Programmation Web Avancée
Cours 1

Yacine Bouzidi[⋆]

[⋆] yacine.bouzidi@inria.fr

IG2I, Centrale Lille.

Objectif du cours

- Initiation à la programmation web côté client
- Conception de sites web efficaces
- Acquisition de bonnes pratiques + culture du web
- Conception d'applications internet riches
- Complémentaire à l'UE application web côté serveur

Organisation

Enseignement :

- Deux heures de cours par semaine
- Deux heures de TP par semaine

Le cours présente différents concepts dans un cadre général, les TPs permettent une application plus ou moins directe.

Evaluation :

- Rendu de code à chaque TP
- Tp noté (2h)
- Projet en groupe

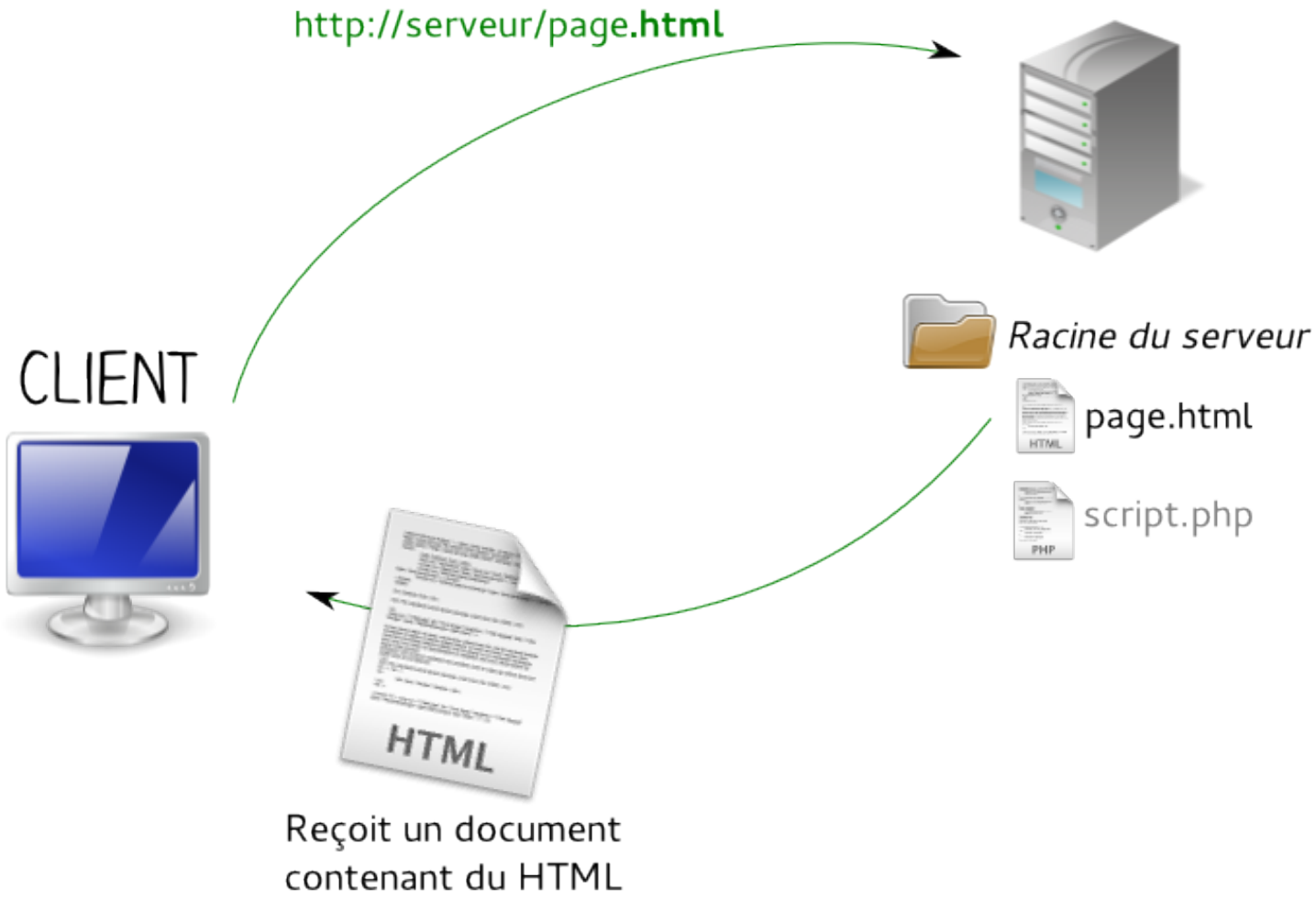
Plan de la séance

Overview

Le Web en quelques mots

- (au commencement) Protocole d'échange de documents (les pages Web)
- Le format de fichier est HTML (on va revenir dessus), spécifié par le W3C
- Les fichiers sont stockés sur des serveurs Web
- Des clients (les navigateurs Web) se connectent au serveur en utilisant le protocole HTTP (protocole d'application au dessus de TCP/IP).
- Les ressources sont identifiées par des URLs (des chaînes de caractères au format proto://machine:port/chemin/vers/la/ressource).
- Les documents HTML contiennent des liens hypertexte qui permettent de naviguer de pages en pages via des URLs. Les ressources d'une page (images, scripts, feuilles de style, . . .) sont aussi dénotées par des URLs.
- Le navigateur Web récupère les ressources et assure le rendu (souvent graphique) de la page.

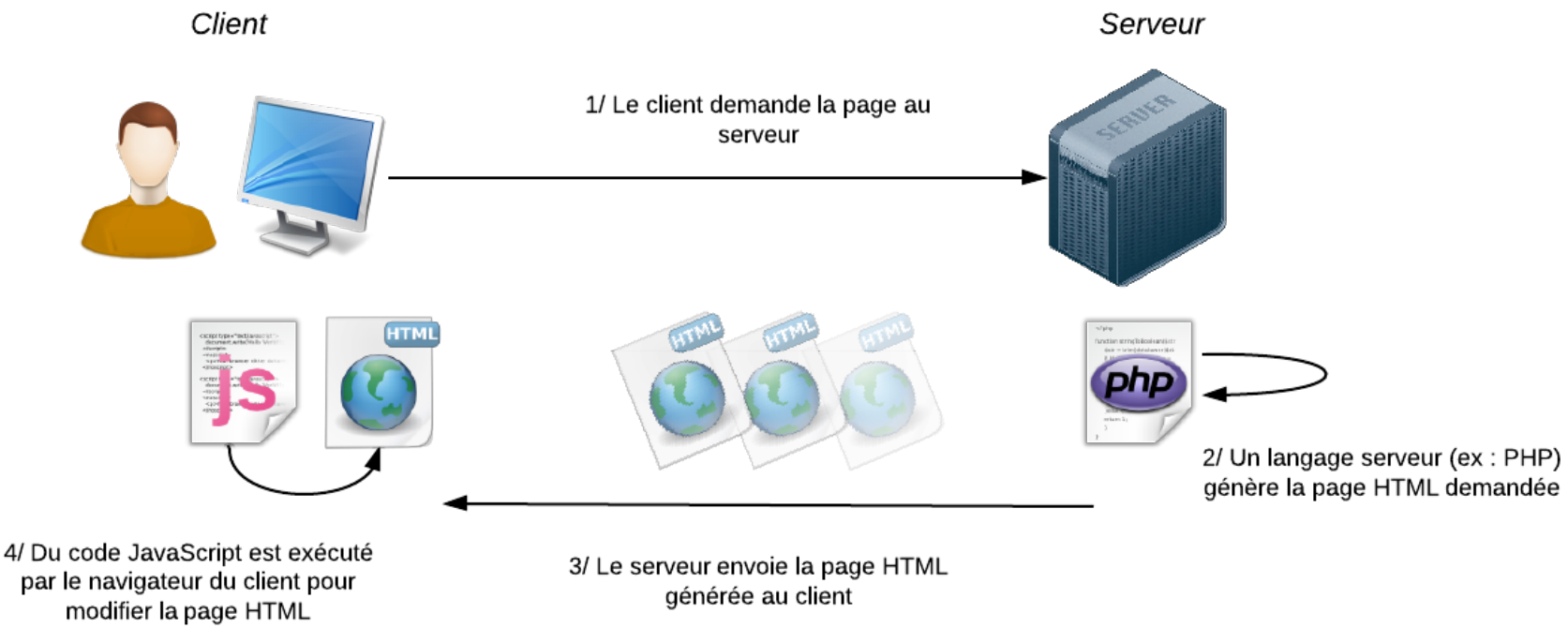
Comment ça marche



Programmation web client/serveur

- Côté serveur
  - Execution réalisée sur le serveur (PHP, Servlet)
  - Resultat de l'exécution : page HTML envoyée par le serveur au navigateur
- Côté client
  - Execution réalisée sur le client
  - Navigateur capable de realiser l'exécution
  - Transfert du code EMBARQUÉS dans la page HTML, depuis le serveur vers le client (HTML-embedded scripting)
  - Exemples : Scripts $\rightsquigarrow$ Javascript, Applets $\rightsquigarrow$ Java, ActiveX, Ajax

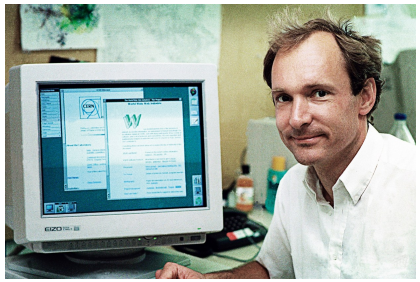
Exemple


Client
Serveur
1/ Le client demande la page au serveur
SERVER
HTML
HTML
HTML
php
2/ Un langage serveur (ex : PHP) génère la page HTML demandée
3/ Le serveur envoie la page HTML générée au client
4/ Du code JavaScript est exécuté par le navigateur du client pour modifier la page HTML

Overview

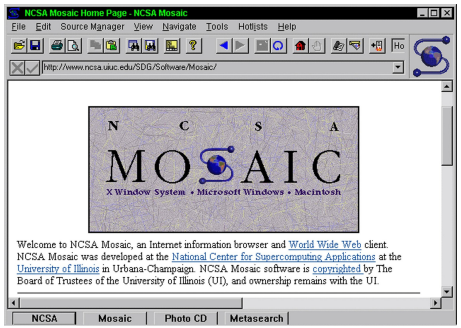
HTML

• **HyperText Markup Language** : langage de mise en forme de documents hypertextes (texte + liens vers d'autres documents). Développé au CERN en 1989 par Tim Berners-Lee.



• 1991 : premier navigateur en mode texte

• 1993 : premier navigateur graphique (mosaic) développé au NCSA (National Center for Supercomputing Applications)



Document HTML

- Est un document semi-structuré
- Dont la structure est donnée par des balises

Exemple	Rendu par défaut
`Un texte <b>en gras</b>`	Un texte **en gras**
`<a href="http://www.ig2i.fr/">Un lien</a>`	Un lien
`<ul > <li>Point 1</li> <li>Point 2</li> </ul>`	• Point 1 • Point 2

- On dit que `<toto>` est une balise ouvrante et `</toto>` une balise fermante. On peut écrire `<toto/>` comme raccourci pour `<toto></toto>`.

Un peu d'histoire

1973 : GML, Generalised Markup Language développé chez IBM. Introduction de la notion de balise.

1980 : SGML, Standardised GML, adopté par l'ISO

1989 : HTML, basé sur SGML. Plusieurs entreprises (microsoft,netscape, ...) interprètent le standard de manière différente

1996 : XML, eXtensible Markup Language norme pour les documents semistructurés (SGML simplifié)

2000 : XHTML, version de HTML suivant les conventions XML

2008 : Première proposition pour le nouveau standard, HTML5

2014 : Standardisation de HTML5

XHTML vs HTML

- XHTML version « XML » de HTML. Principales différences :
  - Les balises sont bien parenthésées (<a> <c> </c> est interdit)
  - Les balises sont en minuscules
- Les avantages sont les suivants
  - HTML autorise les mélanges majuscule/minuscule, de ne pas fermer certaines balise . . . Les navigateurs corrigent ces erreurs de manières différentes
  - Le document est structuré comme un programme informatique (les balises ouvrantes/fermantes correspondent à {et }). Plus simple à débugger.

Rôle de XHTML

• Séparer la structure du document de son rendu. La structure donne une sémantique au document :

- Ceci est un titre
- Ceci est un paragraphe
- Ceci est un ensemble de caractères importants

• Cela permet au navigateur d'assurer un rendu en fonction de la sémantique. Il existe différents types de rendus:

- Graphique interactif (Chrome, Firefox, Internet Explorer, . . .)
- Texte interactif (Lynx, navigateur en mode texte)
- Graphique statique (par ex: sur livre électronique)
- Graphique pour petit écran (terminal mobile)

Exemple de document

Documentsub.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
  <head>
   <title>Un titre</title>
   <meta http-equiv="Content-Type" content="text/html;charset=utf-8" />
  </head>
  <body>
   <h1>Titre de section</h1>
   <p> premier paragraphe de texte. On met
    un <a href="http://www.lri.fr">lien</a> ici.
   </p>
  <!-- on peut aussi mettre des commentaires -->
  </body>
</html>
```

Structure d'un document XHTML

Pour être valide un document XHTML contient au moins les balises suivantes

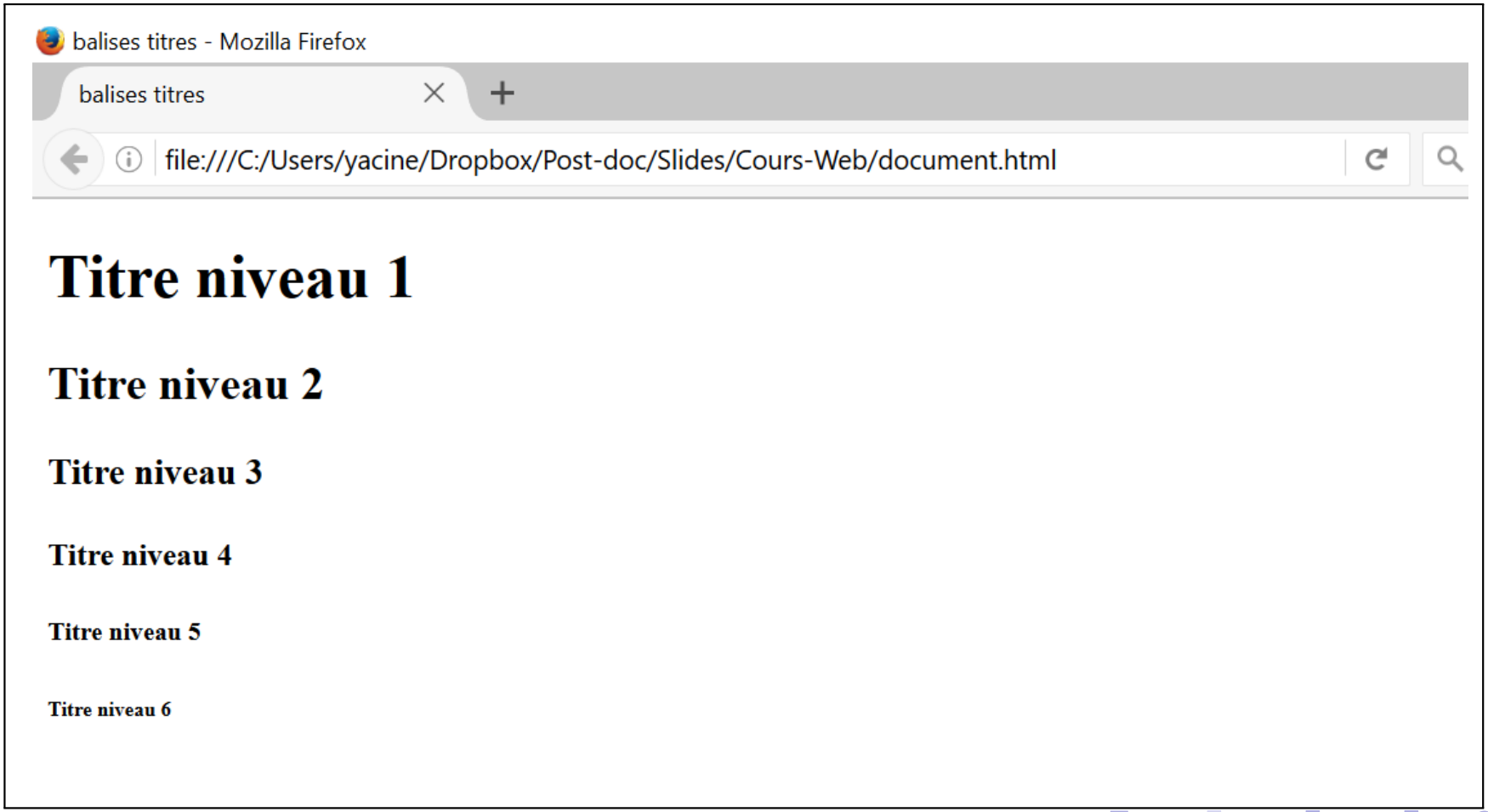
- Une balise `<html>` qui est la racine (elle englobe toutes les autres balises). La balise `html` contient deux balises filles: head et body

- La balise `<head>` représente l'en-tête du document. Elle peut contenir diverses informations (feuilles de styles, titre, encodage de caractères, ...). La seule balise obligatoire dans head est le titre (`<title>`). C'est le texte qui est affiché dans la barre de fenêtre du navigateur ou dans l'onglet.

- La balise `<body>` représente le contenu de la page. On y trouve diverses balises (`<div>, <p>, <table>,` ...) qui formatent le contenu de la page

Titres

- Les balises <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, permettent de créer des titres de section, sous-section , sous-sous-section ,. . .
- Six niveaux hiérarchiques sont possibles



Paragraphes

Des paragraphes de textes sont introduits avec les balises <p>. Par défaut chaque paragraphe implique un retour à la ligne:

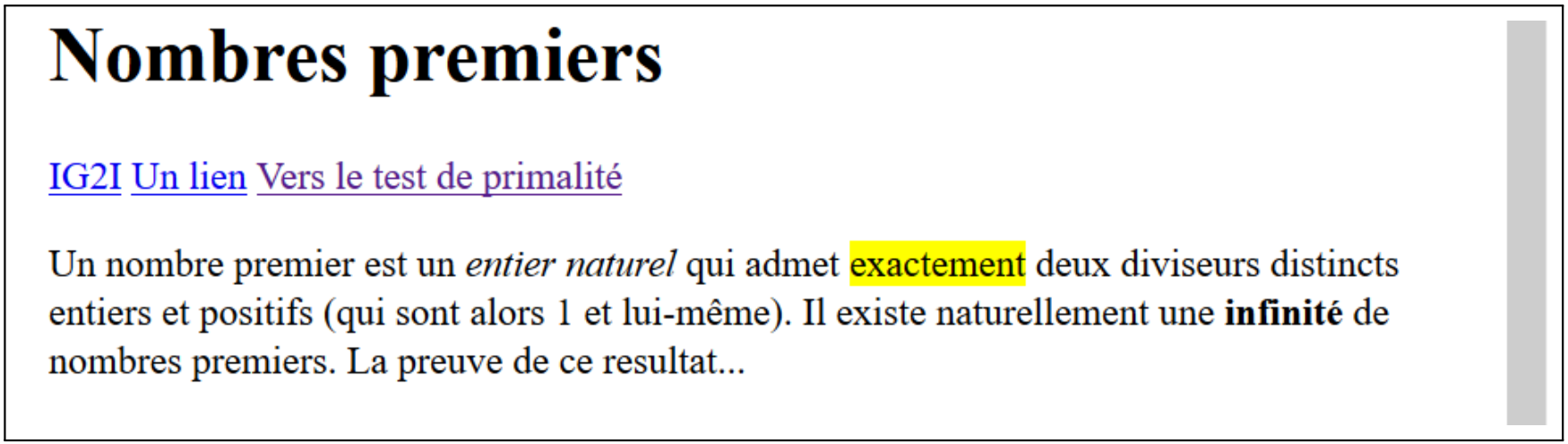
```
9   <h1>Nombres premiers</h1>
10  <p> Un nombre premier est un entier naturel qui admet exactement deux diviseurs
    distincts entiers et positifs (qui sont alors 1 et lui-même). Ainsi, 1 n'est pas
    premier car il n'a qu'un seul diviseur entier positif ; 0 non plus car il est
    divisible par tous les entiers positifs.
11  </p>
```

Remarque : par défaut, les espaces, retour à la ligne, . . . sont ignorés et le texte est reformaté pour remplir la largeur de la page

Mise en valeur du texte

- Les balises `<em>` (emphasis, important), `<strong>` (très important), `<mark>` (visible, pertinent), et bien d'autres, permettent de mettre en valeur du texte.

```
13  <p> Un nombre premier est un <em>entier naturel</em> qui admet <mark>exactement</mark>
    deux diviseurs distincts entiers et positifs (qui sont alors 1 et lui-même). Il existe
    naturellement une <strong>infinité</strong> de nombres premiers. La preuve de ce
    resultat...
14  </p>
```



Liens hyper-text

On peut faire référence à une autre ressource en utilisant un lien hyper-texte (balise `<a/>` et son attribut href). La cible du lien peut être absolue ou relative. Si l'URL est relative, le chemin est substitué à la dernière composante de l'URL de la page courante. Si l'URL commence par un # elle référence, l'attribut id d'un élément de la page:

```
 8    <body>
 9      <h1>Nombres premiers</h1>
10  <a href="https://www.IG2I.fr">IG2I</a>
11  <a href="../../../index.html">Un lien</a>
12  <a href="#foo">Vers le test de primalité</a>
13  <p> Un nombre premier est un entier naturel qui admet exactement deux diviseurs distincts
    entiers et positifs (qui sont alors 1 et lui-même). Ainsi, 1 n'est pas premier car il n'a
    qu'un seul diviseur entier positif ; 0 non plus car il est divisible par tous les entiers
    positifs.
14  </p>
15    <h1 id="foo">Test de primalité</h1>
```

Liste

On peut créer des listes énumérées (avec <ol>, ordered list) ou non énumérées (avec <ul>, unordered list). Chaque ligne est limitée par une balise <li> (list item)

```
    <h1>Listes</h1>
<ul>
  <li> Un élément </li>
  <li> <ol> <li> Un autre élément </li>
            <li> <ol> <li> Un sous-élément</li>
                      <li> Un autre sous-élément</li>
                 </ol>
            </li>
        </ol>
  <li>Le dernier</li>
</ul>
```

Listes

- Un élément
- 1. Un autre élément
  2. 1. Un sous-élément
     2. Un autre sous-élément
- Le dernier

Tableaux

On peut formater des tables en utilisant :

- La balise `<table>` pour délimiter la table
- La balise `<tr>` pour délimiter une ligne de la table
- La balise `<th>` pour délimiter une tête de colonne
- La balise `<td>` pour délimiter une case
- L'attribut `colspan` permet de fusionner des colones

```
 9     <h1>Tableaux</h1>
10  <table>
11  <tr> <th>Nom</th> <th>Prénom</th> <th>Note 1</th> <th>Note 2</th></tr>
12  <tr> <td>Foo</td> <td>Bar</td> <td> 15</td> <td>12</td> </tr>
13  <tr> <td>Doe </td> <td>Jonh</td> <td colspan="2">Absent</td></tr>
14  </table>
```

Tableaux

Nom	Prénom	Note 1	Note 2
Foo	Bar	15	12
Doe	Jonh	Absent	

Remarques générales

- De nombreuses autres balises existent + de nouvelles avec HTML 5
- On n'a normalement pas le droit de mettre n'importe quel élément n'importe où (i.e. pas de `<li>` tout seul)
- Il existe une spécification précise de HTML 5 (plusieurs dizaines de pages uniquement pour les balises)
- Il existe aussi des validateurs, il faut les utiliser le plus possible
- De manière générale, les espaces sont ignorés, on prendra donc bien soin de les utiliser judicieusement pour rendre le code de la page lisible
- Tous les éléments ont un style (moche) par défaut. On peut modifier ce style grâce à des propriétés CSS.

Overview

Cascading Style Sheets (CSS)

CSS : Langage permettant de décrire le style graphique d'une page HTML

On peut appliquer un style CSS

- À un élément en utilisant l'attribut `style`
- À une page en utilisant l'élément `<style>...</style>` dans l'en-tête du document (dans la balise `<head>...</head>`).
- À un ensemble de pages en référençant un fichier de style dans chacune des pages

L'attribut style

```
<a href="http://www.IG2I.fr" style="color:red">Un lien</a>
```

Apperçu :

Un lien

Inconvénients :

- Il faut copier l'attribut style pour tous les liens de la page
- Modification de tous les éléments difficiles

L'élément style

```
<html>
    <head>
       <title>...</title>
       <style>
       a { color:  red; }
       </style>
    </head>
    <body>
       <a href="...">Lien 1</a> <a href="...">Lien
2</a>
    </body>
</html>
```

Apperçu :

Lien 1 Lien 2

Inconvénient : local à une page

Fichier .css séparé

Fichier `style.css` :

```
a { color:  red; }
```

Fichier `test.html` :

```
<html>
   <head>
    ...
    <link href="style.css" type="text/css"
rel="stylesheet" />
    </head>
    ...
</html>
```

Modifications & déploiement aisés

Syntaxe

Une propriété CSS est définie en utilisant la syntaxe:

```
nom_prop :  val_prop ;
```

Si on utilise l'attribut `style` d'un élément:

```
<a href="..."
style="color:red;border-style:solid;border:1pt;">Lien
1</a>
```

Si on utilise un fichier `.css` ou une feuille de style:

```
a {
color :  red;
border-style:  solid;
border:  1pt;
}

h1 { /* Le style des titres de niveau 1 */
text-decoration:  underline;
color:  green;
}
```

Voila pour le rappel

Pour creuser plus en détails le HTML et le CSS, quelques pointeurs :

- Spécification W3C pour HTML : http://www.w3.org/TR/html5/
- Spécification W3C pour CSS : http://www.w3.org/Style/CSS/specs.en.html
- Le site de tutoriels W3Schools :http://www.w3schools.com/
- Le cours HTML5 & CSS3 : http://openclassrooms.com/
- Wysiwyg en ligne : https://www.codevolve.com/

Overview

Web dynamique ?

- Le modèle du Web présentée précedemment est statique. Les documents sont stockés sous forme de fichiers physiques, sur le disque dur d'un serveur.
- Très tôt on a souhaité générer dynamiquement le contenu d'une page.
- 1993 : invention des scripts CGI qui permettent au serveur de récupérer les paramètres d'une requête HTTP et de générer du HTML en réponse.
- La programmation Web côté serveur évolue ensuite (apparition de PHP en 1994, puis possibilité ensuite de programmer le côté serveur dans des langages plus robustes, comme Java, ...)
- Un problème subsiste : le manque d'interactivité. En effet, on est contraint par le modèle :
formulaire HTML $\rightsquigarrow$ envoi au serveur $\rightsquigarrow$ calcul de la réponse $\rightsquigarrow$ retour au client $\rightsquigarrow$ rechargement de page. Problème d'interactivité (latence réseau, rendu graphique d'une nouvelle page, ...).

Web dynamique côté client

- Avec l'arrivée de Java (1995) la notion d'Applet fait son apparition. Ils sont (pour l'époque) une manière portable d'exécuter du code côté client.

- Problème : Java est trop lour à l'époque (c'est un vrai langage, il fait peur aux créateurs de site, les performances sont médiocres, . . .).

- 1995 : Brendan Eich (Netscape) crée Javascript en 10 jours. Il emprunte de la syntaxe à Java/C, et Netscape Navigator 2.0 embarque un interpréteur Javascript en standard

- Le langage est rapidement adopté, mais chaque navigateur implémente sa propre variante. Le langage lui-même est standardisé en 1996 (ECMAScript, standardisé par l'ECMA).

- Malheureusement l'API (c'est à dire les fonctions permettant d'accéder au navigateur depuis Javascript) n'est standardisée que tardivement par le W3C (et tout n'est pas encore supporté par tout le monde à ce jour

Javascript Execution

• Côté client : le code javascript est exécuté par le navigateur Web. Il doit donc être référencé dans une page HTML :

- Soit en utilisant l'attribut src d'une balise script (externe)
- Soit en mettant le code directement dans une balise script (interne)

```
3  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" >
4  …
5  <script type="text/javascript" src="toto.js"></script>
6  …
7  <script type="text/javascript">
8  alert("Hello, World!");
9  </script>
10 </html>
```

• Côté serveur : on peut maintenant utiliser Javascript comme un langage généraliste grâce à l'interpréteur Node.js

Description du langage

Javascript est un langage :

- Dynamique (tout est fait à l'exécution)
- En particulier dynamiquement typé (donc pas typé)
- Impératif (effets de bords, boucles for, notion d'instruction, . . .)
- Fonctionnel (les fonctions sont des objets de première classe que l'on va manipuler à haute dose)
- Objet (mais sans notion de classe, ce qui rend la chose amusante)
- Interprété, avec une compilation à la volée (JIT). Les navigateurs Web moderne ont des performances incroyables (possibilité de faire des jeux 3D par exemple)

Environnement de développement

- Pour les premiers cours, on utilisera le navigateur Chrome™.
- Il est recommendé d'utiliser le même navigateur pour s'abstraire dans un premier temps des problèmes de compatibilité
- On peut utiliser un éditeur de texte simple (Eclipse est a proscrire, le support Javascript est notoirement mauvais)
- On utilisera la console de débuggage Javascript de Chrome (Ctrl-Shift-J)

Number

Il n'y a pas de type entier, uniquement des numbers qui sont flottants IEEE-754 double précision (64 bits : 53 bits pour la mantisse, 11 bits pour l'exposant, 1 bit de signe).

Notation décimale entière :	$10, 3444, -25, 42, \ldots$
Notation scientifique :	$1.3, 0.99, 00.34e102, -2313.2313E-23, \ldots$
Notation octale :	$0755, -01234567 \ldots$
Notation hexadécimale :	$0x12b, -0xb00b5, 0xd34db33f, \ldots$

Le standard garanti que tous les entiers 32bits sont représentatbles exactement (sans arrondi). On peut écrire des entiers plus grands que $2^{31} - 1$ mais au bout d'un moment on a une perte de précision.

Opérateurs arithmétiques :

$-$:	Moins unaire
$+, -, *, \%$:	addition, soustraction, produit, modulo
$/$:	Division (flottante)

Booléens

true/false : vrai/faux

Opérateurs logiques :

! : négation (unaire)

&&, || : « et » logique, « ou » logique

Variables, affectations

- Un nom de variable commence toujours par une lettre (majuscule ou minuscule), $ ou _ et se poursuit par un de ces caractères ou un chiffre.
- On utilise le mot clé **var**

Exemples :

```
var $foo = 123;

var bar = 1323e99;

var _toto = bar;
```

- Attention on peut définir une variable sans l'avoir déclarée, et ça marche mais ça ne fait pas ce que l'on pense.

Chaîne de caractères (string)

Encodées en UTF-16 (ou UCS-2), délimitées par des « ’ » ou « " »

Opérations sur les chaînes :

`foo[10]` :	accès au 11 ème caractère
pas de mise à jour :	les chaînes sont immuables
`+` :	concaténation
`s.length` :	longueur
`s.concat("23")` :	concaténation (bis)

Un grand nombre de méthodes sont disponible, on les verra prochainement (expressions régulières, recherche, remplacement, . . .)

Comparaisons

Opérateurs de comparaisons

Opérateur	Description
`$a == $b`	Égal, après conversion de type
`$a != $b`	Différent, après conversion de type
`$a === $b`	Égal et de même type
`$a !== $b`	Différent ou de type différent
`$a < $b`	Strictement plus petit, après conversion de type
`$a > $b`	Strictement plus grand, après conversion de type
`$a <= $b`	Plus petit, après conversion de type
`$a >= $b`	Plus grand, après conversion de type

Objet

La structure de donnée de base est l'objet

```
{ }                    //Un objet vide
{ x : 1, y : 2 }       //Un objet avec deux champs x et y.
o.x                    //Accès à un champ
o['x']                 //Syntaxe alternative
o.z = 3;               //rajoute le champ z à l'objet o!
```

En javascript, tout est objet

```
"123".concat("456")    //renvoie la chaîne "123456"
3.14.toString()        //renvoie la chaîne "3.14"
```

Instructions

Comme en C/C++/Java . . . les expressions sont aussi des instructions

```
x = 34;
25; //la valeur est jetée.
f(1999);
```

Javascript essaye d'insérer automatiquement des « ; ». Pour ce cours on ne lui fait pas confiance et on termine toutes les instructions, sauf les blocs par un « ; »

Conditionnelle

```
if ( c ) {
      // cas then
} else {
      // cas else
}
```

Les parenthèses autour de la condition "c" sont obligatoires. La branche else { . . . } est optionnelle. Les accolades sont optionnelles pour les blocs d'une seule instruction.

Boucles

```
while ( c ) {
      //corps de la boucle while
}

do {
      //corps de la boucle do
} while ( c );

for(init ; test ; incr) {
      //corps de la boucle for
}
```

`break` : sort de la boucle immédiatement

`continue` : reprend à l'itération suivante

Fonctions

On peut définir des fonctions globales :

```
function f(x1, ..., xn) {
// instructions
};
```

On utilise le mot clé return pour renvoyer un résultat (ou quitter la fonction sans rien renvoyer)

On peut aussi créer des fonctions « inline » :

```
var z = 1 + (function (x, y ) { return x * y; })(2,3);
// x contient 7
```

On dispose donc de la syntaxe alternative pour la déclaration de fonction :

```
var f = function (z) { return x+1; };
```

Fonctions: suite

En première approximation, « les méthodes » ne sont que des fonctions stockées dans le champs d'un objet :

```
var obj = { x :  1, y :  1 }; // objet
obj.move = function (i, j) {
obj.x += i;
obj.y += j;
};

obj.move(2,3);
```

On verra que c'est beaucoup plus subtil que ça.

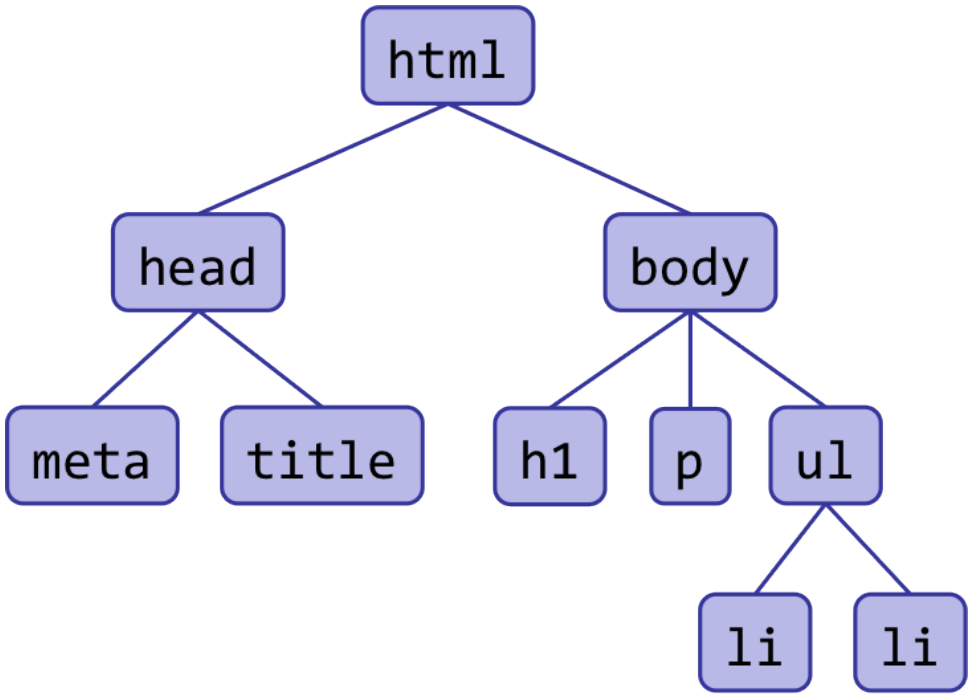
Overview

Quelque fonctionalités de base

On ne fait ici qu'un très bref survol des fonctionalités, suffisamment pour faire le TP 1. On reviendra en détails sur tous ces concepts.

Objet Global `document`

L'objet global `document` représente le document HTML. Il implémente l'interface DOM et on peut donc le parcourir comme un arbre (propriétés `firstChild, parent, nextSibling ...`).



Méthodes `document`

La méthode `document.getElementById("foo")` permet de récupérer un objet représentant l'élément HTML de la page ayant l'attribut id valant "foo" (null si cet élément n'existe pas).

La méthode `document.getElementsByTagName("div")` permet de récupérer sous la forme d'un tableau tout les elements de la famille `<div>`.

- Le résultat de ces méthodes sont des élements HTML: `Element` qui sont également des objets. Tous ces objets sont de type `Node` (héritage).

Traitement des élements HTML

Les attributs

- Méthodes `getAttribute()` et `setAttribute()` de l'objet `Element` permettant respectivement de récupérer et d'éditer un attribut.
- Pour les éléments courants il suffit d'une `Element.attribut`

Le contenu

- La méthode `innerHTML` permet de récupérer le code HTML d'un noeud fils d'un élément sous forme de texte.
- Les méthodes `innerText/ textContent` récupère uniquement le texte

Résumé

- DOM et Javascript permettent d'accéder au éléments HTML présents dans un document, de les modifier, d'interagir avec.

- Les élement HTML sont structuré comme un arbre généalogique

- Différentes méthodes pour accéder/modifier les éléments : `getElementById("foo")`, `getAttribute()`, `innerHTML`, ...