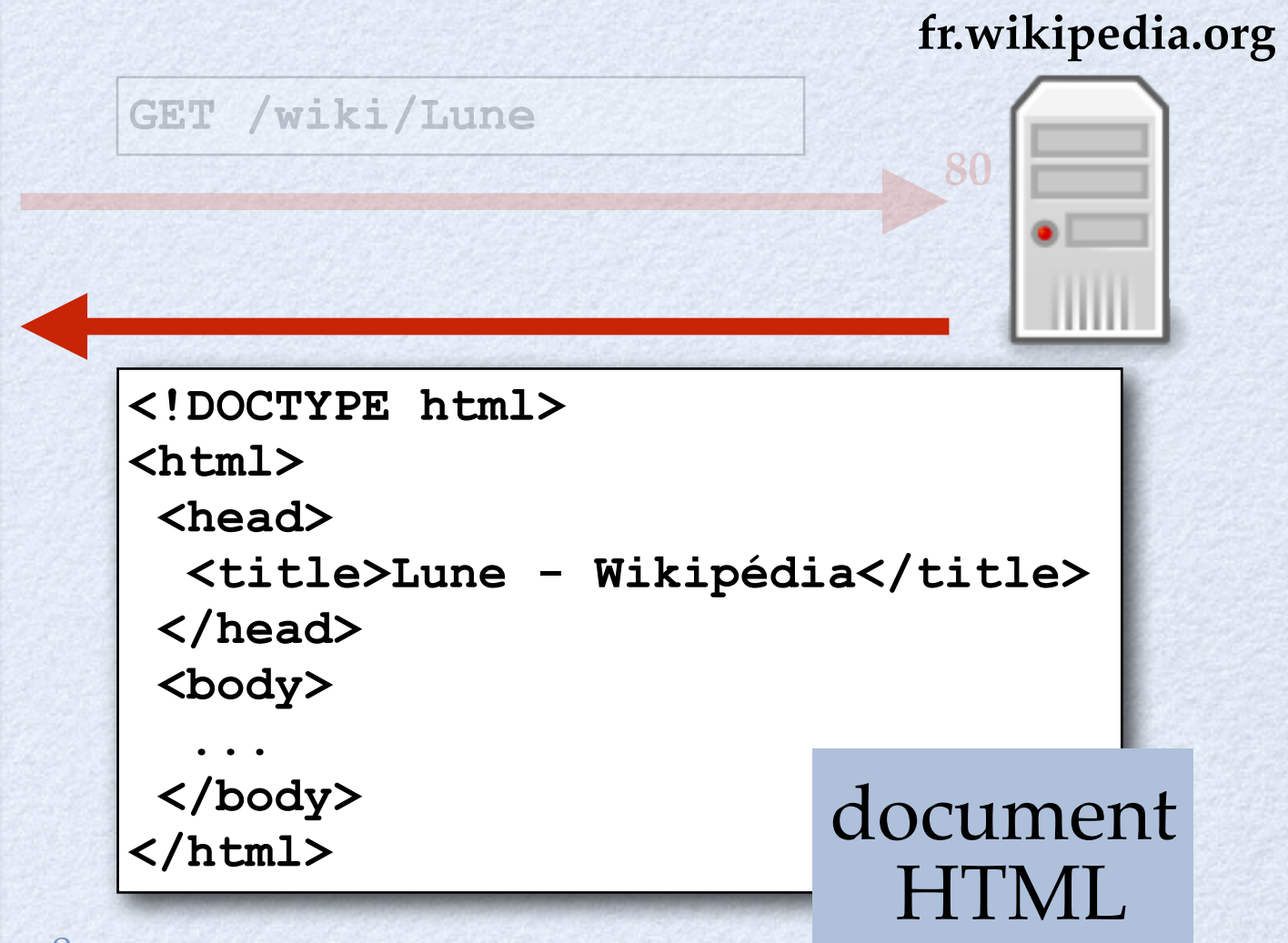
étape 1 : envoyer “**GET /wiki/Lune**” au serveur



Web

- Pour visionner `http://fr.wikipedia.org:80/wiki/Lune`

étape 2 : le serveur réponds en envoyant le document demandé au fureteur, qui l'affiche

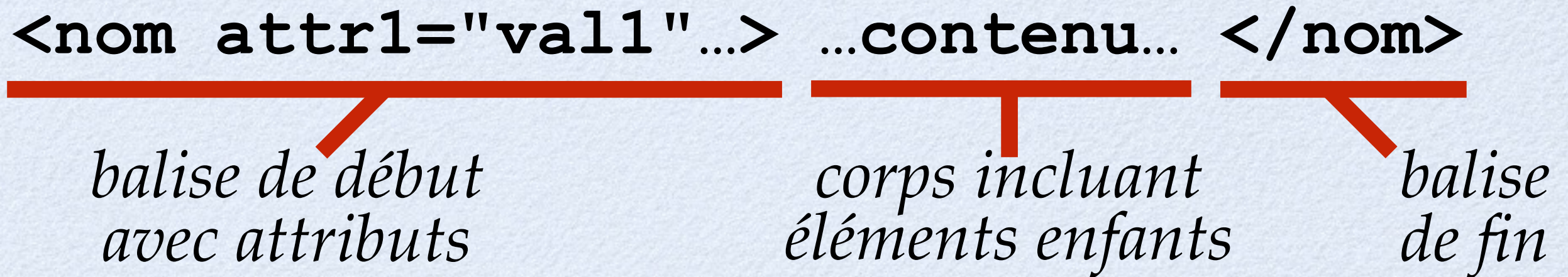


HTML

- Le langage **HTML** (*“Hypertext Markup Language”*) est utilisé pour encoder les documents à afficher
- C’est un fichier textuel contenant le texte du document et les directives de **disposition logique**
- Le document se décompose **hiérarchiquement** en **éléments**
- Un élément peut contenir des **éléments enfants**
- Le début et la fin d’un élément sont marqués par des **balises** (“tag”) qui indique le **type de l’élément** et ses **attributs** (identité, couleur, fonte, etc)

HTML

- Forme générale d'un élément balisé :



- Exemples :

`<i>je pense donc je suis</i>`

résultat
je pense donc je suis

`<p style="color: red">danger!</p>`

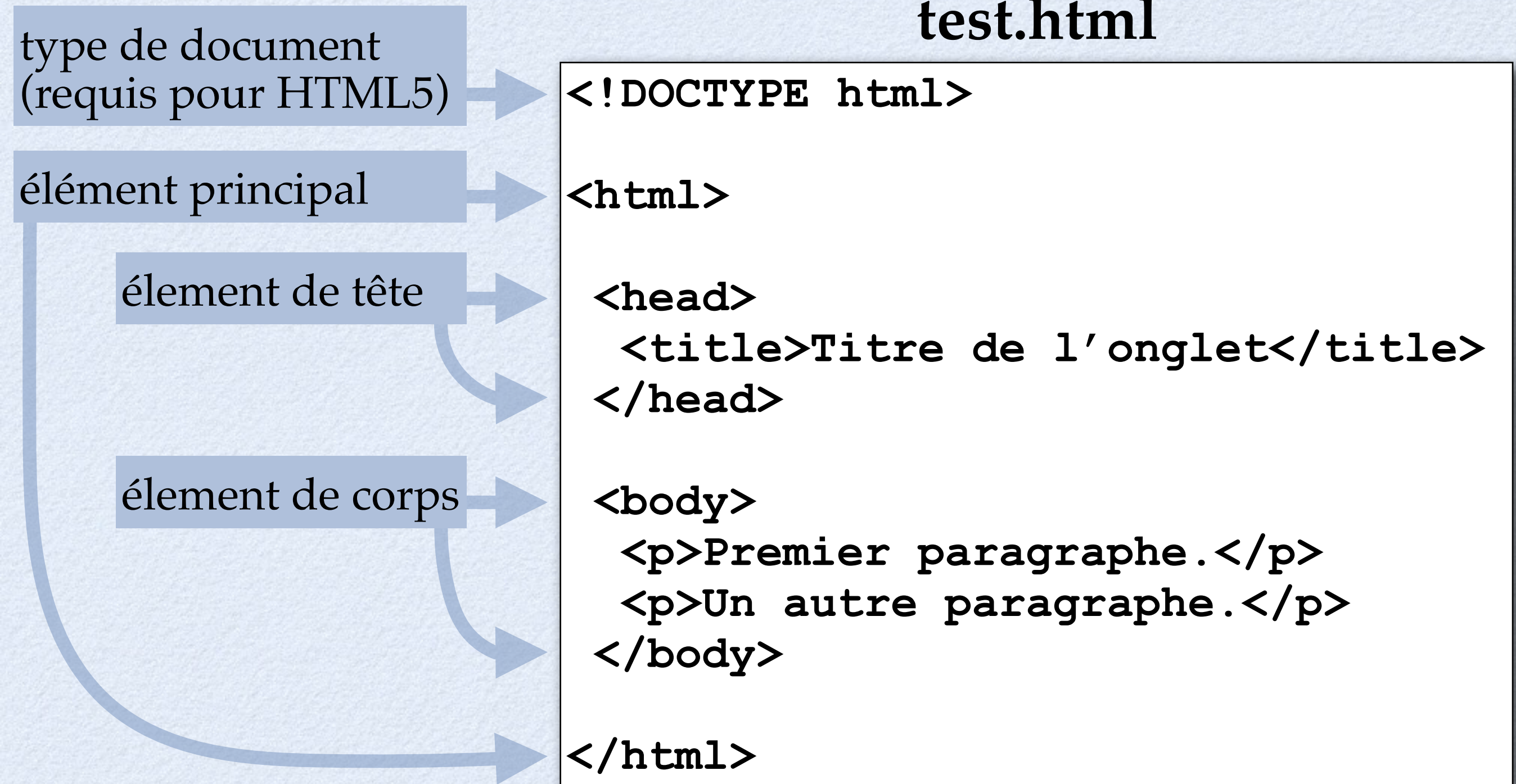
danger!

`<p> <i>beau</i> vélo! </p>`

beau **vélo!**

HTML

- Forme générale d'un document HTML :
test.html

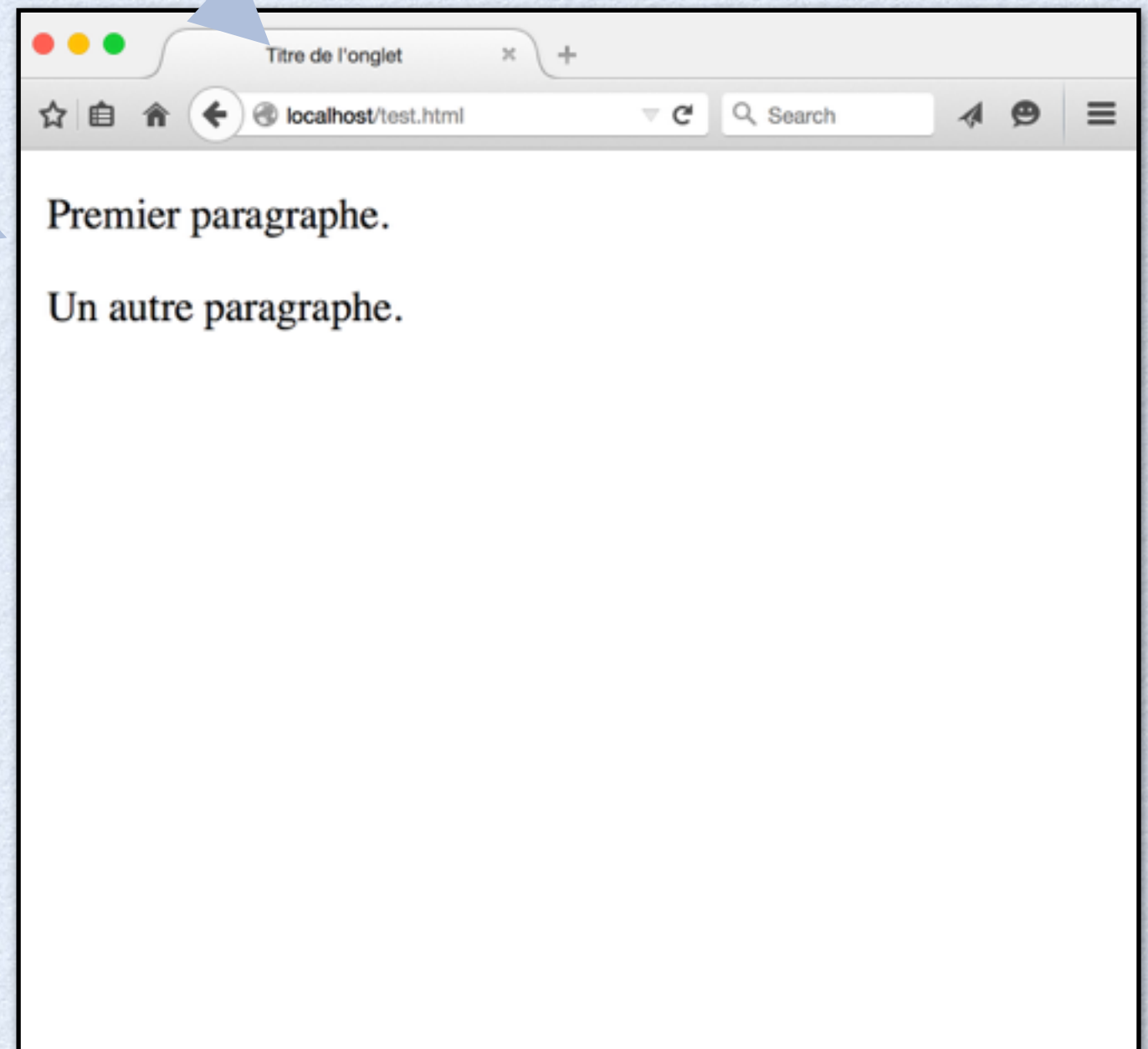


HTML

- Résultat dans le navigateur :

titre de l'onglet

corps du document



HTML

- Balises courantes :
 - ` . . . ` mettre de l'emphasis (*italique*)
 - ` . . . ` texte important (**gras**)
 - `<h1> . . . </h1>` titre de section principale
 - `<h2> . . . </h2>` titre de section secondaire
 - `<p> . . . </p>` paragraphe
 - `<pre> . . . </pre>` texte préformaté
 - `
` forcer saut de ligne
 - `<!-- . . . -->` commentaire non affiché

HTML

test2.html

résultat

```
<!DOCTYPE html>

<html>

  <head>
    <meta charset="UTF-8">
    <title>Test2</title>
  </head>

  <body>

    <h1>Chapitre 1</h1>

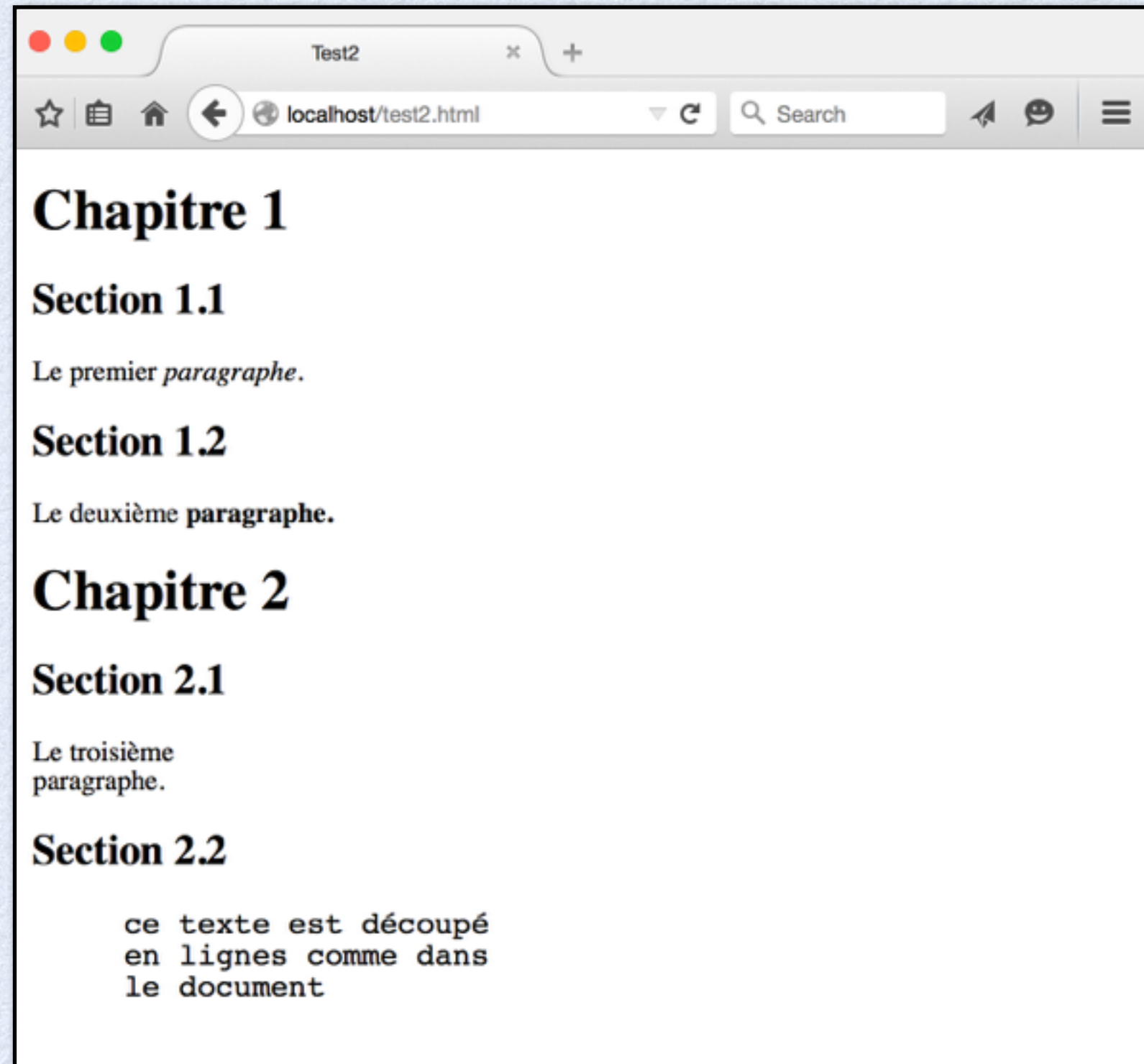
    <h2>Section 1.1</h2>
    <p>Le premier <em>paragraphe.</em></p>
    <h2>Section 1.2</h2>
    <p>Le deuxième
      <strong>paragraphe.</strong></p>

    <h1>Chapitre 2</h1>

    <h2>Section 2.1</h2>
    <p>Le troisième <br> paragraphe.</p>
    <h2>Section 2.2</h2>
    <!-- cette section est trop courte -->
    <pre>
      ce texte est découpé
      en lignes comme dans
      le document
    </pre>

  </body>

</html>
```



HTML

- Autres balises importantes :
 - `...` élément dans une liste
 - `...` liste d'éléments numérotés
 - `...` liste d'éléments non-numérotés
 - `...` un hyperlien
 - `` une image
- *URL* peut être un URL absolu ou un chemin d'accès relatif au document qui le contient :
 - `Google`
 - `Terre`

HTML

test3.html

résultat

```
<!DOCTYPE html>

<html>

  <head>
    <meta charset="UTF-8">
    <title>Test3</title>
  </head>

  <body>

    <h1>Astres</h1>

    <ul>

      <li> Le soleil :
        </li>

      <li> Planètes:
        <ol>
          <li>Mercure</li>
          <li>Venus</li>
          <li>Terre</li>
        </ol>
      </li>

      <li> La <a href="http://fr.wikipedia.org/wiki/Lune">Lune</a>.</li>

    </ul>

  </body>

</html>
```



HTML

- Balises l'interaction usager :
 - `<button>...</button>` un bouton cliquable
 - `<textarea>...</textarea>` entrée de texte
 - `<select>` menu d'options
 - `<option>...</option>`
 - `<option>...</option>`
 - `<option>...</option>`
 - `</select>`

HTML

test4.html

résultat

```
<!DOCTYPE html>

<html>

  <head>
    <meta charset="UTF-8">
    <title>Test4</title>
  </head>

  <body>

    Nom : <br> <textarea></textarea> <br> <br>

    Numéro de téléphone : <br> <textarea></textarea> <br> <br>

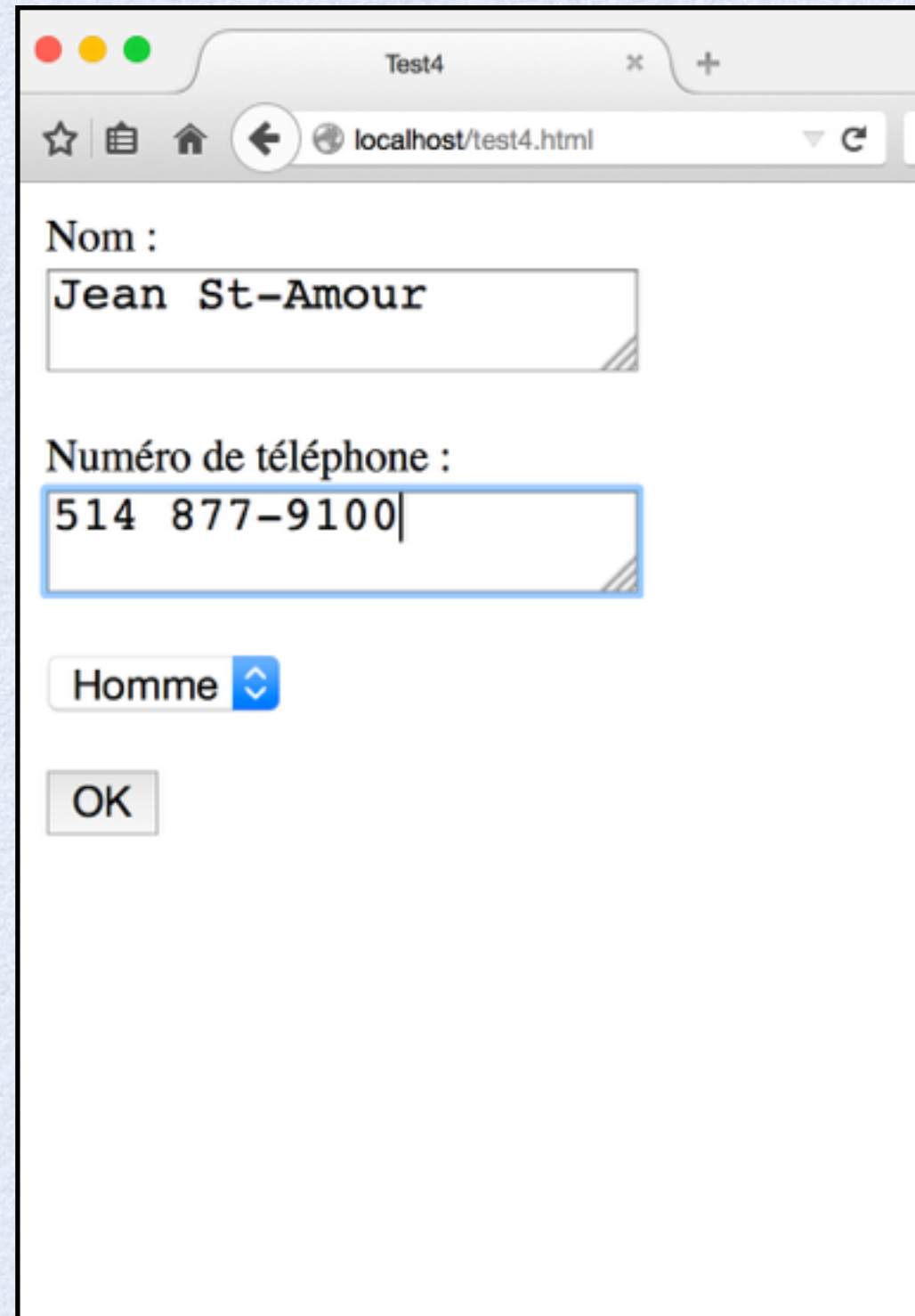
    <select>
      <option>Homme</option>
      <option>Femme</option>
    </select>

    <br> <br>

    <button>OK</button>

  </body>

</html>
```



The screenshot shows a web browser window titled "Test4" with the address bar displaying "localhost/test4.html". The rendered form contains the following elements:

- A label "Nom :" followed by a text input field containing "Jean St-Amour".
- A label "Numéro de téléphone :" followed by a text input field containing "514 877-9100".
- A dropdown menu with "Homme" selected.
- An "OK" button.

JavaScript et le web

- JavaScript a été conçu peu après la création du web pour la programmation d'applications qui exécutent dans le navigateur ("**web app**"), c'est-à-dire que du code exécutable JavaScript est placé dans le document envoyé au navigateur
- Aujourd'hui, JavaScript peut aussi être utilisé pour programmer le **serveur web** (exécution de code JavaScript côté serveur)
- C'est une des motivations importantes qui a mené à la création de **node.js**
- C'est le rôle du module **http** de node.js

Serveur web

- Avec node.js un serveur web se code en peu de lignes :

```
var http = require("http");
```

```
var n = 0;
```

```
var obtenirDocument = function (requete) {
```

```
    var url = requete.url;
```

```
    print(++n + " " + url);
```

```
    return "url = " + url;
```

```
};
```

```
http.createServer(function (requete, reponse) {
```

```
    var doc = obtenirDocument(requete);
```

```
    reponse.writeHead(200, {"Content-Type": "text/html"});
```

```
    reponse.end(doc);
```

```
}).listen(8000);
```

serveur-web.js

sera envoyé au fureteur

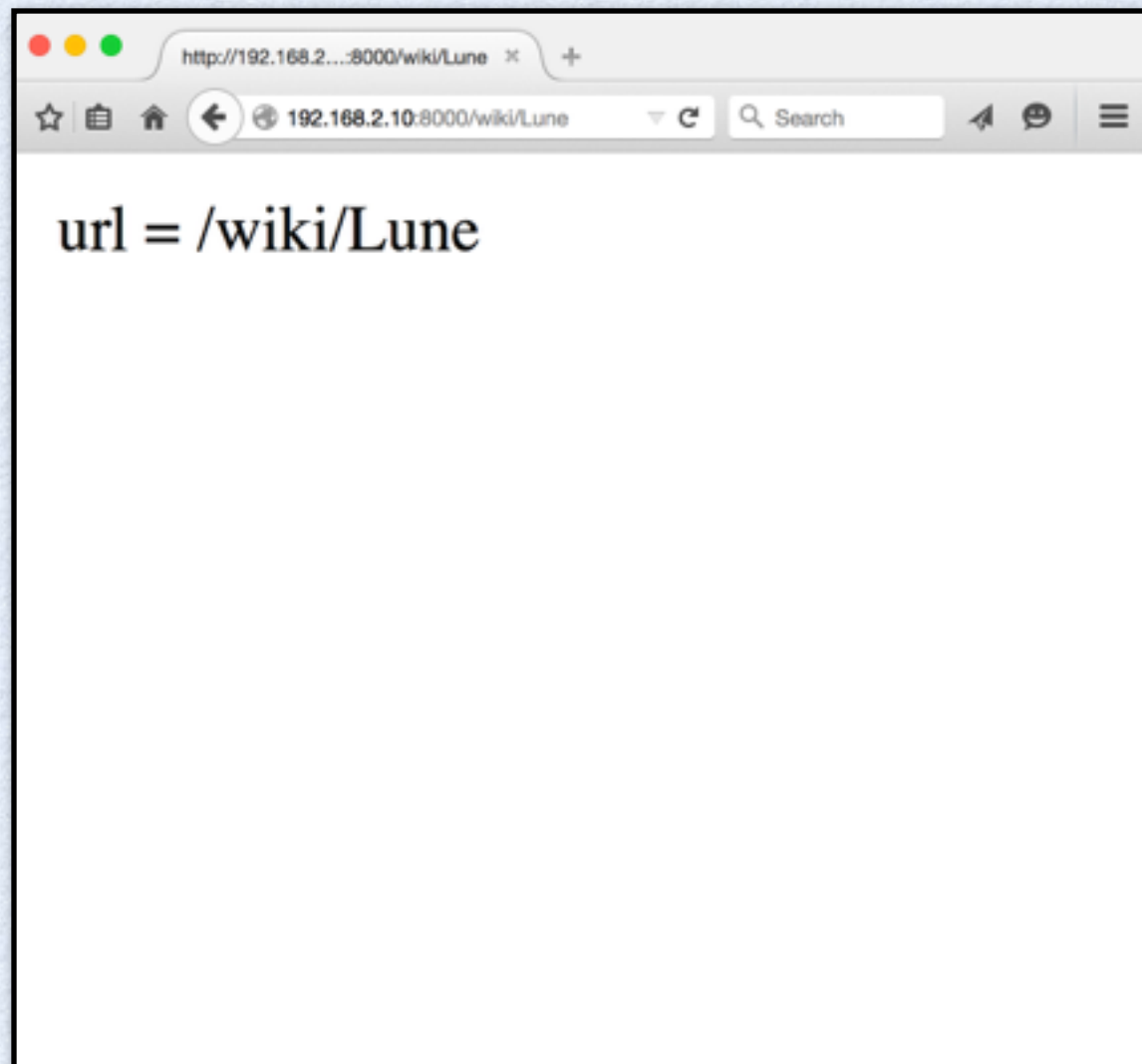
écouter sur le port 8000

- La fonction **obtenirDocument** se fait appeler à chaque fois qu'une requête est reçue par le serveur

Serveur web

- C'est une forme de **programmation par événements** où la réception d'une requête est un **événement** qui engendre le traitement approprié (envoi du doc.)

fureteur



GET /wiki/Lune

shell

```
% nodejs serveur-web.js  
1 /wiki/Lune  
2 /wiki/Lune
```

url = /wiki/Lune

c'est **obtenirDocument**
qui traite l'événement de
"réception de requête"

Serveur web

- C'est plus propre d'envoyer du HTML bien formé :

```
var n = 0;


var obtenirDocument = function (requete) {

    var url = requete.url;

    print(++n + " " + url);

    return "<!DOCTYPE html>\n" +
        "<html>\n" +
        " <head>\n" +
        "  <meta charset=\"UTF-8\">\n" +
        "  <title>" + url + "</title>\n" +
        " </head>\n" +
        " <body>\n" +
        "  <h1>requête #" + n + ": url = " + url + "</h1>\n" +
        " </body>\n" +
        "</html>\n";

};
```

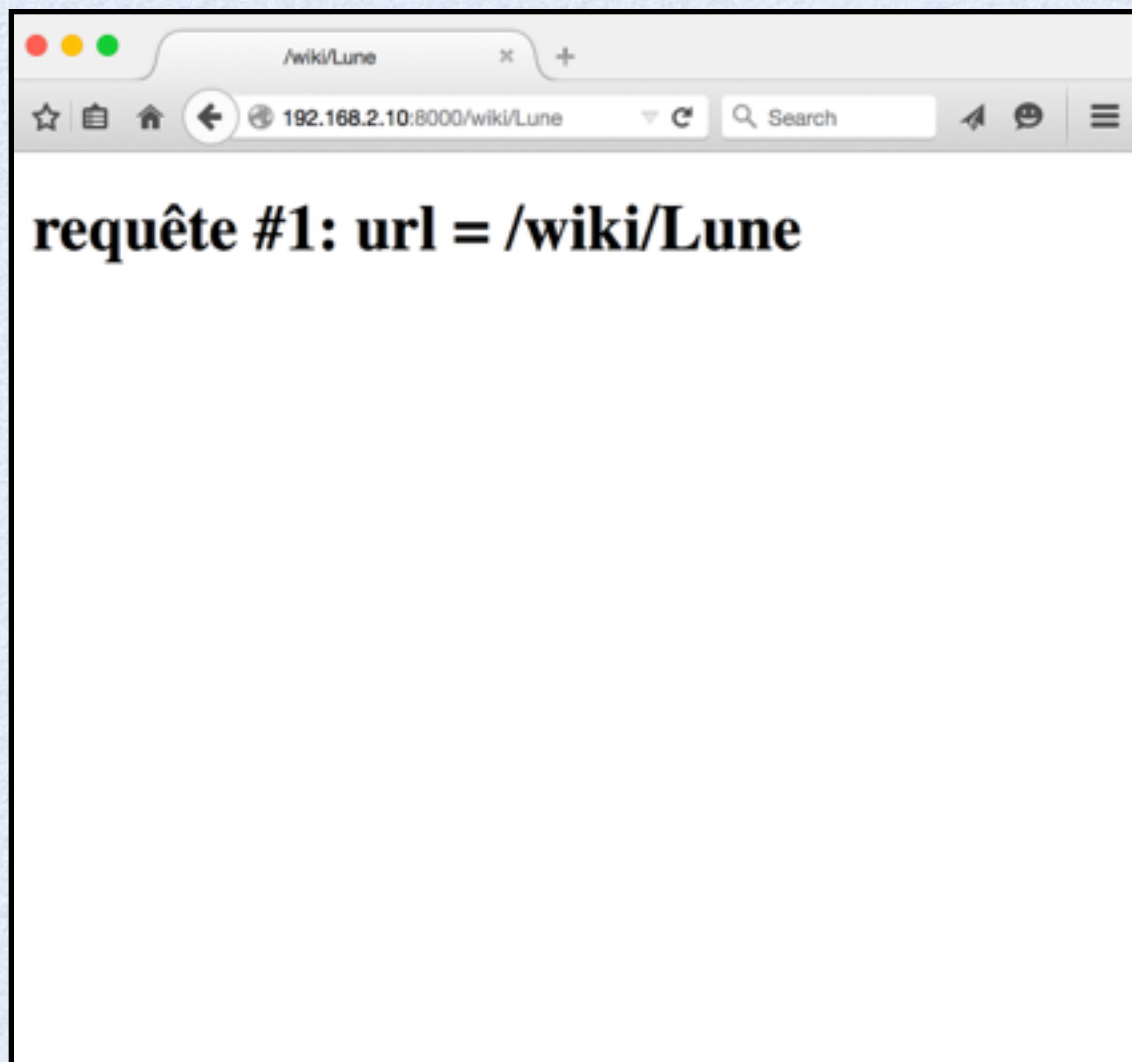


sera envoyé au fureteur

Serveur web

- Avec le changement à `obtenirDocument` :

fureteur



GET /wiki/Lune

shell

```
% nodejs serveur-web.js  
1 /wiki/Lune
```

```
<!DOCTYPE html>  
<html>  
  <head>  
    <meta charset="UTF-8">  
    <title>/wiki/Lune</title>  
  </head>  
  <body>  
    <h1>requête #1: url = /wiki/Lune</h1>  
  </body>  
</html>
```


Serveur web

- Normalement les documents maintenus par un serveur web sont stockés dans le **système de fichier**
- On peut les stocker dans un sous-répertoire **“documents”** et concaténer l’URL pour construire le chemin d’accès vers le fichier contenant le document :

```
var obtenirDocument = function (requete) {  
  
    var url = requete.url;  
  
    return readFile("documents" + url);  
};
```

Note : ce n’est pas très sécuritaire... il faudrait faire une validation de l’URL

création du chemin d’accès et lecture du document

Serveur web

- Utilisation du nouveau serveur web : **shell**

création de fichiers dans le sous répertoire **documents**, par exemple **documents/index.html** et **documents/test.html**

```
% mkdir documents
% emacs &
% nodejs serveur-web.js
```

8000

GET /index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Mon site</title>
  </head>
  <body>
    <h1>Mon site</h1>
    <p>Bienvenue à mon site web!</p>
  </body>
</html>
```

documents/index.html



CSS

CSS

- Les **propriétés d’affichage** (couleur, fonte, pos., ...) sont précisées avec **CSS** (“Cascading Style Sheet”)
- Les éléments peuvent posséder des **attributs** :
`<nom attr1="val1"...> ...contenu... </nom>`
- Attributs applicables à n’importe quel élément :
 - `style="style"` style CSS explicite
 - `class="classe"` classe du style CSS
 - `id="identifiant"` identification de l’élément

Éléments `<style>` et `<link>`

- L'information **CSS** est placée dans la tête, soit dans un élément `<style>` directement, ou dans un document séparé spécifié dans un élément `<link>` :

selecteurs
de balise

selecteur
de classe

selecteur
d'identifiant

```
...  
<head>  
  <meta charset="UTF-8">  
  <title>Test5</title>  
  
  <style>  
    a { color: violet; font-family: Courier; }  
    p { color: orange; font-size: 20px; }  
    .important { color: red; }  
    #conclusion { font-size: 8px; }  
  </style>  
  
  <link rel="stylesheet" href="monstyle.css">  
  
</head>  
...
```


Exemple de CSS

test5.html

résultat

```
<!DOCTYPE html>

<html>

  <head>
    <meta charset="UTF-8">
    <title>Test5</title>

    <style>
      a { color: violet; font-family: Courier; }
      p { color: orange; font-size: 20px; }
      .important { color: red; }
      #conclusion { font-size: 8px; }
    </style>

  </head>

  <body>

    <p id="intro"> Ceci est l'introduction. </p>

    <p> Deuxième paragraphe. </p>

    <p style="color: green;"> Troisième paragraphe. </p>

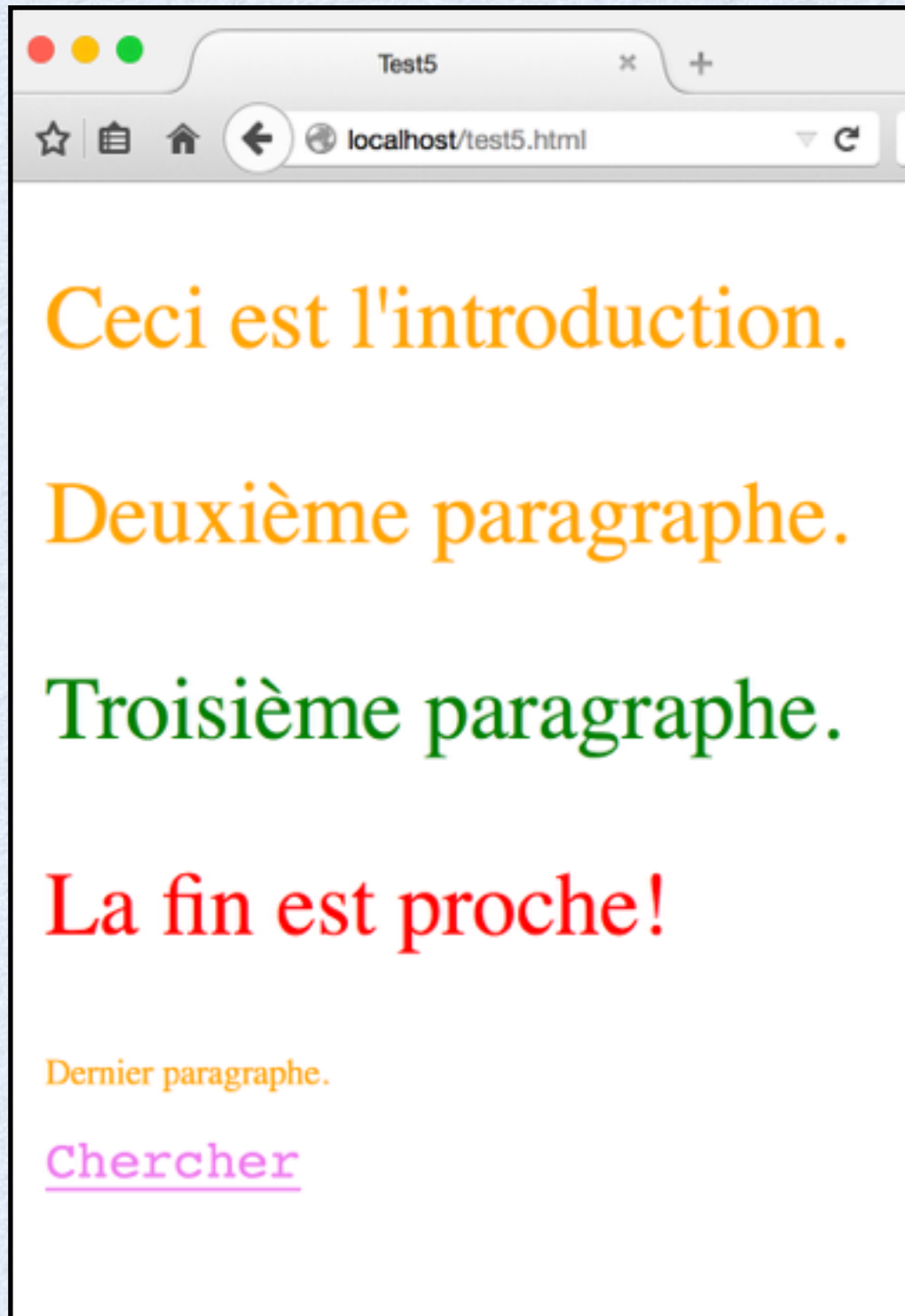
    <p class="important"> La fin est proche! </p>

    <p id="conclusion"> Dernier paragraphe. </p>

    <a href="http://google.com">Chercher</a>

  </body>

</html>
```



Traitement d'événements

Traitement d'événements

- Les éléments du documents peuvent réagir à des **événements** les concernant (cliquer sur l'élément, survoler l'élément avec la souris, etc)
- On peut spécifier du **code JavaScript à exécuter** lorsqu'un événement spécifique survient
- C'est le *traiteur d'événement* ("event handler")
- Exemple :

```
<button onclick="alert('clic');">OK</button>
```

traiteur d'événement clic de souris sur ce bouton

Attributs d'événements

- Traiteurs d'événements courants :
 - `onclick="code"` cliquer sur l'élément
 - `onmouseover="code"` débiter le survol de l'élément
 - `onmouseout="code"` cesser le survol de l'élément
 - `onkeydown="code"` presser une touche de clavier
 - `onkeyup="code"` relâcher une touche de clavier
 - `onload="code"` l'élément est tout chargé
- Exemple :

```
<body onkeydown="alert(event.which) ;"></body>
```

 *code du caractère*

La balise `<script>`

- La balise `<script>` placée dans la tête permet d'inclure du code JavaScript dans le document :
test6.html

```
<!DOCTYPE html>

<html>

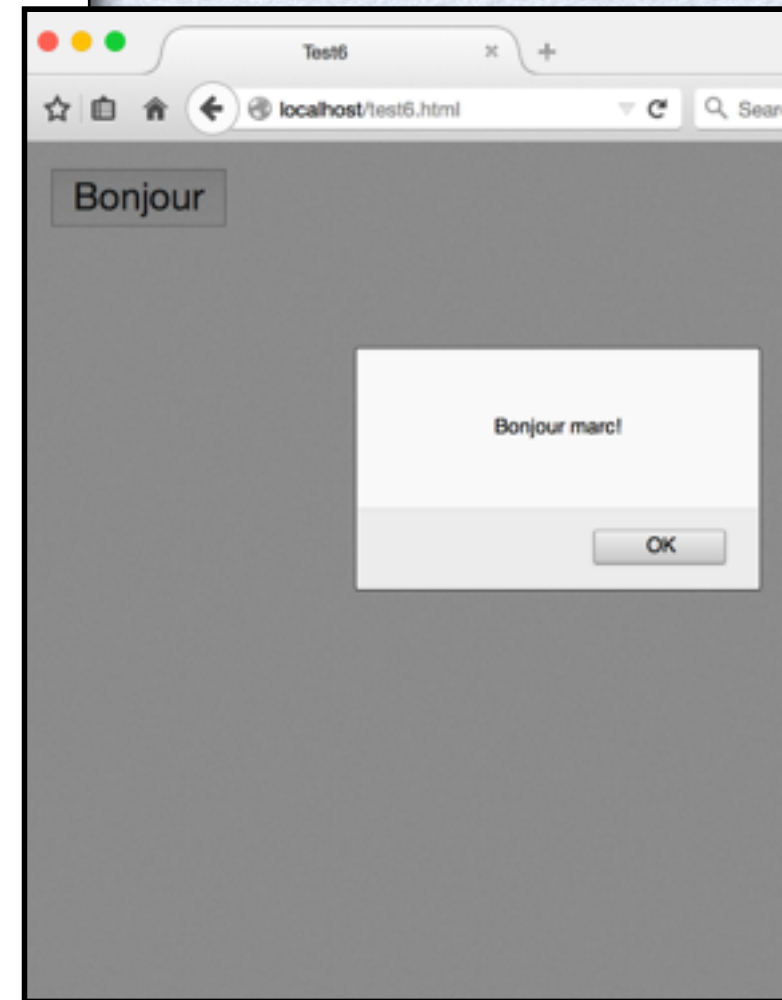
  <head>
    <meta charset="UTF-8">
    <title>Test6</title>

    <script>
      var nom = "**pas de nom**";
      var clic = function () { alert("Bonjour "+nom+"!"); };
      var init = function () { nom = prompt("Votre nom?"); };
    </script>

  </head>

  <body onload="init();" >
    <button onclick="clic();" >Bonjour</button>
  </body>

</html>
```



La balise `<script>`

- La balise `<script>` peut aussi référer à un URL contenant le code JavaScript, ce qui est plus propre :

test7.html

```
<!DOCTYPE html>

<html>

  <head>
    <meta charset="UTF-8">
    <title>Test7</title>

    <script src="test7.js"></script>

  </head>

  <body onload="init();" >
    <button onclick="clic();" >Bonjour</button>
  </body>

</html>
```

test7.js

```
var nom = "***pas de nom**";

var clic = function () {
  alert("Bonjour "+nom+"!");
};

var init = function () {
  nom = prompt("Votre nom?");
};
```


DOM

DOM

- L'attrait principal de JavaScript dans le navigateur c'est que le document est **modifiable** par des méthodes JavaScript après le chargement du document
- On peut donc faire en sorte que le document **change** en réaction aux événements (souris, clavier, etc)
- La variable globale JavaScript **document** contient une référence vers l'objet qui représente le document
- Cet objet représente le document **hiérarchiquement** avec le **DOM** ("Document Object Model") qui définit les méthodes d'accès

DOM

- Chaque **élément** du document est représenté par un **objet** sur lequel on peut appeler des méthodes
- **document.getElementById("id")** retourne l'objet correspondant à l'élément avec l'identifiant *id*
- *élément*.**innerHTML** est le texte correspondant au contenu de l'élément *élément*, c'est-à-dire le texte entre les balises de l'élément dans le document
- *élément*.**value** est le texte correspondant à la valeur de l'élément *élément* (particulièrement utile pour les éléments **<textarea>**, **<select>** et **<input>**)
- **innerHTML** et **value** sont modifiables

Exemple innerHTML

test8.html

après 3 clics

```
<!DOCTYPE html>
<html>

<head> <meta charset="UTF-8"> <title>Test8</title>

  <script>

    var b; // l'élément "body"

    var init = function () {
      b = document.getElementById("b");
      b.innerHTML = "bonjour";
    };

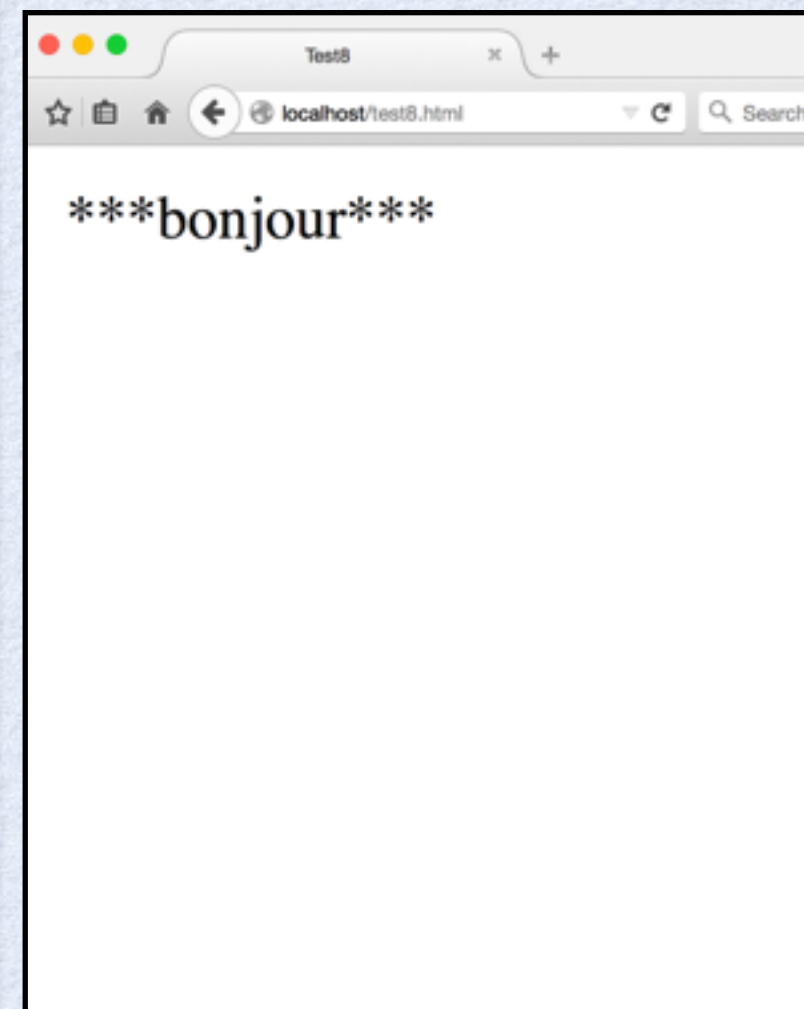
    var clic = function () {
      b.innerHTML = "*" + b.innerHTML + "*";
    };

  </script>

</head>

<body id="b" onload="init();" onclick="clic();"></body>

</html>
```



Exemple value

test9.html

```
<!DOCTYPE html>
<html>
  <head> <meta charset="UTF-8"> <title>Test9</title>
    <script>

      var clic = function () {
        var nom = document.getElementById("nom").value;
        if (document.getElementById("sexe").value == "H") {
          alert("Bonjour Mr. " + nom);
        } else {
          alert("Bonjour Mme. " + nom);
        }
      };

    </script>
  </head>

  <body>

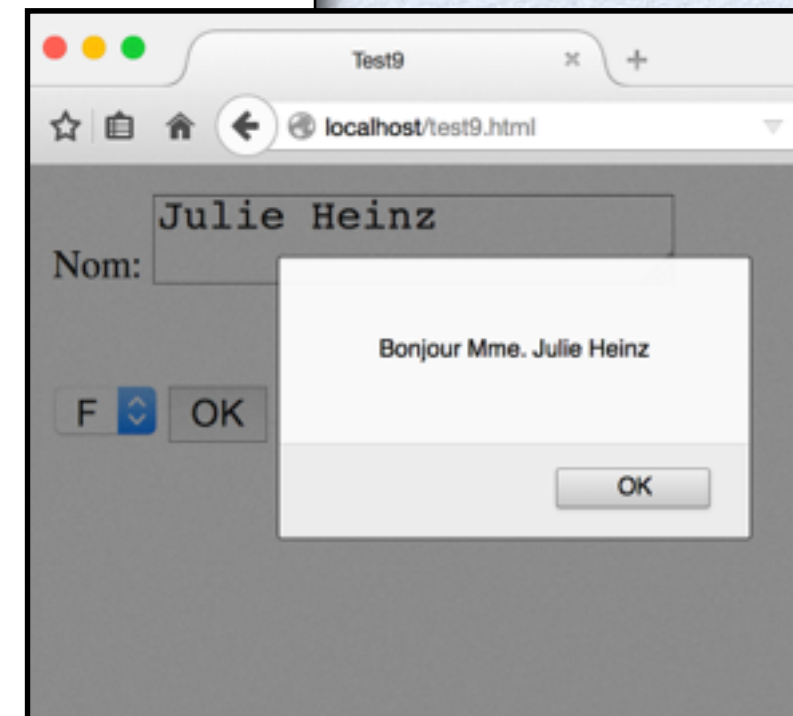
    Nom: <textarea id="nom"></textarea> <br><br><br>

    <select id="sexe"> <option>H</option> <option>F</option> </select>

    <button onclick="clic();">OK</button>

  </body>
</html>
```

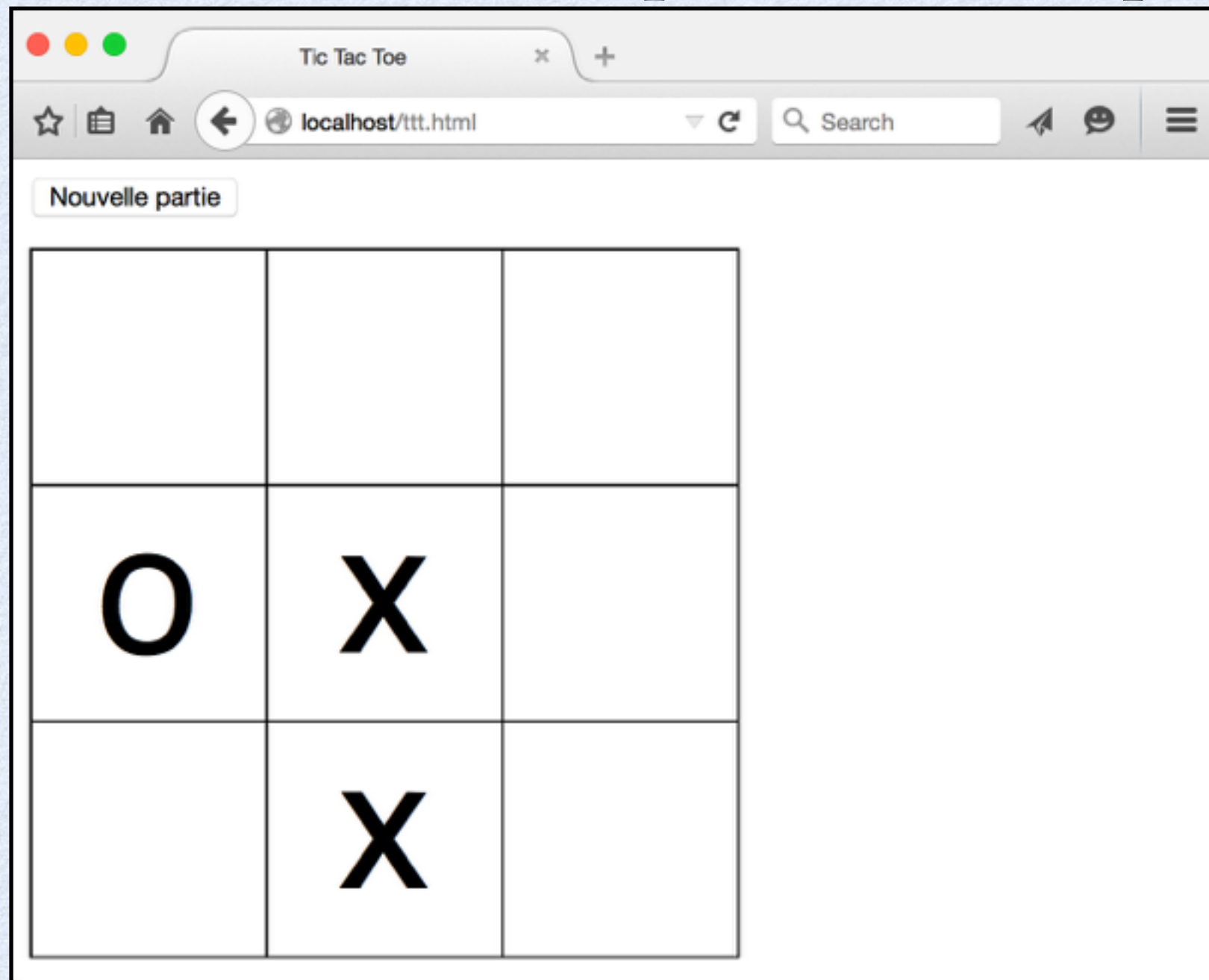
après avoir
cliqué OK



Exemple complet : Tic Tac Toe

Exemple complet : Tic Tac Toe

- **Spécification** : jeu de tic tac toe entre 2 joueurs sur un même ordinateur, cliquer une case pour y jouer :



Tables

- Pour créer des tables, HTML a les balises suivantes :

`<table>`

début de
rangée 1

`<tr>`

`<td>un</td>`

`<td>deux</td>`

`<td>trois</td>`

`</tr>`

début de
rangée 2

`<tr>`

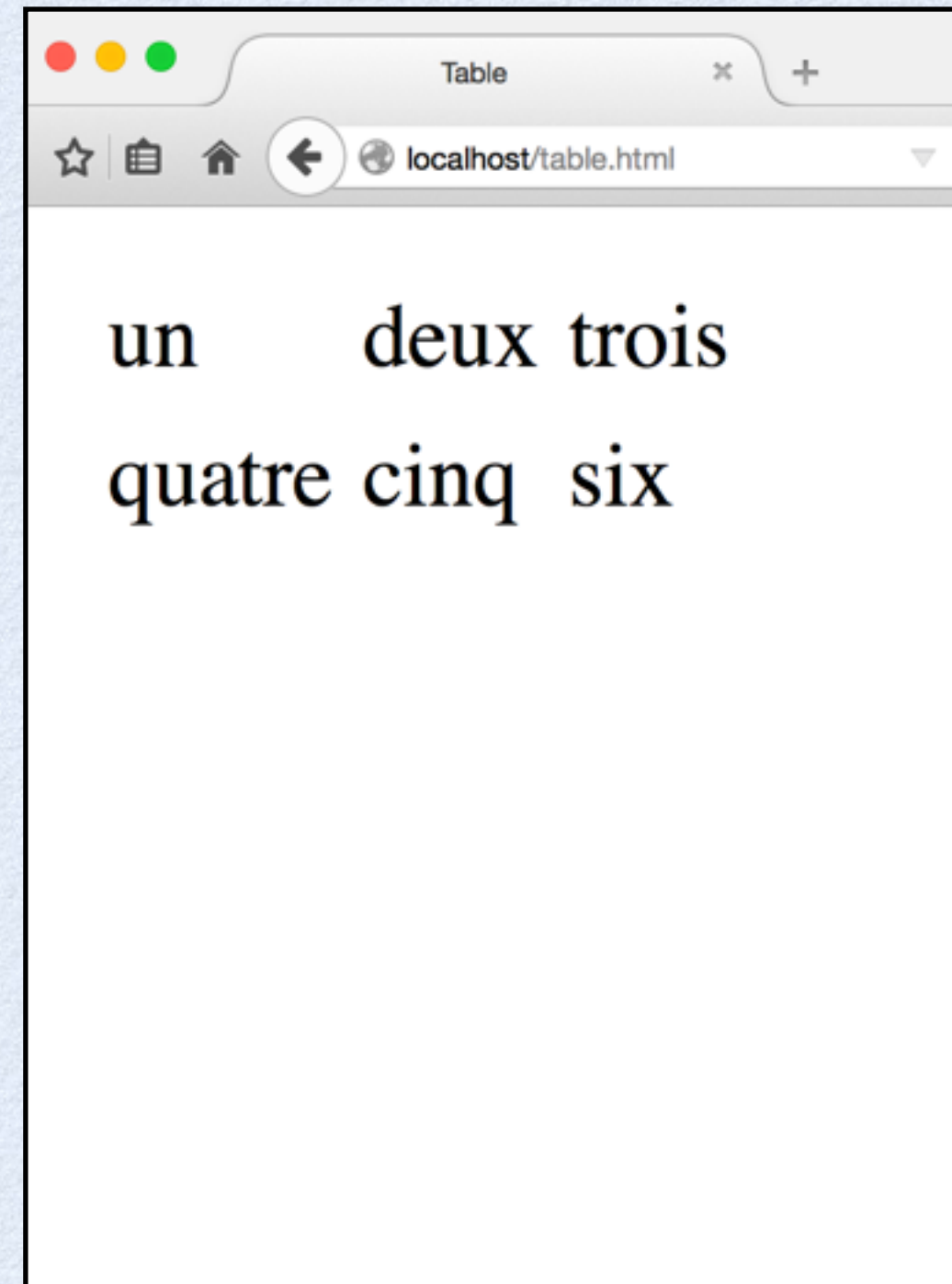
`<td>quatre</td>`

`<td>cinq</td>`

`<td>six</td>`

`</tr>`

`</table>`



ttt.css

- Pour le jeu, il faut utiliser des attributs CSS pour avoir une grille, fixer la taille des cases, la fonte, etc

```
table {  
    table-layout: fixed;  
    border-collapse: collapse;  
}
```

border-collapse: collapse	
Grey table border	Black cell border

border-collapse: separate	
Grey table border	Black cell border

```
td {  
    width: 100px;  
    height: 100px;  
    border: 1px solid black;  
    font-size: 80px;  
    font-family: Helvetica;  
    text-align: center;  
    vertical-align: center;  
}
```

taille des
cases

police

alignement
au centre
des cases

ttt.html

```
<!DOCTYPE html>
<html>
  <head> <meta charset="UTF-8"> <title>Tic Tac Toe</title>
    <script src="ttt.js"></script>
    <link rel="stylesheet" href="ttt.css">
  </head>

  <body onload="init();"

    <button onclick="init();">Nouvelle partie</button> <br><br>

    <table>
      <tr>
        <td id="0" onclick="clie(0);"></td>
        <td id="1" onclick="clie(1);"></td>
        <td id="2" onclick="clie(2);"></td>
      </tr>
      <tr>
        <td id="3" onclick="clie(3);"></td>
        <td id="4" onclick="clie(4);"></td>
        <td id="5" onclick="clie(5);"></td>
      </tr>
      <tr>
        <td id="6" onclick="clie(6);"></td>
        <td id="7" onclick="clie(7);"></td>
        <td id="8" onclick="clie(8);"></td>
      </tr>
    </table>

  </body>
</html>
```

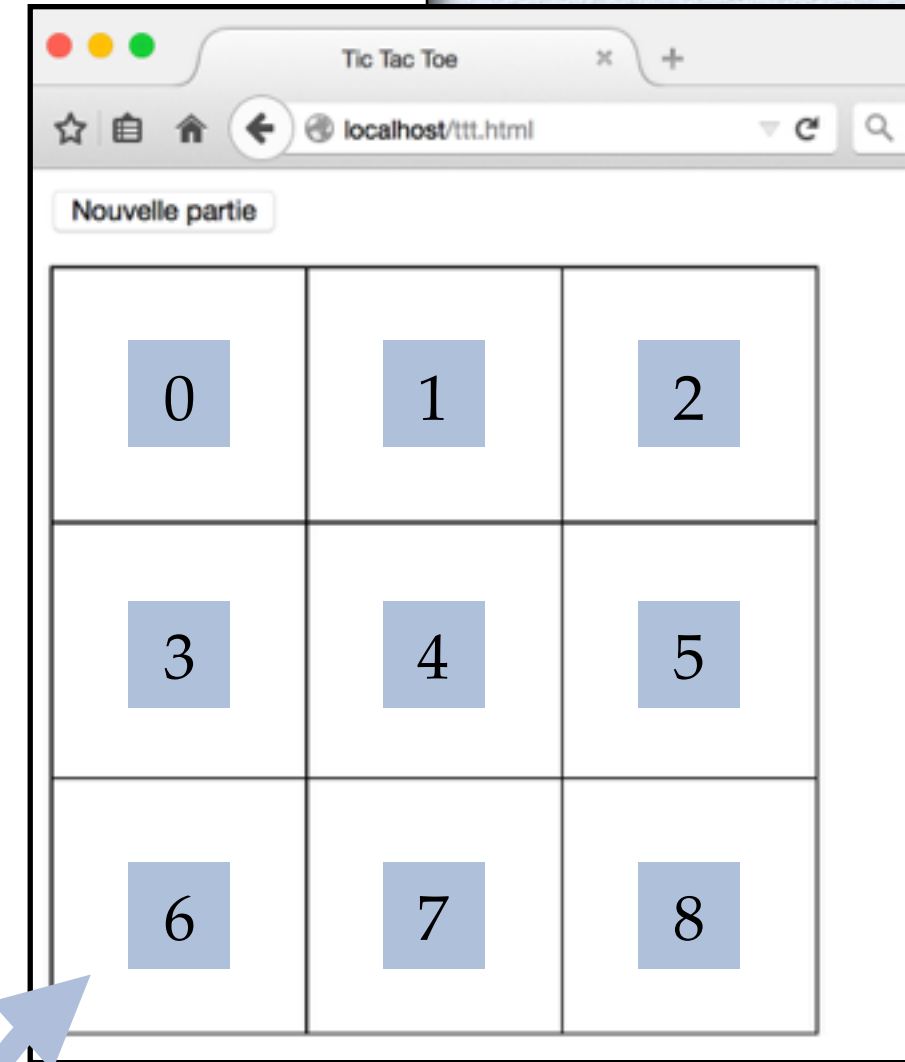


table 3x3 où chaque case est vide et a un identifiant de 0 à 8 et un traiteur de clic

ttt.js

```
var tour; // vaut "x" ou "o" en fonction du tour

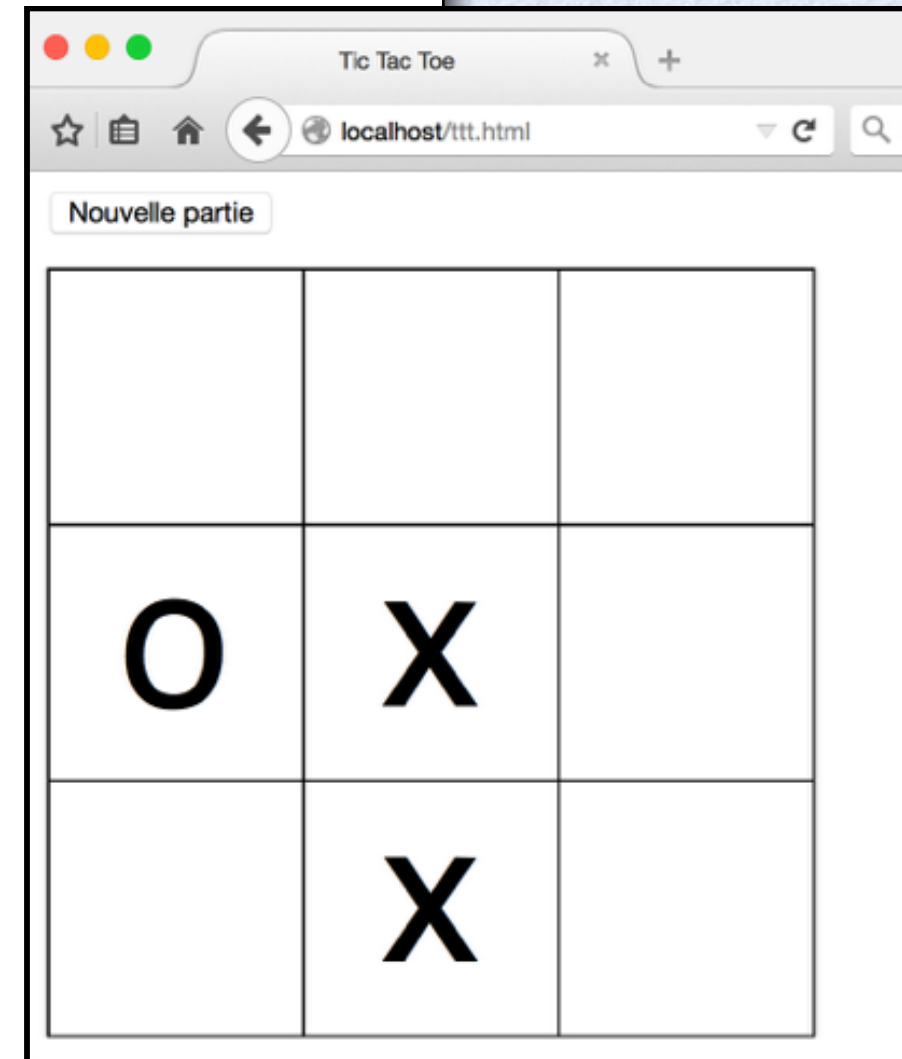
var clic = function (id) {

    var elem = document.getElementById(id);

    elem.innerHTML = tour;

    if (tour == "x") {
        tour = "o";
    } else {
        tour = "x";
    }
};

var init = function () {
    tour = "x";
    for (var i=0; i<9; i++) {
        var elem = document.getElementById(i);
        elem.innerHTML = "";
    }
};
```



Comportements à régler

- On peut jouer où il y a déjà un “x” ou “o”
- Aucune détection de 3 “x” ou “o” alignés
- Aucune détection de partie nulle (9 cases occupées sans 3 “x” ou “o” alignés)

Validation de la case

```
var tour; // vaut "x" ou "o" en fonction du tour

var clic = function (id) {

    var elem = document.getElementById(id);

    if (elem.innerHTML == "") {

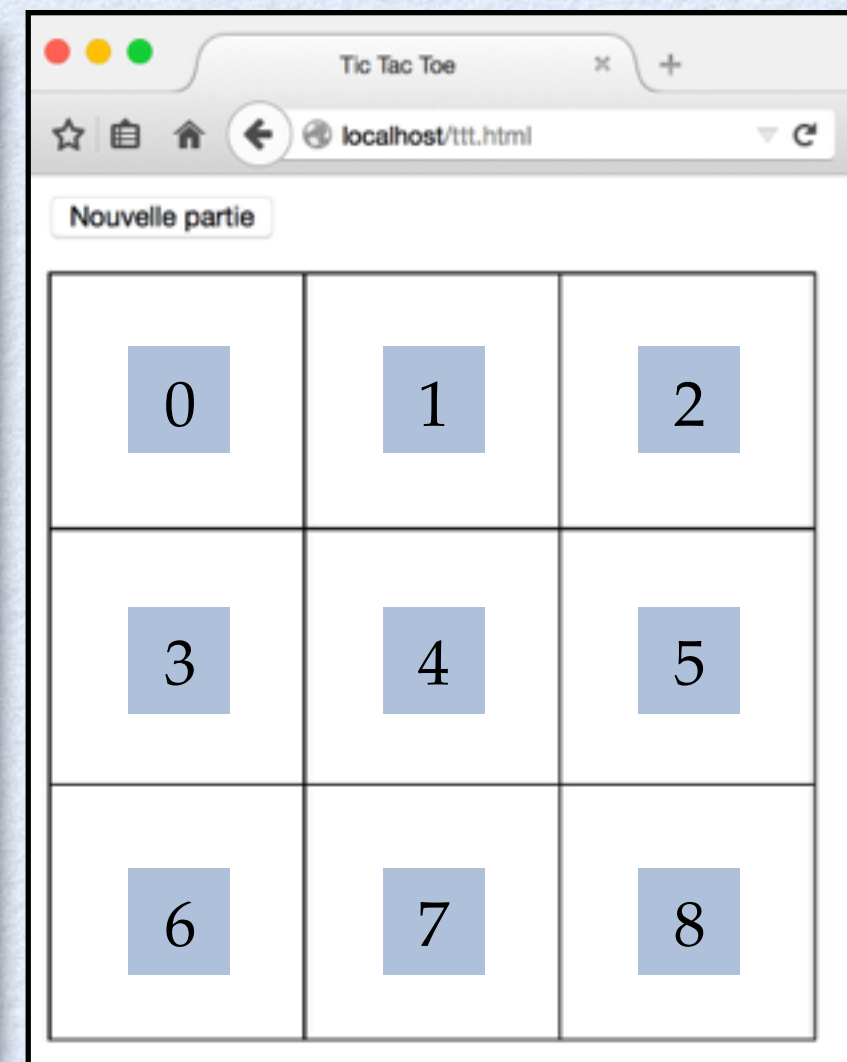
        elem.innerHTML = tour;

        if (tour == "x") {
            tour = "o";
        } else {
            tour = "x";
        }
    }
};
```


Détection d'une victoire

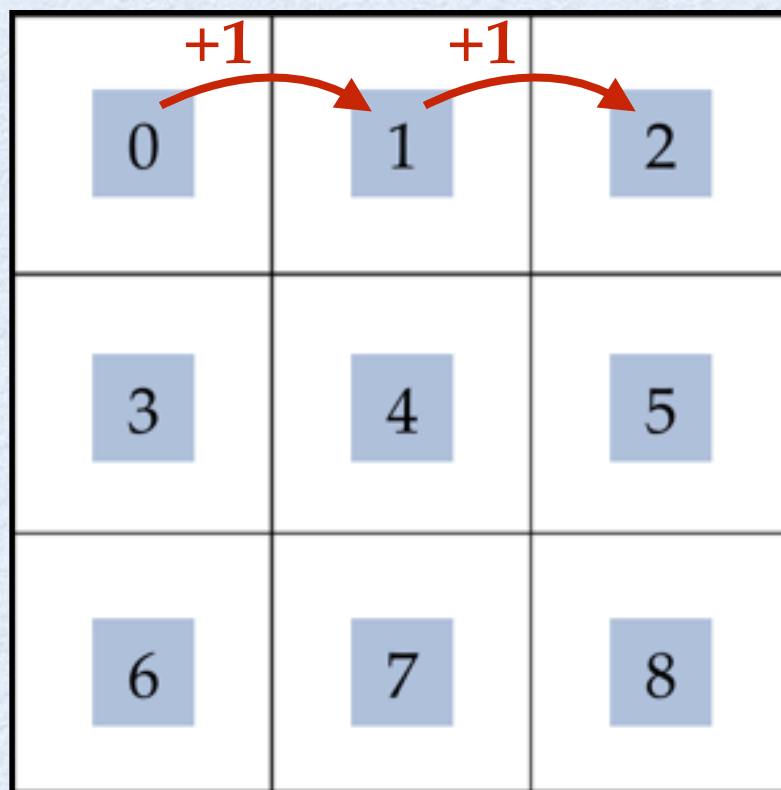
- Il faut vérifier si sur une des 3 rangées, ou une des trois colonnes, ou une des 2 diagonales ont retrouvé 3 fois le même symbole "x" ou "o"
- On pourrait utiliser la fonction **victoire** suivante :

```
var sym = function (id) {  
    return document.getElementById(id) ;  
};  
  
var victoire = function () {  
  
    if (sym(0) !== "" && sym(0) == sym(1) && sym(0) == sym(2))  
        return sym(0) ;  
  
    if (sym(3) !== "" && sym(3) == sym(4) && sym(3) == sym(5))  
        return sym(3) ;  
  
    if (sym(6) !== "" && sym(6) == sym(7) && sym(6) == sym(8))  
        return sym(6) ;  
  
    ... // 5 autres cas similaires à vérifier  
  
    return "" ;  
};
```



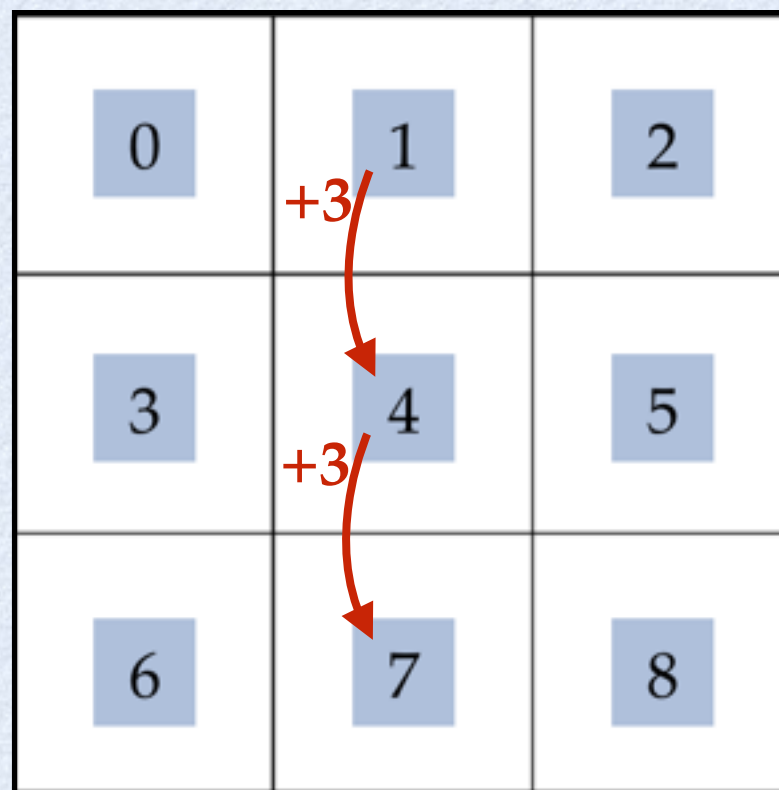
Détection d'une victoire

- Ça manque de généralité et contient beaucoup de redondance (répétition de la même logique)
- Chaque alignement de trois cases a un **point de départ** et un **pas**, par exemple :



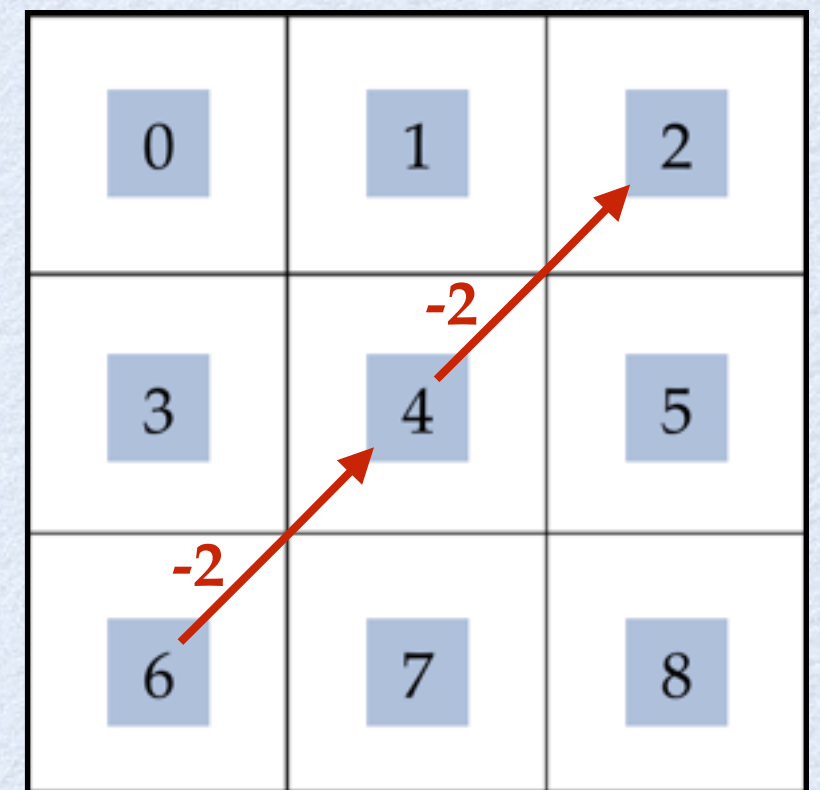
A 3x3 grid with cells containing numbers 0 to 8. Red curved arrows indicate a horizontal sequence starting at 0, moving to 1 (+1), and then to 2 (+1).

0	1	2
3	4	5
6	7	8



A 3x3 grid with cells containing numbers 0 to 8. Red curved arrows indicate a vertical sequence starting at 1, moving to 4 (+3), and then to 7 (+3).

0	1	2
3	4	5
6	7	8



A 3x3 grid with cells containing numbers 0 to 8. Red straight arrows indicate a diagonal sequence starting at 6, moving to 4 (-2), and then to 2 (-2).

0	1	2
3	4	5
6	7	8

Détection d'une victoire

- Idée : structure pour représenter les alignements

```
var alignements = [{pas: +1, departs: [0,3,6]},  
                   {pas: +3, departs: [0,1,2]},  
                   {pas: +4, departs: [0]},  
                   {pas: -2, departs: [6]}];  
  
var victoire = function () {  
  
    for (var i=0; i<alignements.length; i++) {  
  
        var pas = alignements[i].pas;  
        var departs = alignements[i].departs;  
  
        for (var j=0; j<departs.length; j++) {  
            var pos = departs[j];  
            if (sym(pos) != "") {  
                for (var k=1; k<3; k++) {  
                    if (sym(pos) != sym(pos+k*pas))  
                        break;  
                }  
                if (k == 3) return sym(pos);  
            }  
        }  
    }  
  
    return "";  
};
```


Programme final (partie 1)

```
var tour; // vaut "x" ou "o" en fonction du tour
var libres; // nombre de cases libres

var sym = function (id) {
    return document.getElementById(id).innerHTML;
};

var alignements = [{pas: +1, departs: [0,3,6]},
                   {pas: +3, departs: [0,1,2]},
                   {pas: +4, departs: [0]},
                   {pas: -2, departs: [6]}];

var victoire = function () {

    for (var i=0; i<alignements.length; i++) {

        var pas = alignements[i].pas;
        var departs = alignements[i].departs;

        for (var j=0; j<departs.length; j++) {
            var pos = departs[j];
            if (sym(pos) != "") {
                for (var k=1; k<3; k++) {
                    if (sym(pos) != sym(pos+k*pas))
                        break;
                }
                if (k == 3) return sym(pos);
            }
        }
    }

    return "";
};
```


Programme final (partie 2)

```
var clic = function (id) {  
  
    var elem = document.getElementById(id);  
  
    if (elem.innerHTML == "") {  
  
        elem.innerHTML = tour;  
  
        var v = victoire();  
        if (v != "") {  
            alert(v + " est le gagnant!");  
            init();  
        } else {  
            if (tour == "x") tour = "o"; else tour = "x";  
            if (--libres == 0) {  
                alert("match nul!");  
                init();  
            }  
        }  
    }  
};  
  
var init = function () {  
    tour = "x";  
    libres = 9;  
    for (var i=0; i<9; i++) {  
        var elem = document.getElementById(i);  
        elem.innerHTML = "";  
    }  
};
```


Améliorations

- Meilleur codage (découplage de la logique et de l'affichage)
- Remplacer les x et o par des images
- Bouton pour reprendre le dernier coup
- Bouton pour refaire le coup repris
- Garder un score du nombre de parties gagnées par chaque joueur
- Permettre de jouer contre l'ordinateur
- Permettre de jouer contre un humain à distance

Découplage logique/affichage

```
var tour; // entier indiquant à qui c'est le tour, 1 pour "x", 2 pour "o"
var grille; // contenu de la grille de jeu, un 0 indique une case vide
var libres; // nombre de cases libres

var symboleJoueur = function (joueur) { return (joueur==1) ? "x" : "o"; };

var autreJoueur = function (joueur) { return 3-joueur; };

var clic = function (id) {

    if (grille[id] == 0) {

        grille[id] = tour;
        document.getElementById(id).innerHTML = symboleJoueur(tour);

        var gagnant = victoire();
        if (gagnant != 0) {
            alert(symboleJoueur(gagnant) + " est le gagnant!");
            init();
        } else {
            tour = autreJoueur(tour);
            if (--libres == 0) {
                alert("match nul!");
                init();
            }
        }
    }
};

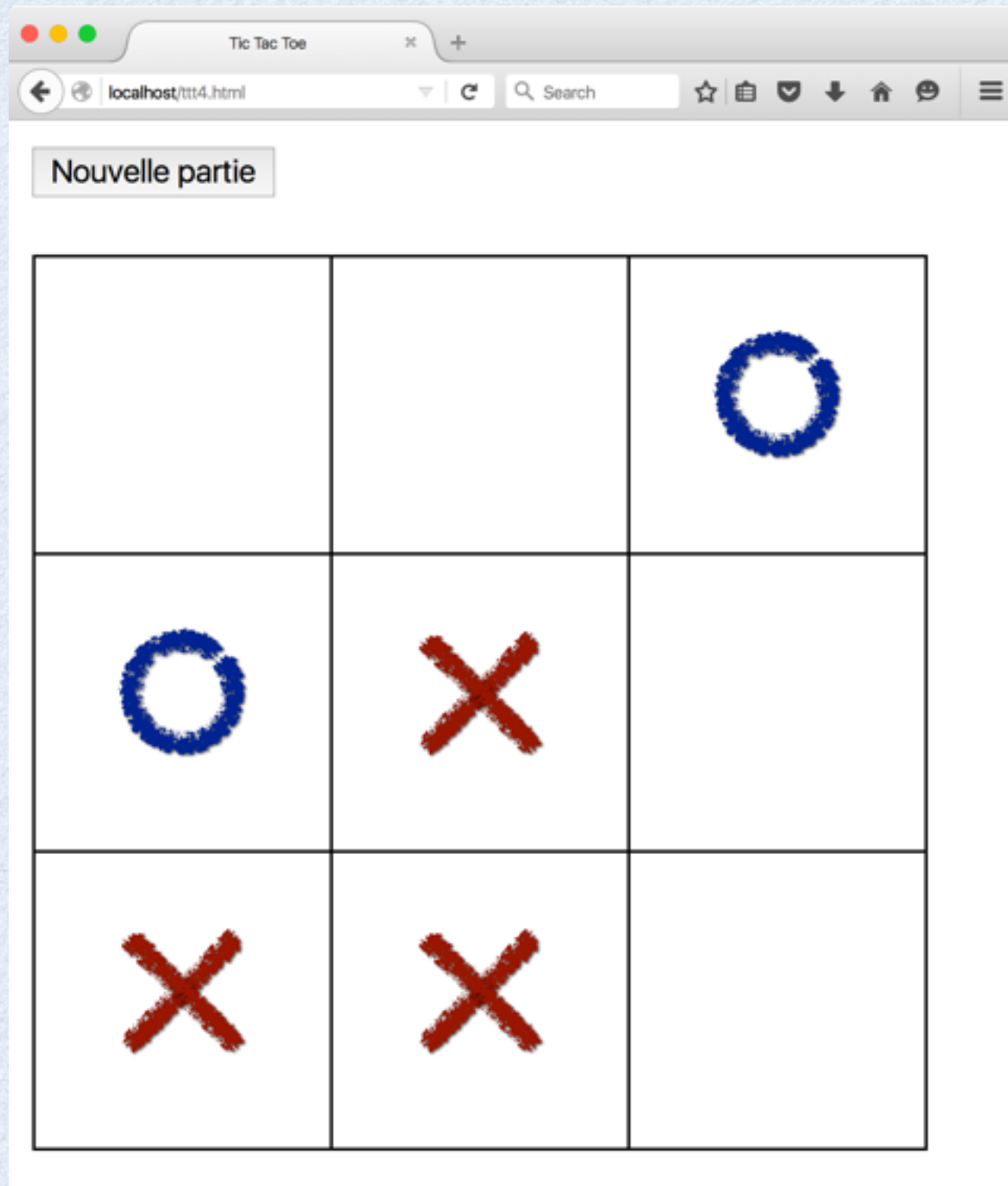
var init = function () {
    tour = 1;
    grille = Array(9).fill(0);
    ...
};
```

l'utilisation d'un encodage numérique pour le joueur permet de **découpler** le choix des symboles (x ou o) de la logique du jeu

Découplage logique/affichage

```
var alignements = [{pas: +1, departs: [0,3,6]},  
                   {pas: +3, departs: [0,1,2]},  
                   {pas: +4, departs: [0]},  
                   {pas: -2, departs: [6]}];  
  
var victoire = function () {  
  
    for (var i=0; i<alignements.length; i++) {  
  
        var pas = alignements[i].pas;  
        var departs = alignements[i].departs;  
  
        for (var j=0; j<departs.length; j++) {  
            var pos = departs[j];  
            if (grille[pos] != 0) {  
                for (var k=1; k<3; k++) {  
                    if (grille[pos] != grille[pos+k*pas])  
                        break;  
                }  
                if (k == 3) return grille[pos];  
            }  
        }  
    }  
  
    return 0;  
};
```


Utiliser des images



joueur1.png



joueur2.png



Utiliser des images

```
var symboleJoueur = function (joueur) {  
    return (joueur == 1)  
        ? "<img src=\"joueur1.png\" width=50>"  
        : "<img src=\"joueur2.png\" width=50>";  
};  
  
var nomJoueur = function (joueur) {  
    return (joueur == 1) ? "x" : "o";  
};
```

```
var symboleJoueur = function (joueur) {  
    return "<img src=\"joueur\" + joueur + ".png\" width=50>";  
};
```


Boutons pour reprendre/refaire un coup

```
<!DOCTYPE html>
<html>
  <head> <meta charset="UTF-8"> <title>Tic Tac Toe</title>
    <script src="ttt.js"></script>
    <link rel="stylesheet" href="ttt.css">
  </head>

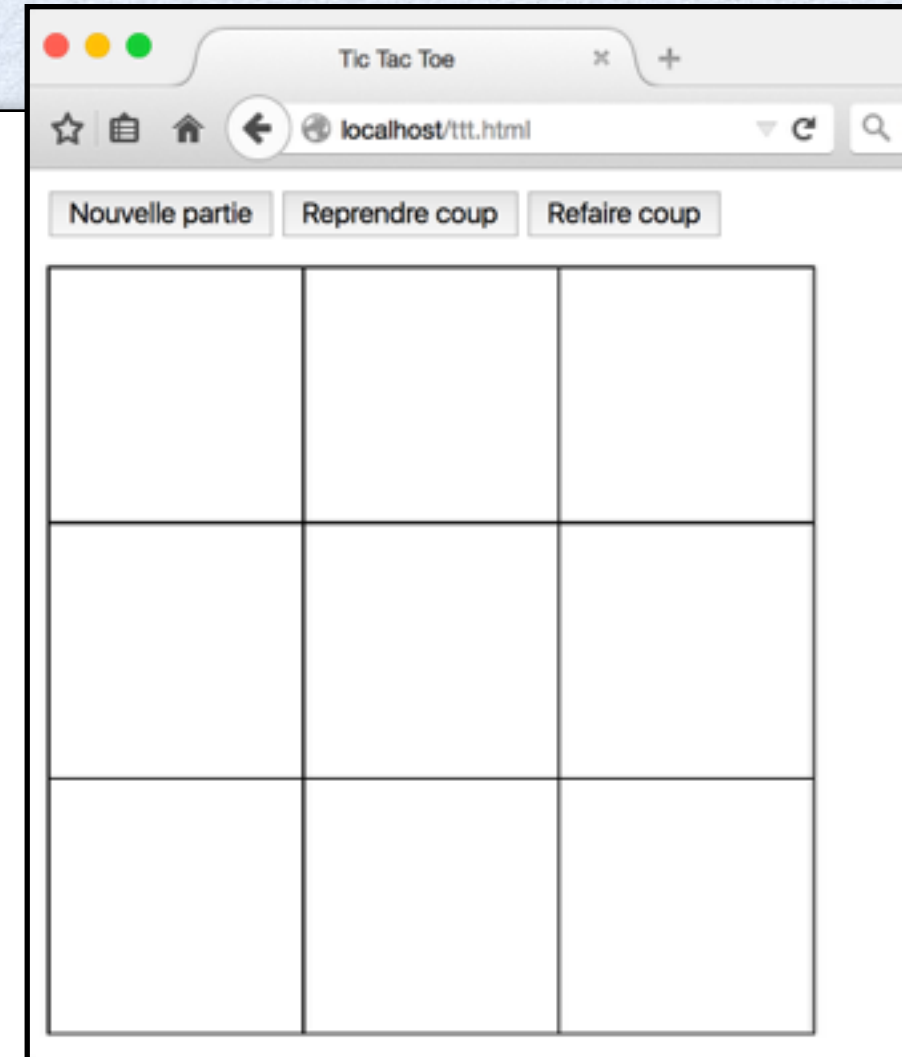
  <body onload="init();"

    <button onclick="init();">Nouvelle partie</button>
    <button id="undo" onclick="undo();">Reprendre coup</button>
    <button id="redo" onclick="redo();">Refaire coup</button>

    <br><br>

    <table>
      ...
    </table>

  </body>
</html>
```



Code

```
var tour; // entier indiquant à qui c'est le tour, 1 pour "x", 2 pour "o"
var grille; // contenu de la grille de jeu, un 0 indique une case vide
var occupees; // nombre de cases occupées
var coups; // position des coups joués

var init = function () {
    tour = 1;
    grille = Array(9).fill(0);
    occupees = 0;
    coups = [];
    for (var i=0; i<9; i++) {
        var elem = document.getElementById(i);
        elem.innerHTML = "";
    }
    document.getElementById("undo").style.visibility = "hidden";
    document.getElementById("redo").style.visibility = "hidden";
};
```


Code

```
var clic = function (id) {  
  
    if (grille[id] == 0) {  
  
        grille[id] = tour;  
        document.getElementById(id).innerHTML = symboleJoueur(tour);  
  
        coups = coups.slice(0, occupees);  
        coups.push(id);  
  
        document.getElementById("undo").style.visibility = "visible";  
        document.getElementById("redo").style.visibility = "hidden";  
  
        var gagnant = victoire();  
        if (gagnant != 0) {  
            alert(nomJoueur(gagnant) + " est le gagnant!");  
            init();  
        } else {  
            tour = autreJoueur(tour);  
            if (++occupees == 9) {  
                alert("match nul!");  
                init();  
            }  
        }  
    }  
};
```


Code

```
var undo = function () {
    if (occupees > 0) {
        occupees--;
        grille[coups[occupees]] = 0;
        document.getElementById(coups[occupees]).innerHTML = "";
        tour = autreJoueur(tour);
        if (occupees == 0) {
            document.getElementById("undo").style.visibility = "hidden";
        }
        document.getElementById("redo").style.visibility = "visible";
    }
};

var redo = function () {
    if (occupees < coups.length) {
        grille[coups[occupees]] = tour;
        document.getElementById(coups[occupees]).innerHTML = symboleJoueur(tour);
        occupees++;
        tour = autreJoueur(tour);
        if (occupees == coups.length) {
            document.getElementById("redo").style.visibility = "hidden";
        }
        document.getElementById("undo").style.visibility = "visible";
    }
};
```