

Développement d'Applications Web

Cours Licence L2 - 2018-2019

Version du 03/02/2019 23:37

Sommaire

1	Introduction à la programmation Web	4
1.1	Définitions	4
1.2	Fonctionnement du Web	8
1.3	Historique	11
1.4	Programmation web client/serveur :	12
2	Programmation coté client.....	13
2.1	HTML et CSS.....	13
2.1.1	Basiques d'un document HTML	15
2.1.2	Tables HTML, forms, lists, frames	34
2.1.3	Basiques de CSS.....	34
2.1.4	HTML5.....	41
2.1.5	CSS3.....	41
2.2	XML	41
2.2.1	Structure	41
2.2.2	DTD, XML schema, XSLT	41
2.3	JavaScript	42
2.3.1	Basiques de JavaScript.....	43
2.3.1.1	Indentation.....	44
2.3.1.2	Le Javascript « dans la page ».....	44
2.3.1.3	Javascript externe	45
2.3.1.4	Les variables.....	47
2.3.1.5	Les types de variables	47
2.3.1.6	Tester l'existence de variables avec typeof	48
2.3.1.7	Opérateurs	49
2.3.1.8	Interagir avec l'utilisateur	51
2.3.1.9	Convertir entre chaînes de caractères et nombres	52
2.3.1.10	Les alternatives.....	52
2.3.1.11	Petit intermède : la fonction confirm().....	53
2.3.1.12	La condition « switch »	53
2.3.2	Les boucles.....	54
2.3.2.1	Portée des variables de boucle	55
2.3.3	Fonctions, objets et tableaux	56
2.3.3.1	Variables globales et variables locales	56
2.3.3.2	Les arguments et les valeurs de retour	58
2.3.3.3	Les fonctions anonymes	59
2.3.4	Les objets et les tableaux.....	60
2.3.4.1	Les objets littéraux	61

2.3.5	Modeler vos pages Web (DHTML et DOM).....	63
2.3.5.1	Le Document Object Model.....	63
2.3.5.2	L'objet window.....	63
2.3.5.3	Le document.....	64
2.3.5.4	Naviguer dans le document : La structure DOM	65
2.3.5.5	Accéder aux éléments : getElementById()	66
2.3.5.6	getElementsByName()	67
2.3.5.7	Accéder aux éléments grâce aux technologies récentes	68
2.3.5.8	L'héritage des propriétés et des méthodes.....	69
2.3.5.9	La classe	71
2.3.5.10	Le contenu : innerHTML	72
2.3.5.11	Naviguer entre les nœuds : La propriété parentNode	73
2.3.5.12	nodeType et nodeName	74
2.3.5.13	Lister et parcourir des nœuds enfants	75
2.3.5.14	nodeValue et data.....	76
2.3.5.15	childNodes.....	76
2.3.5.16	Créer et insérer des éléments : Ajouter des éléments HTML..	77
2.3.5.17	Cloner, remplacer, supprimer.....	79
2.3.5.18	Remplacer un élément par un autre.....	80
2.3.5.19	Supprimer un élément	80
2.3.5.20	Insérer à la bonne place : insertBefore()	81
2.3.6	Interaction back-end via Ajax	83
2.3.7	La bibliothèque jquery	83
2.4	Framework Angular	83
2.4.1	Basiques de Angular et le langage typescript	83
2.4.2	Premier exemple.....	83
2.4.3	Composants maître/détails	83
2.4.4	Services	83
2.4.5	Routing	83
2.4.6	HTTP	83
2.5	Exemples d'application.....	83
2.5.1	Application 1	83
2.5.2	Application 2	83
2.5.3	83
3	Programmation côté serveur	84
3.1	PHP	84
3.1.1	Basiques de PHP	84
3.1.2	Expressions et flots de contrôle.....	84
3.1.3	Fonctions et objets.....	84
3.1.4	Tableaux	84
3.1.5	Pratiques PHP.....	84
3.1.6	Manipulation des formes.....	84
3.1.7	Cookies, sessions, et authentification	84
3.2	Connexion BDD	84

3.2.1	MySQL.....	84
3.2.2	Postgres	84
3.2.3	Accès BDD via PHP	84
3.3	Framework Laravel	84
3.4	Exemples d'application.....	84
3.4.1	Application 1	84
3.4.2	Application 2	84
3.4.3	84
4	Services Web	85
4.1	Introduction	85
4.2	SOA	85
4.3	Standards des services Web	85
4.4	Plateformes de développement.....	85
4.5	Plateformes .NET et Java	85
4.5.1	Application 1	85
4.5.2	Application 2	85
4.5.3	85
5	Fiche TP n°1 : Prise en main HTML/CSS	86
5.1	Des couleurs	86
5.2	Paragraphes.....	86
5.3	Titres.....	87
5.4	Paragraphes spéciaux	87
5.5	Les barres horizontales	87
5.6	Styles, couleurs, tailles, fontes	88
5.7	Énumérations	88
5.8	Tableaux	88
5.9	Liens	89
5.10	Images	90
5.11	CSS dans le corps du code	91
5.12	Les CSS dans l'en-tête de la page.....	91
5.13	Les CSS dans une feuille de style totalement séparée	92
5.14	Comment utiliser des classes pour appliquer un style	93
6	Fiche TP n°2 : Prise en main JavaScript	95
6.1	Un petit exercice pour la forme !	95
6.2	Un deuxième petit exercice	95
6.3	Travail pratique	96
6.4	Mini-TP : recréer une structure DOM.....	97
7	Fiche TP n°3 : Prise en main PHP	98

1 Introduction à la programmation Web

Références : Le texte de ce document est une compilation des textes provenant de :

- [Introduction au fonctionnement du Web, Support de cours, Ecole IGN (http://cours-fad-public.ensg.eu/pluginfile.php/1572/mod_resource/content/1/1-Introduction_au_fonctionnement_du_web-1.pdf)]
- [Prog. Web, HTML 5 et CSS 3 (<https://www.lipn.univ-paris13.fr/~lacroix/>)]
- [Programmation Web Avancée, (<http://researchers.lille.inria.fr/~yabouzid/cours1.pdf>)]
- [Cours HTML, IUT Montpellier, (http://www.lirmm.fr/~croitoru/teaching/iut_special_year.html)]

L'objectif de ce cours est de vous former à programmer des applications web (au niveau client ou au niveau serveur).

1.1 Définitions

- 1) **Réseau** : machines connectées ensemble
- 2) **Internet** : réseau connectant tous les réseaux
- 3) **Architecture client/serveur** : Lorsqu'on connecte les ordinateurs en réseau, il devient intéressant de concentrer certaines ressources (c'est-à-dire des informations et des programmes) sur un seul ordinateur, et de permettre aux autres ordinateurs de se servir de ces ressources uniquement lorsqu'ils en ont besoin. C'est l'architecture client-serveur.
 - i) **Le serveur**, c'est l'ordinateur sur lequel **se trouve une ressource**.
 - ii) **Le client**, c'est l'ordinateur qui a le droit d'accès à la ressource sur le serveur
- 4) **Web** : application permettant de consulter des pages Web à l'aide d'un navigateur.
 - a) Caractérisé par des **hyperliens**
 - b) Distinction entre client et serveur
 - c) Utilise Internet si nécessaire
- 5) **Navigateur** : logiciel de visualisation de pages Web
- 6) **HTML** : langage de balises permettant d'écrire des pages Web

7) Protocole :

langage utilisé pour la communication entre ordinateurs.

Un protocole est une **convention de communication** entre deux ordinateurs. Lorsque deux ordinateurs communiquent entre eux, ils utilisent à chaque fois **plusieurs protocoles** en même temps, et ces protocoles sont inclus les uns dans les autres.

Les protocoles d'internet :

i) TCP/IP

Le protocole de base qui structure Internet (pas seulement le web, mais aussi les autres composantes d'Internet, comme le FTP et le mail) s'appelle TCP/IP. De fait, il s'agit déjà de deux protocoles imbriqués : TCP et IP.

La couche supérieure: Transmission Control Protocol gère le **découpage** d'un message ou un fichier en petits paquets qui sont transmis sur Internet et reçus par une couche **TCP** qui **rassemble** les paquets dans le message original rassemble les paquets dans le message original

La couche inférieure: Internet Protocol gère la partie **adresse** de chaque paquet (**adresse IP**). Chaque ordinateur sur le réseau vérifie cette adresse pour transmettre le message.

ii) HTTP

Au dessus de TCP/IP, le web repose sur le protocole HTTP. HTTP est le protocole spécifique du web. HTTP signifie : **Hyper-Text Transfert Protocole**

HTTP est donc la **langue** dans laquelle le serveur et le client dialoguent.

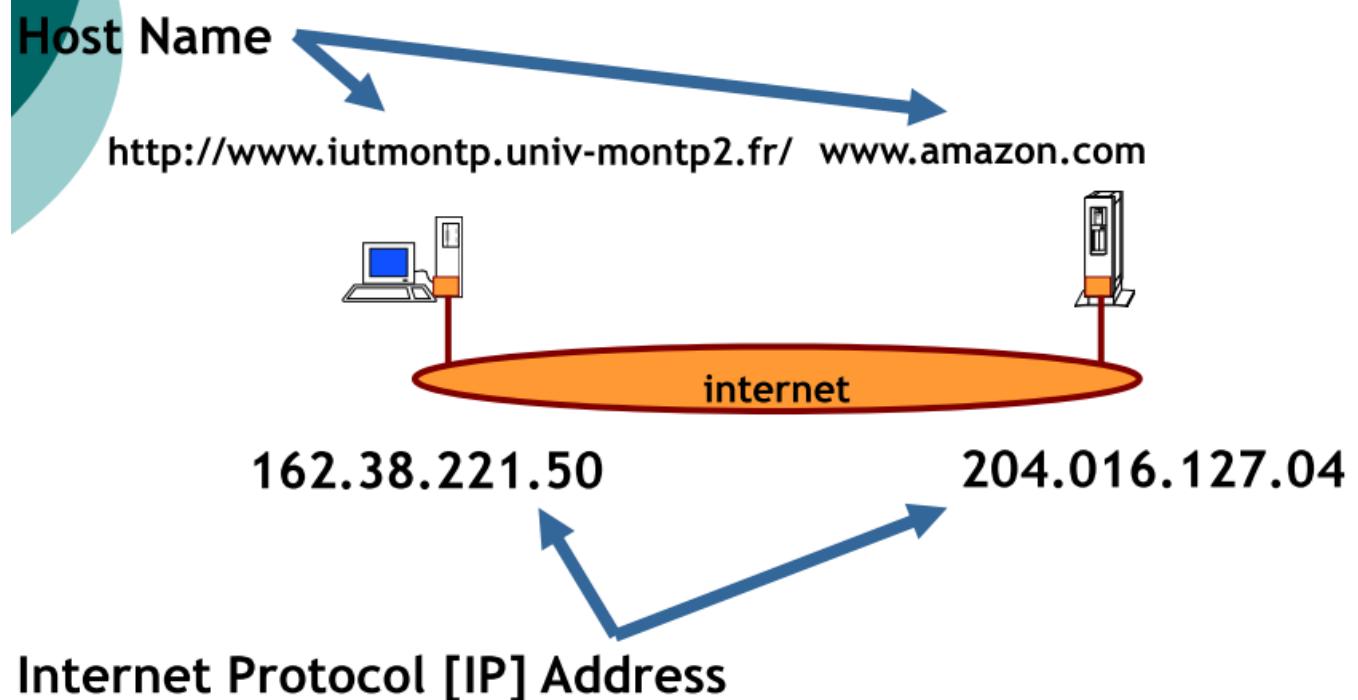
Utilisé pour la **communication entre les clients** Web et serveurs Web clients Web et serveurs Web.

Pour demander une page Web, un navigateur envoie un message **HTTP GET**

Pour envoyer des données vers un serveur Web, un navigateur peut utiliser un message **HTTP POST**

8) Domain Name Service [DNS] :

- a) DNS Domain Name Service: Chaque ordinateur sur Internet a un **nom unique**, utilisée pour référencer à l'ordinateur: www.amazon.co.uk, www.abdn.ac.uk
- b) Chaque ordinateur sur Internet possède également une adresse IP unique, utilisé par TCP / IP pour transmettre ces données DNS (DNS Software)



9) URL :

protocole://adresse_site/chemin_reperatoire/fichier

- a) **protocole**
- b) **adresse_site** : Adresse du serveur (numéro IP ou nom de domaine) contenant le fichier
- c) **chemin_reperatoire** : répertoire où se trouve le fichier sur le serveur
- d) **fichier** : nom du fichier que l'on veut afficher

Exemples d'url :

https://www.univ-oran1.dz/index.php/en/nos-services/uci

http://157.240.21.39/index.php

http://www.facebook.com/

URL=Uniform Resource Locator

URI=Uniform Resource Identifier:

- i) Une manière commune de se référer à n'importe quelle page Web, n'importe où
 - (1)Format... **Protocol://web_server_name/page_name**
 - (2)Example...**http://www.iutmtp.univ-montp2.fr/index.html**
- URL / URI sont utilisés en conjonction avec HTTP pour obtenir soit une URL soit des données POST sur une URL

1.2 Fonctionnement du Web

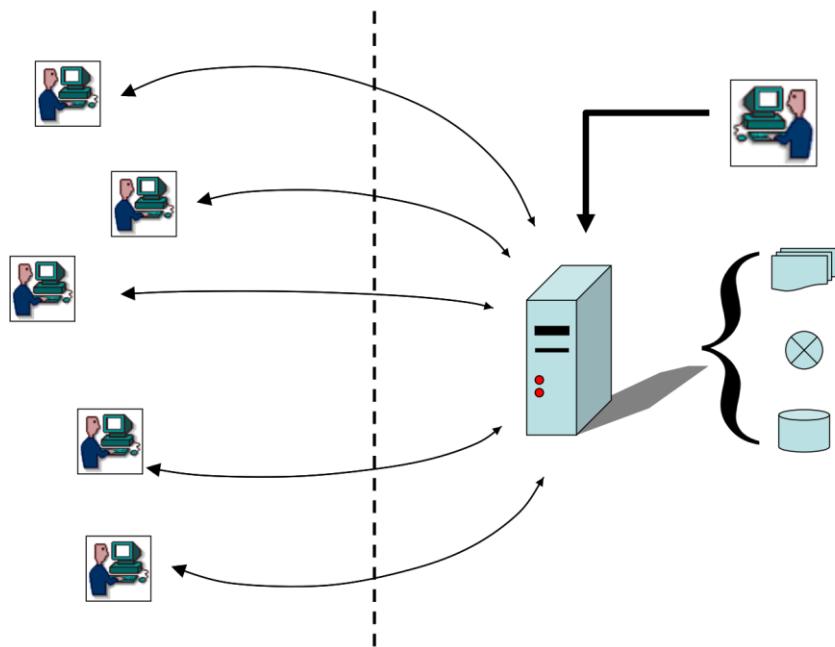


Figure - schéma de l'architecture client-serveur. Plusieurs clients partagent les fichiers, les applications et les bases de données placées en partage sur un serveur. Le trait pointillé entre le serveur et les clients illustre le fait qu'un client et un serveur peuvent être très éloignés l'un de l'autre, par exemple lorsqu'on utilise Internet. A côté des clients, le serveur possède un administrateur, qui l'alimente en données, en programme et en base de données.

Les avantages de l'architecture client-serveur sont :

- 1) **Une économie d'espace mémoire** (tous les postes clients vont chercher l'information en fonction de leurs besoins, et ne doivent pas la stocker durablement)
- 2) **Une garantie de cohérence** (à chaque instant, le serveur constitue un référentiel unique, identique pour tous les clients, alors qu'une duplication sur chaque poste client entraîne généralement une divergence entre les différentes versions, et une perte du référentiel commun). Cette garantie de cohérence est particulièrement souhaitable lorsque l'information subit des mises à jour.
- 3) **Une maîtrise du service** (le gestionnaire du serveur contrôle ce que les programmes clients peuvent faire ou ne pas faire sur le serveur).

Le premier avantage est de nature physique. Les deux autres touchent directement à l'organisation du travail et des processus.

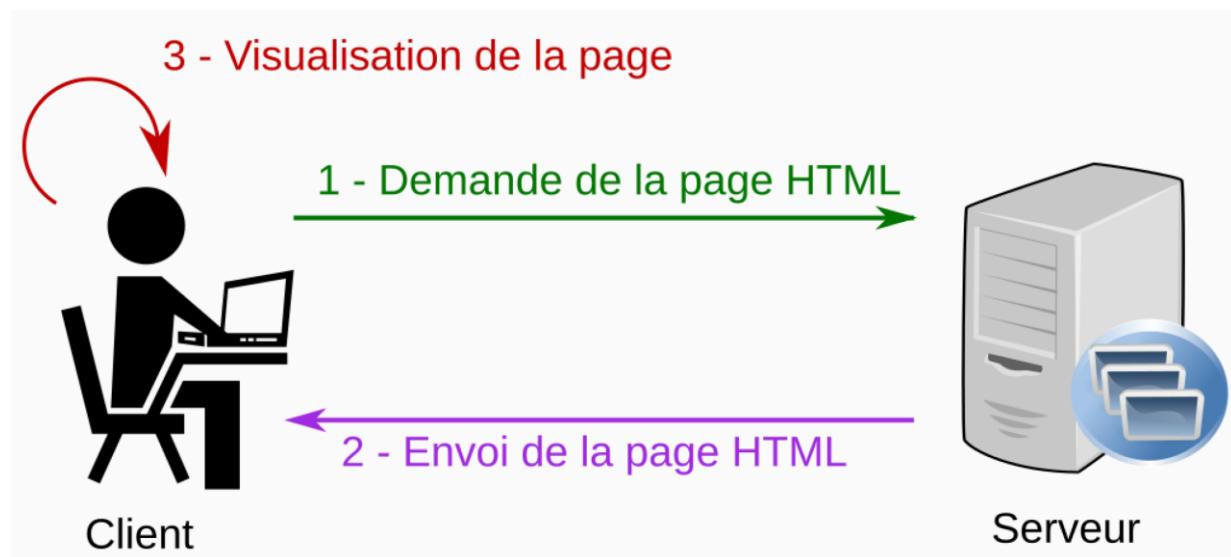
Lorsqu'on parle de « **client** » et de « **serveur** », il peut s'agir à la fois de l'**ordinateur** et du **programme** qui tourne sur cet ordinateur.

Dans la suite de ce cours, « **client** » et « **serveur** » désigneront les **programmes**. Les ordinateurs seront explicitement désignés par « **ordinateur client** » et « **ordinateur serveur** ».

Internet est l'exemple extrême d'architecture client-serveur : chaque serveur est potentiellement accessible par des millions de clients. On peut dire que le Web illustre de manière aiguë les avantages de l'architecture client-serveur : chacun est libre de décider de ce qu'il met en ligne, **chacun possède en permanence le droit d'accéder à un volume de données plus d'un million de fois supérieur à ses capacités de stockage personnelles**.

Sur le web,

- 1) le serveur s'appelle « **serveur internet** ». Le programme le plus utilisé est Apache.
- 2) le client s'appelle **navigateur**, (ou bien browser, ou butineur). Les programmes les plus utilisés sont Chrome, Internet Explorer et Firefox, ...



Quand vous invoquez le site :

<https://geometrica.saclay.inria.fr/team/Marc.Glisse/enseignement/html/>

Que se passe-t-il ?

Etape 1	GET /team/Marc.Glisse/enseignement/html/ Host : geometrica.saclay.inria.fr
Etape 2	<p>HTTP/1.1 200 OK Content-Type: text/html</p> <pre><html xmlns="http://www.w3.org/1999/xhtml"> <head> <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" /> <meta http-equiv="Content-Language" content="fr" /> <meta name="author" content="Marc Glisse" /> <style type="text/css"> h1 { text-align: center; } div.photo { float: right; } </style> <title> TP de HTML </title> </head> <body> <h1>TP de HTML</h1> <p> </pre>
Etape 3	Le navigateur interprète le code HTML et affiche la page web
Etape 1-3...	... ainsi de suite ...

1.3 Historique

Internet à la fin des années 1980 était un ensemble de réseaux qui peuvent communiquer les uns avec les autres dans le cadre du protocole TCP / IP.

Son utilisation était encore largement limité à l'éducation, le gouvernement et les organismes scientifiques.

Deux développements ont déterminé la croissance explosive de l'Internet dans **1990**. Le premier a été l'augmentation rapide des **PC** (en privé et business). L'autre a été la conception et le **développement du World Wide Web**.

Dates :

1992: 1,000,000 utilisateurs de Web connecté a l'Internet et WWW

1993: White House, UK Government, United Nations, World Bank en ligne

1993/94 Utilisation commerciale: transactions par carte de crédit carte de crédi

1995: Netscape, Internet Explorer – video, audio etc.

1995: 40-50,000,000 utilisateurs

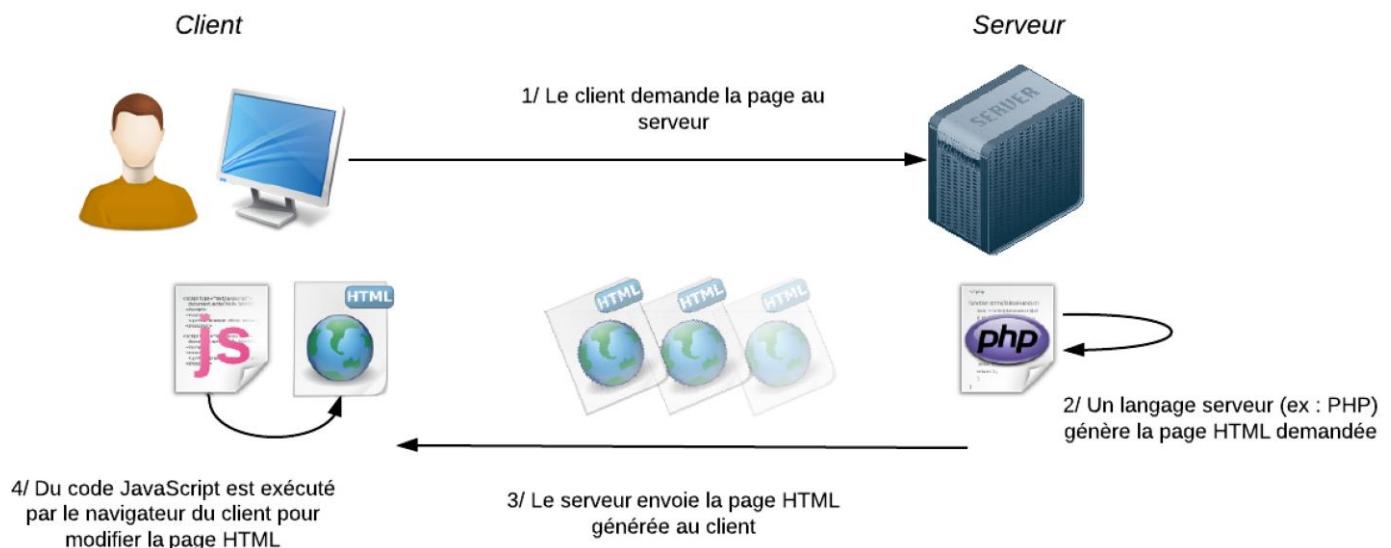
Le plus ancien HTML sur le web : 13 Novembre 1990

"Last-Modified: Wed, 13 nov 1990 15:17:00 GMT".

<https://www.w3.org/History/19921103-hypertext/hypertext/WWW/Link.html>

1.4 Programmation web client/serveur :

- 1) Côté serveur :
 - a) Exécution réalisée sur le serveur (PHP, Servlet)
 - b) Résultat de l'exécution : page HTML envoyée par le serveur au navigateur
- 2) Côté client :
 - a) Exécution réalisée sur le client
 - b) Navigateur capable de réaliser l'exécution
 - c) Transfert du code EMBARQUÉS dans la page HTML, depuis le serveur vers le client (HTML-embedded scripting)
 - d) Exemples : **Scripts Javascript, Applets Java, ActiveX, Ajax, ...**



Technologies :

- Technologies orientées Serveur: génération des pages à la volée
 - **PHP**
 - **Python / Django**
 - **ASP.NET** : <http://www.asp.net/>
 - **Perl** : <http://www.perl.org/>
 - **ColdFusion** : <http://www.adobe.com/products/coldfusion/>
 - **Ruby on Rails** : <http://rubyonrails.org/>
 - ...
- Technologies orientées Client: choses plus sophistiquées que l'affichage statique
 - Chargées et exécutées par le client Web
 - **JavaScript**: langage de script démarqué dans les pages **HTML** pour les rendre plus fonctionnelles
 - Validation formulaire côté client (oubli de fournir tous les détails, format incorrect d'adresse email etc.)
 - Affichage non statique
 - Autres technologies: **Applets Java, Macromedia Flash, Angular, REACT, ...**

2 Programmation côté client

Références : Le texte de ce document est une compilation des textes provenant de :

- [Programmation Web Avancée, (<http://researchers.lille.inria.fr/~yabouzid/cours1.pdf>)]
- [Cours HTML, IUT Montpellier, (http://www.lirmm.fr/~croitoru/teaching/iut_special_year.html)]
- [Écrire ses pages Web en quelques leçons..., (<http://lita.univ-metz.fr/~paris/>)]
- [CSS : vos premiers pas, (<http://www.css-faciles.com/premiers-pas-css.php>)]
- [Dynamisez vos sites web avec Javascript !, (http://www.fr.capgemini.com/carrieres/technology_services/)]

2.1 HTML et CSS

- 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)

HTML est un document semi-structuré, dont la structure est donnée par des balises.

Exemple	Rendu par défaut
Un texte en gras	Un texte en gras
Un lien	Un lien
 Point 1 Point 2 	• 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>.

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 :

- Les balises sont bien parenthésées (`<a> <c> </c> ` est interdit)
- Les balises sont en minuscules

Les avantages de XHTML :

- 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)

2.1.1 Basiques d'un document HTML

Structure d'un document HTML :

- 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

```
<html>
  <head>
    <title> ... </title>
  </head>
  <body>
    ...
  </body>
</html>
```

Exemple

```
<html>
  <head>
    <title>First HTML Document</title>
  </head>
  <body>
Hello, world!
  </body>
</html>
```

Hello, world!

Balises :

- Utilisées pour marquer le début et la fin d'un élément en HTML
- Toutes les balises ont le même format :
 - o commencent par un signe inférieur "<"
 - o terminent par un signe supérieur ">".
- Deux types de tags :
 - o balise d'ouverture: <html>
 - o balises de fermeture: </ html>.
- Différence entre une balise d'ouverture et une balise de fermeture est la barre oblique "/".

Balises (tags) pour :

- Page Title
- Headings
- Paragraphs Paragraphs
- Bold & italic text
- Lists
- Images
- Links

Une page HTML est un fichier ASCII (texte)

Elle est tapée avec un traitement de texte (ex : **Notepad++**, **Emacs**, **Visual Studio**, ...) et enregistrer avec le suffixe **.html ou .htm**

Mais surtout ne pas utiliser le format HTML de **Word** !

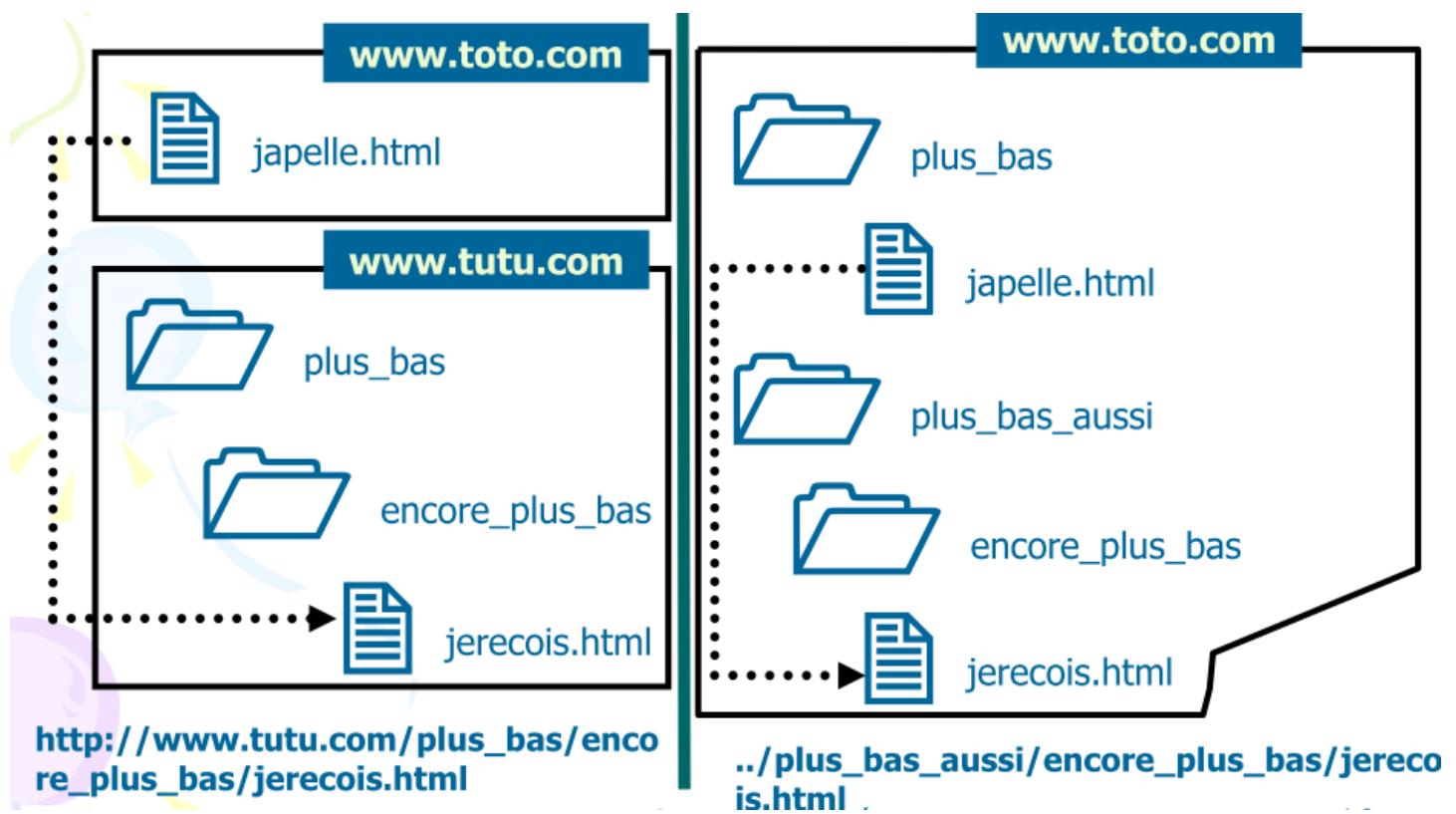
Un commentaire s'écrit entre <!-- et -->

Codage des couleurs : Pour être universel, le codage des couleurs en HTML s'appuie sur le modèle **RVB (Rouge - Vert - Bleu)**

- La couleur décrite par (0,0,0) est le noir.
- La couleur décrite par (255,255,255) est le blanc.
- Le rouge est décrit par (255,0,0)
- Le codage se fait en hexadécimal sur 3x2 chiffres (2 par couleur primaire) précédés par le caractère #
 - o Le noir est représenté par #000000,
 - o le blanc par #FFFFFF et
 - o le rouge par #FF0000
- Les fichiers **images** sont de deux types :
 - o Les images **JPEG**, principalement pour les images de qualité photographique; Leur extension est **.jpg**; Elles offrent une possibilité de compression intéressante pour les utilisateurs ayant une connexion à faible débit.
 - o Les image **GIF**, pour les images de tout type et pour les images animées. Leur extension est **.gif**; Elles sont plus lourdes.

Les noms de fichier :

- Si le fichier est stocké sur le même site que la page appelante, il faut toujours indiquer les noms de fichiers sous leur forme relative, c'est à dire indiquer en plus de leur nom, leur chemin d'accès par rapport à la page qui est affichée
 - o . pour désigner le même répertoire
 - o .. pour le répertoire père
- Si le fichier est stocké sur site web différent de la page appelante, il faut utiliser la notation absolue
 - o toujours commencer le nom du fichier par *http://...*
 - o il faut reprendre ce qui est affiché dans la barre d'adresse du navigateur



Entête :

```
<html>
  <head>
    <title>This is a page title</title>
  </head>
  <body>
    <h1>A heading On My First
    Webpage</h1>
  </body>
</html>
```



La balise META

<meta name="Author" content="auteur1[,auteur2,...]">

permet de rechercher la page dans certains moteurs de recherche en se basant sur les **noms des auteurs indiqués**

<meta name=" Keywords " content="mot-clé1[,mot-clé2,...]">

permet de rechercher la page dans certains moteurs de recherche en se basant sur les **mots-clés indiqués**

<meta http-equiv="Refresh" content="n;url=adresse de page">

permet de passer automatiquement à un autre site après un certain délai

- n = délai en seconde avant l'appel de la deuxième page.
- adresse de page = adresse (absolue ou relative) de la deuxième page Web à afficher

- Les balises **<h1>, <h2>, <h3>, <h4>, <h5>, <h6>**, permettent de créer des titres de section, sous-section , sous-sous-section , . . .

<h1> </h1> : En général, on s'en sert pour afficher le titre de la page.

<h2> </h2> : signifie "titre important".

<h3> </h3> : c'est un titre un peu moins important (on peut dire un "sous-titre" si vous voulez).

<h4> </h4> : titre encore moins important.

<h5> </h5> : titre pas important.

<h6> </h6> : titre pas important du tout.

Titre niveau 1

Titre niveau 2

Titre niveau 3

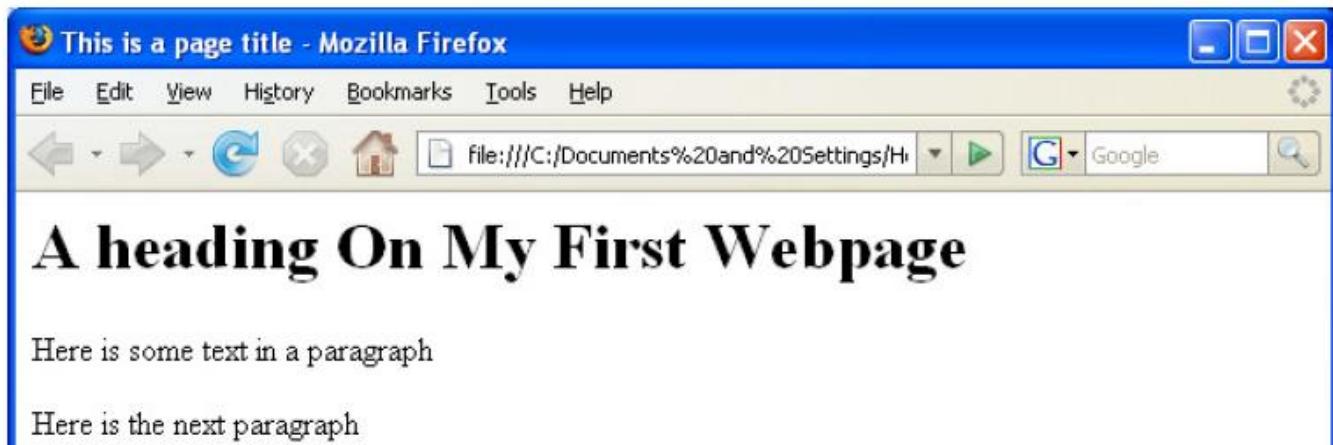
Titre niveau 4

Titre niveau 5

Titre niveau 6

- Un premier contenu : paragraphe

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>A heading On My First Webpage</h1>
    <p>Here is some text in a paragraph</p>
    <p>Here is the next paragraph</p>
  </body>
</html>
```



On peut écrire un paragraphe sans la balise p

<html>

 <head>

 <title>Example</title>

 </head>

 <body>

 <h1>A heading On My First Webpage</h1>

 Here is some text in a paragraph

 Here is the next paragraph

 </body>

</html>



Mise en forme des caractères

- **** : gras
- **<i></i>** : italique
- **<u></u>** : souligné
- **<basefont size="s">** sert à fixer la taille par défaut dans une page Web (s un entier compris entre 1 et 7)
La taille par défaut des caractères est fixée à 3

Mise en forme des caractères

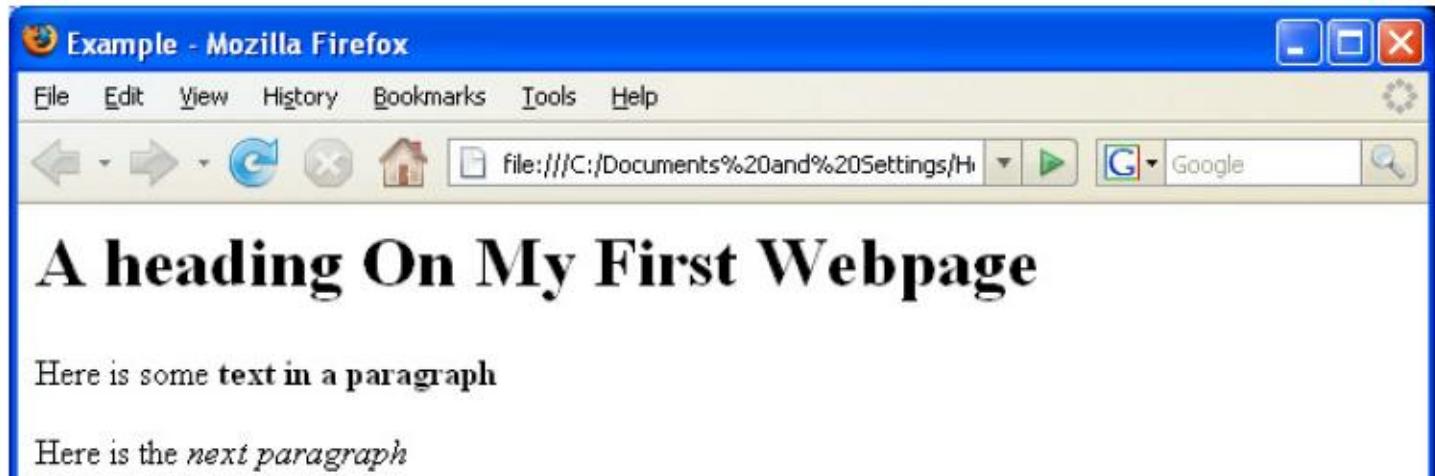
` ... `

- FACE : la première police installée est utilisée. Ne proposer que des polices dont on est sûr qu'elles sont disponibles sur l'ordinateur de consultation (Arial/Helvetica et Times New Roman/Times)
- SIZE : 1..7 (taille absolue) ou +1, +2... (taille relative par rapport à la valeur spécifiée dans BASEFONT) Les balises ` ` peuvent être imbriquées

`toto` est équivalent à `toto`

Gras et italique dans le texte :

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>A heading On My First Webpaqe</h1>
    <p>Here is some <b>text in a paragraph</b></p>
    <p>Here is the <i>next paragraph</i></p>
  </body>
</html>
```



Mise en forme des paragraphes :

- <h1> </h1> : format d'entête
-
 retour à la ligne simple
- <p> retour à la ligne avec saut d'une ligne
- <div align="center"></div> centre le texte
- <div align="right"></div> positionne le texte à droite dans la page
- <div align="left"></div> positionne le texte à gauche dans la page

- Tags imbriqués :

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>A heading On My First Webpage</h1>
    <p>Here is some <b>text in a paragraph</b></p>
    <p>Here is the <i>next paragraph</i></p>
  </body>
</html>
```

Liste :

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>A heading On My First Webpage</h1>
    <p>Here is some <b>text in a paragraph</b></p>
    <p>Here is the <i>next paragraph</i></p>
    <ul>
      <li>First item</li>
      <li>Second item</li>
      <li>Third item</li>
    </ul>
  </body>
</html>
```



type="disc|circle|square" dans permet de choisir sa puce.

Exemple : <ul type="disc">.

Liste ordonnée :

```
<html>
  <head>
    <title>Example</title>
  </head>
  <body>
    <h1>A heading On My First Webpage</h1>
    <p>Here is some <b>text in a paragraph</b></p>
    <p>Here is the <i>next paragraph</i></p>
    <ol>
      <li>First item</li>
      <li>Second item</li>
      <li>Third item</li>
    </ol>
  </body>
</html>
```



L'option `start=nombre_de_départ` pour démarrer la liste à une valeur spécifique autre que 1.
l'option `type=i|I|a|A|1` pour changer le type de numérotation.

- 1=chiffres arabes (option par défaut),
- i=romains minuscules,
- I=majuscules,
- a=lettres minuscules,
- A=majuscules.

Exemple : `<ol type="I" start=4>`

Les listes hiérarchiques

- listes imbriquées l'une dans l'autre sans puce ni numéro.
- Encadrées par `<dl></dl>`
- Chaque nouvelle ligne de niveau 1 commence par `<dt>`
- Chaque nouvelle ligne de niveau 2 commence par `<dd>`

The diagram illustrates a hierarchical list structure. It starts with an open `<dl>` tag on the left. Inside, there are three levels of nesting indicated by blue arrows pointing downwards. The first level contains three `<dt>niv1` tags. The second level contains three `<dd>niv2` tags, each corresponding to a `<dt>niv1` tag from the first level. The third level contains two `<dt>niv1` tags, each corresponding to a `<dd>niv2` tag from the second level. The entire structure is enclosed in a closing `</dl>` tag on the right.

```
<dl>
  <dt>niv1
    <dd>niv2
    <dd>niv2
    <dd>niv2
  <dt>niv1
    <dd>niv2
  <dt>niv1
  <dt>niv1
</dl>
```

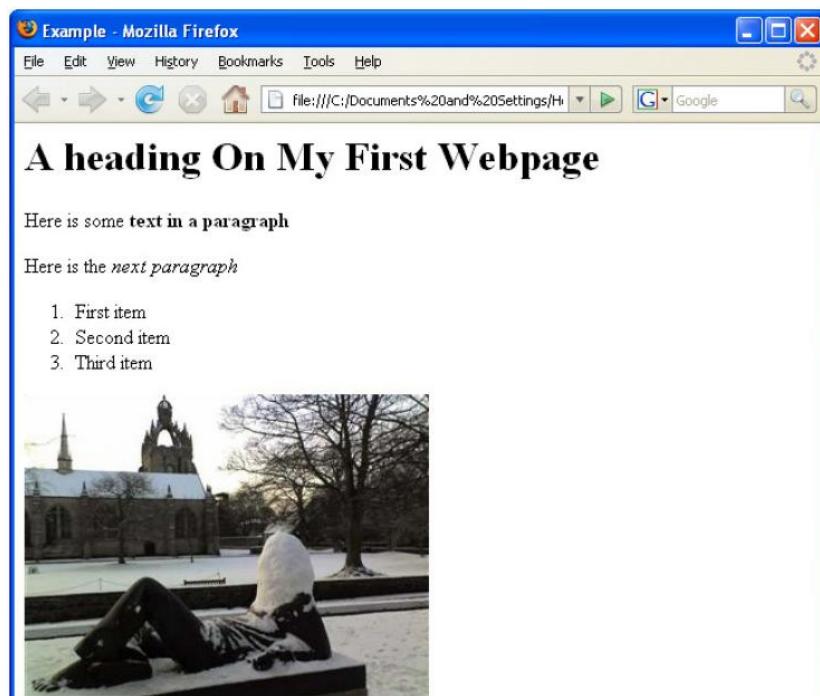
Création de lignes horizontales

<hr size="s" width="w|w%" align = "LEFT|CENTER|RIGHT" color = "#RRVVBB" noshade>

- SIZE : épaisseur en pixel de la ligne
- WIDTH : largeur de la ligne. Peut s'exprimer en valeur absolue (pixels) ou relative (en pourcentage de la largeur de la fenêtre)
- ALIGN : alignement à gauche|au centre|à droite dans la fenêtre (à utiliser si la ligne ne fait pas toute la largeur de la fenêtre)
- COLOR : couleur de la ligne. Option valable avec Internet Explorer
- **NOSHADE : désactivation de l'ombrage de la ligne.**

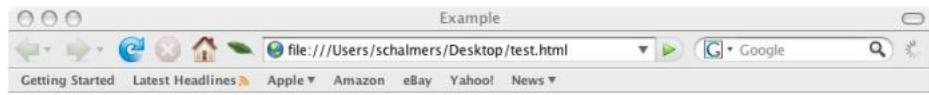
Images

```
<html>
<head>
    <title>Example</title>
</head>
<body>
    <h1>A heading On My First Webpage</h1>
    <p>Here is some <b>text in a paragraph</b></p>
    <p>Here is the <i>next paragraph</i></p>
    <ol>
        <li>First item</li>
        <li>Second item</li>
        <li>Third item</li>
    </ol>
    
</body>
</html>
```



Liens :

```
<html>
<head>
    <title>Example</title>
</head>
<body>
    <h1>A heading On My First Webpage</h1>
    <p>Here is some <b>text in a paragraph</b></p>
    <p>Here is the <i>next paragraph</i></p>
    <ol>
        <li>First item</li>
        <li>Second item</li>
        <li>Third item</li>
    </ol>
    
    <a href="http://fr.wikipedia.org/wiki/CSD">CSD Homepage</a>
</body>
</html>
```



A heading On My First Webpage

Here is some text in a paragraph

Here is the next paragraph

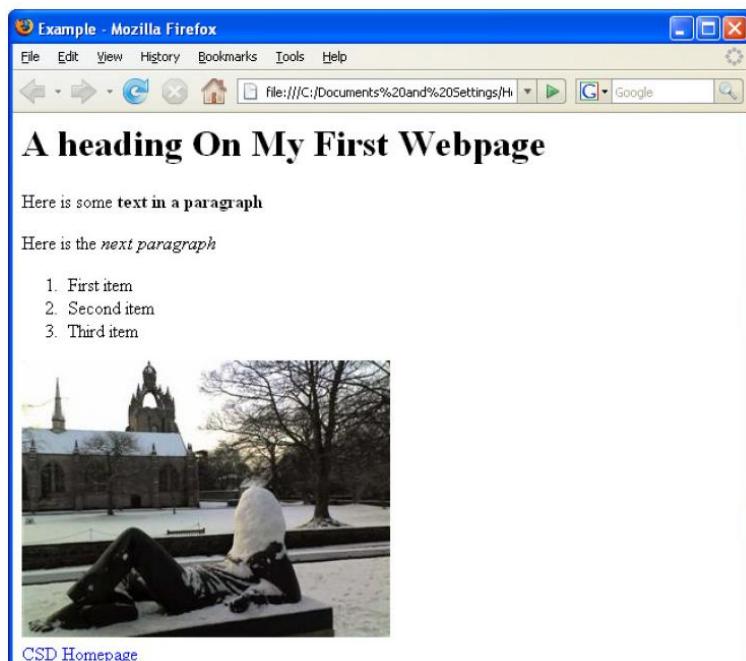
1. First item
2. Second item
3. Third item



[CSD Homepage](http://fr.wikipedia.org/wiki/CSD)

Retour chariot :

```
<html>
<head>
    <title>Example</title>
</head>
<body>
    <h1>A heading On My First Webpage</h1>
    <p>Here is some <b>text in a paragraph</b></p>
    <p>Here is the <i>next paragraph</i></p>
    <ol>
        <li>First item</li>
        <li>Second item</li>
        <li>Third item</li>
    </ol>
    
    <br/>
    <a href="http://fr.wikipedia.org/wiki/CSD">CSD Homepage</a>
</body>
</html>
```



2.1.2 Tables HTML, forms, lists, frames

Voir TD.

2.1.3 Basiques de CSS

Vous pouvez définir des styles CSS directement dans la définition d'une balise (X)HTML. Dans l'exemple ci-dessous, nous utilisons une balise <div> qui permet de définir une "boîte" à l'intérieur d'un contenu :

```
<div style="background-color:orange; border:1px solid black; color:yellow; font-size:150%; padding:1em;">  
    Cette balise div a du style !  
</div>
```

Ce qui donne :



Cette balise div a du style !

Cette approche est extrêmement proche de l'ancienne façon de définir des styles et présente les mêmes inconvénients. Elle ne présente un intérêt que lorsque vous êtes certain que le style défini ne sera utilisé à aucun autre endroit ni sur aucune autre de vos pages. S'il y a la moindre chance pour que vous ayez à nouveau besoin de ce style à un autre endroit, vous devriez absolument utiliser l'une des deux autres méthodes proposées plus bas, afin de faciliter la maintenance et l'évolution de votre site.

Plutôt que par la méthode précédente, il est préférable de définir vos styles CSS une fois pour toute dans une section particulière de votre page Web (on utilise normalement la section <head>).

```
<head>
  <style type="text/css">
    div
    {
      background-color:#339;
      color:#fff;
      padding:15px;
      border-bottom:5px solid red;
      margin-bottom:15px;
    }
  </style>
</head>

<body>
  <div>
    Cette phrase est présentée en fonction du style défini dans l'en-tête
  </div>
  <div>
    Cette phrase aussi, est pourtant le style n'a été défini qu'une fois !
  </div>
</body>
```

Ce qui donne :



Cette phrase est présentée en fonction du style défini dans l'en-tête

Cette phrase aussi, est pourtant le style n'a été défini qu'une fois !

Grâce à cette nouvelle façon de procéder, vous n'avez besoin de définir votre style qu'une seule fois. Dans notre exemple, le style défini s'appliquera automatiquement à toutes les balises <div> de la page. Avec cette méthode, vous pouvez appliquer le même style plusieurs fois dans la même page, mais pas à plusieurs pages d'un coup. Pour aller plus loin dans la standardisation de vos pages, vous devrez utiliser la méthode qui suit.

La façon idéale de définir les CSS consiste à les enregistrer dans un document indépendant de vos pages (X)HTML. Grâce à cette méthode, toutes les pages qui font référence à cette feuille de style externe hériteront de toutes ses définitions.

Un autre intérêt de cette méthode est de pouvoir définir plusieurs feuilles de styles pour le même contenu et de basculer d'une feuille à l'autre en fonction du support sur lequel le contenu est affiché (écran, imprimante, etc.). Nous reviendrons plus tard sur cet aspect.

Une page (X)HTML peut faire référence à plusieurs feuilles de styles en même temps. Dans ce cas, les définitions contenues dans ces différentes feuilles seront combinées entre elles.

Voici un exemple de styles définis dans un document séparé :

Document 'mes-styles.css'

```
body
{
    background-color:#ccf;
    letter-spacing:.1em;
}

p
{
    font-style:italic;
    font-family:times,serif;
}
```

Document 'ma-page.html'

```
<head>
    <link href="mes-styles.css" media="all" rel="stylesheet" type="text/css" />
</head>
<body>
    <p>Voici un exemple de paragraphe.</p>
    <p>Et voici un deuxième paragraphe.</p>
</body>
```

Et voici le résultat :



Voici un exemple de paragraphe.

Et voici un deuxième paragraphe.

La méthode "<link href=..." permet également de mettre en place plusieurs feuilles de styles destinées aux différents medias (imprimante, navigateurs de PDA, etc.). Vous pouvez en effet souhaiter mettre en place une présentation particulière (sans les menus, par exemple) destinée à l'impression de vos documents. Voici une liste des valeurs les plus couramment utilisées pour link :

<pre><link href="general.css" rel="stylesheet" type="text/css" media="all"></pre>	Remplacez <i>general.css</i> par le nom que vous souhaitez donner à votre feuille de style. Cette définition vous permettra de mettre en place une feuille de style commune à tous les medias.
<pre><link href="ecran.css" rel="stylesheet" type="text/css" media="screen, projection"></pre>	Remplacez <i>ecran.css</i> par le nom que vous souhaitez donner à votre feuille de style. Cette définition vous permettra de mettre en place une feuille de style destinée aux écrans.
<pre><link href="mobile.css" rel="stylesheet" type="text/css" media="handheld"></pre>	Remplacez <i>mobile.css</i> par le nom que vous souhaitez donner à votre feuille de style. Cette définition vous permettra de mettre en place une feuille de style destinée aux PDA et téléphones mobiles.
<pre><link href="impression.css" rel="stylesheet" type="text/css" media="print"></pre>	Remplacez <i>impression.css</i> par le nom que vous souhaitez donner à votre feuille de style. Cette définition vous permettra de mettre en place une feuille de style destinée aux imprimantes.

Vous pouvez attribuer à chaque élément (X)HTML une ou plusieurs **classes**. C'est vous qui définirez le nom de ces classes et qui déciderez de leurs styles. Les styles définis dans les classes remplaceront les styles "normaux" des éléments auxquels ils s'appliquent. Pour créer une classe, vous devez simplement faire figurer son nom précédé d'un point. Pour éviter toute ambiguïté, votre nom de classe ne doit pas comporter d'espace.

Cette définition de style est à placer dans une feuille de styles ou dans la section <head> de votre page.

```
.mon-style  
{  
    color:red;  
}
```

Pour appliquer le style défini dans votre classe à un élément, ajouter la mention class="nom-du style" dans la définition de la balise :

```
<p class="mon-style">Le style s'applique à ce paragraphe</p>  
<p>Mais pas à celui-là</p>
```

Cette façon de procéder est très pratique car elle permet d'appliquer les réglages de votre classe à de nombreux éléments, même s'ils ne sont pas du même type :

```
<p class="mon-style">Le style peut s'appliquer à ce paragraphe</p>  
<div class="mon-style">Et aussi à cette balise !</div>
```

Chaque élément (X)HTML peut avoir aucune, une ou plusieurs classes. Pour appliquer plusieurs classes au même élément, précisez simplement la liste de classes en séparant leurs noms par un espace :

Cette définition de styles est à placer dans une feuille de styles ou dans la section <head> de votre page.

```
.mon-style1
{
    color:yellow;
}

.mon-style2
{
    background-color:#A0A0A0;
    font-weight:bold;
}
```

Code (X)HTML associé au CSS

```
<p class="mon-style1 mon-style2">Les styles des deux classes s'appliquent à ce paragraphe</p>
<p class="mon-style2">Alors que ce paragraphe n'a qu'une seule classe </p>
```

Les éléments (X)HTML peuvent se voir attribuer un **id (identification)** en plus ou à la place d'une classe. Il ne doit y avoir qu'un seul élément ayant un **id** donné.

Pour créer un id, vous devez simplement faire précéder son nom d'un dièse #. Pour éviter toute ambiguïté, votre nom d'id ne doit pas comporter d'espace.

Cette définition de style est à placer dans une feuille de styles ou dans la section <head> de votre page.

```
#mon-style
{
    color:red;
}
```

Pour appliquer le style défini dans votre id à un élément, ajouter la mention id="nom-du style" dans la définition de la balise

```
<p id="mon-style">Le style s'applique à ce paragraphe</p>
<p>Mais pas à celui-là</p>
```

Par défaut, chaque type de balise a une présentation particulière dans chaque navigateur. Si vous n'avez défini aucun style particulier pour la balise `<p>`, il est possible que le texte contenu dans ces balises ne se présente pas exactement de la même façon dans tous les navigateurs.

Lorsque vous définissez vous-même le style d'une balise standard avec une définition telle que `p {blablabla}`, vous obligez tous les navigateurs à afficher votre texte de la même façon.

En résumé, voici l'ordre des priorités (ceci est valable pour toutes les balises (X)HTML) :

1. Style standard défini par le navigateur
2. Style standard redéfini en CSS (la dernière définition est prioritaire)
3. Style de classe CSS (la dernière définition est prioritaire)
4. Style d'id (la dernière définition est prioritaire)

Les différents styles attribués aux balises `<body>`, `<div>` et `<p>` vont se combiner pour définir quels sont les styles définitifs qui seront appliqués au texte "bla bla bla".

Si le même style est défini, puis redéfini dans ces différentes balises (la couleur du texte, par exemple), c'est la dernière définition qui l'emportera sur les autres.

Si un style particulier n'est défini que dans `<body>` il doit logiquement s'appliquer à l'ensemble du contenu de `<body>`.

En vérité, certains styles (tels que la couleur, par exemple) se transmettent automatiquement aux balises emboîtées, tandis que d'autres (tels que les marges, par exemple) ne se transmettent pas. Lorsqu'ils se transmettent, on dira que le contenu des balises `<div>` et `<p>` aura "hérité" des propriétés de `<body>`.

2.1.4 HTML5

2.1.5 CSS3

2.2 XML

2.2.1 Structure

2.2.2 DTD, XML schema, XSLT

2.3 JavaScript

Le Javascript est un langage de programmation de scripts orienté objet.

Il existe trois manières d'utiliser du code source :

- **Langage compilé** : le code source est donné à un programme appelé compilateur qui va lire le code source et le convertir dans un langage que l'ordinateur sera capable d'interpréter : c'est le langage binaire, fait de 0 et de 1. Les langages comme le C ou le C++ sont des langages dits compilés.
- **Langage précompilé** : ici, le code source est compilé partiellement, généralement dans un code plus simple à lire pour l'ordinateur, mais qui n'est pas encore du binaire. Ce code intermédiaire devra être lu par ce que l'on appelle une « machine virtuelle », qui exécutera ce code. Les langages comme le C# ou le **Java** sont dits précompilés.
- **Langage interprété** : dans ce cas, il n'y a pas de compilation. Le code source reste tel quel, et si on veut exécuter ce code, on doit le fournir à un interpréteur qui se chargera de le lire et de réaliser les actions demandées.

Les scripts sont majoritairement interprétés. Et quand on dit que le **Javascript est un langage de scripts, cela signifie qu'il s'agit d'un langage interprété** ! Il est donc nécessaire de posséder un interpréteur pour faire fonctionner du code Javascript, et un interpréteur, vous en utilisez un fréquemment : il est **inclus dans votre navigateur Web** !

Chaque navigateur possède un interpréteur Javascript, qui diffère selon le navigateur. Si vous utilisez **Internet Explorer**, son interpréteur Javascript s'appelle **JScript** (l'interpréteur de la version 9 s'appelle Chakra), celui de Mozilla Firefox se nomme SpiderMonkey et celui de Google Chrome est V8.

Le Javascript est à ce jour utilisé majoritairement sur Internet, conjointement avec les pages Web (HTML ou XHTML). Le Javascript s'inclut directement dans la page Web (ou dans un fichier externe) et permet de dynamiser une page HTML, en ajoutant des interactions avec l'utilisateur, des animations, de l'aide à la navigation, comme par exemple :

- *Afficher/masquer du texte* ;
- *Faire défiler des images* ;
- *Créer un diaporama avec un aperçu « en grand » des images* ;
- *Créer des infobulles*.

Le Javascript est un langage dit **client-side**, c'est-à-dire que les scripts sont exécutés par le navigateur chez l'internaute (le client).

Si le Javascript a été conçu pour être utilisé conjointement avec le HTML, le langage a depuis évolué vers d'autres destinées. Le Javascript est régulièrement utilisé pour réaliser **des extensions pour différents programmes**, un peu comme les scripts codés en Python.

Le Javascript peut aussi être utilisé pour réaliser des applications. Mozilla Firefox est l'exemple le plus connu : **l'interface du navigateur est créée avec une sorte de HTML** appelé XUL et c'est le Javascript qui est utilisé pour animer l'interface.

2.3.1 Basiques de JavaScript

Ne dérogeons pas à la règle traditionnelle qui veut que tous les tutoriels de programmation commencent par afficher le texte «**Hello World!** » (« Bonjour le monde ! » en français) à l'utilisateur. Voici un code HTML simple contenant une instruction (nous allons y revenir) Javascript, placée au sein d'un élément <script> :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
  </head>

  <body>
    <script>
      alert('Hello world!');
    </script>
  </body>
</html>
```

Écrivez ce code dans un fichier HTML, et ouvrez ce dernier avec votre navigateur habituel. Une boîte de dialogue s'ouvre, vous présentant le texte « Hello World! » :



2.3.1.1 Indentation

L'indentation, en informatique, est une façon **importante** de structurer du code pour le rendre plus lisible. Les instructions sont hiérarchisées en plusieurs niveaux et on utilise des espaces ou des tabulations pour les décaler vers la droite et ainsi créer une hiérarchie. Voici un exemple de code indenté :

```
function toggle(elemID) {  
    var elem = document.getElementById(elemID);  
  
    if (elem.style.display == 'block') {  
        elem.style.display = 'none';  
    } else {  
        elem.style.display = 'block';  
    }  
}
```

2.3.1.2 Le Javascript « dans la page »

Pour placer du code Javascript directement dans votre page Web, rien de plus simple, on fait comme dans l'exemple du Hello world! : on place le code au sein de l'élément <script>

2.3.1.3 Javascript externe

Il est possible, et même conseillé, d'écrire le code Javascript dans un fichier externe, portant l'extension .js. Ce fichier est ensuite appelé depuis la page Web au moyen de l'élément <script> et de son attribut src qui contient l'URL du fichier .js.

Code : JavaScript - Contenu du fichier hello.js

```
alert('Hello world!');
```

Code : HTML - Page Web

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
  </head>

  <body>

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

  </body>
</html>
```

La plupart des cours de Javascript, et des exemples donnés un peu partout, montrent qu'il faut placer l'élément <script> au sein de l'élément <head> quand on l'utilise pour charger un fichier Javascript. C'est correct, oui, mais il y a mieux !

Une page Web est lue par le navigateur de façon linéaire, c'est-à-dire qu'il lit d'abord le <head>, puis les éléments de <body> les uns à la suite des autres. Si vous appelez un fichier Javascript dès le début du chargement de la page, le navigateur va donc charger ce fichier, et si **ce dernier est volumineux, le chargement de la page s'en trouvera ralenti**. C'est normal puisque le navigateur va charger le fichier avant de commencer à afficher le contenu de la page.

Pour pallier ce problème, il est conseillé de placer les éléments `<script>` juste avant la fermeture de l'élément `<body>`, comme ceci :

Code : HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>Hello World!</title>
  </head>

  <body>

    <p>
      <!--

Contenu de la page Web

...
-->

    </p>

    <script>
      // Un peu de code Javascript...
    </script>

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

  </body>
</html>
```

2.3.1.4 Les variables

Pour faire simple, une variable est un espace de stockage sur votre ordinateur permettant d'enregistrer tout type de donnée, que ce soit une chaîne de caractères, une valeur numérique ou bien des structures un peu plus particulières.

Pour déclarer une variable, il vous suffit d'écrire la ligne suivante :

Code : JavaScript

```
var myVariable;  
myVariable = 2;
```

2.3.1.5 Les types de variables

Contrairement à de nombreux langages, le **Javascript est un langage typé dynamiquement**. Cela veut dire, généralement, que toute déclaration de variable se fait avec le mot-clé var sans distinction du contenu, tandis que dans d'autres langages, comme le C, il est nécessaire de préciser quel type de contenu la variable va devoir contenir.

En Javascript, nos variables sont typées dynamiquement, ce qui veut dire que l'on peut y mettre du texte en premier lieu puis l'effacer et y mettre un nombre quel qu'il soit, et ce, sans contraintes.

Le type **numérique** (alias number) :

```
var number = 0x391;
```

Les **chaînes de caractères** (alias string) :

```
var text1 = "Mon premier texte"; // Avec des guillemets  
var text2 = 'Mon deuxième texte'; // Avec des apostrophes
```

Les **booléens** (alias boolean) :

```
var isTrue = true;  
var isFalse = false;
```

Les **tableaux** :

```
face =  
[  
  ['R', 'G', 'Y'],  
  ['W', 'R', 'O'],  
  ['Y', 'W', 'G']  
]
```

2.3.1.6 Tester l'existence de variables avec `typeof`

Il se peut que vous ayez un jour ou l'autre besoin de tester l'existence d'une variable ou d'en vérifier son type. Dans ce genre de situations, l'instruction `typeof` est très utile, voici comment l'utiliser :

Code : JavaScript

```
var number = 2;
alert(typeof number); // Affiche : « number »

var text = 'Mon texte';
alert(typeof text); // Affiche : « string »

var aBoolean = false;
alert(typeof aBoolean); // Affiche : « boolean »
```

Simple non ? Et maintenant voici comment tester l'existence d'une variable :

Code : JavaScript

```
alert(typeof nothing); // Affiche : « undefined »
```

2.3.1.7 Opérateurs

Table 13-2. Arithmetic operators

Operator	Description	Example
+	Addition	j + 12
-	Subtraction	j - 22
*	Multiplication	j * 7
/	Division	j / 3.13
%	Modulus (division remainder)	j % 6
++	Increment	++j
--	Decrement	--j

Table 13-3. Assignment operators

Operator	Example	Equivalent to
=	j = 99	j = 99
+=	j += 2	j = j + 2
+=	j += 'string'	j = j + 'string'
-=	j -= 12	j = j - 12
*=	j *= 2	j = j * 2
/=	j /= 6	j = j / 6
%=	j %= 7	j = j % 7

Table 13-4. Comparison operators

Operator	Description	Example
==	Is equal to	j == 42
!=	Is not equal to	j != 17
>	Is greater than	j > 0
<	Is less than	j < 100
>=	Is greater than or equal to	j >= 23
<=	Is less than or equal to	j <= 13
==>	Is equal to (and of the same type)	j ==> 56
!==>	Is not equal to (and of the same type)	j !==> '1'

Table 13-5. Logical operators

Operator	Description	Example
&&	And	j == 1 && k == 2
	Or	j < 100 j > 0
!	Not	! (j == k)

Concaténation**Code : JavaScript**

```
var hi = 'Bonjour', name = 'toi', result;
result = hi + name;
alert(result); // Affiche : « Bonjourtoi »
```

Table 13-6. JavaScript's escape characters

Character	Meaning
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\'	Single quote (or apostrophe)
\"	Double quote
\\\	Backslash
\xxx	An octal number between 000 and 377 that represents the Latin-1 character equivalent (such as \251 for the © symbol)
\xxx	A hexadecimal number between 00 and FF that represents the Latin-1 character equivalent (such as \xA9 for the © symbol)
\uxxxx	A hexadecimal number between 0000 and FFFF that represents the Unicode character equivalent (such as \u00A9 for the © symbol)

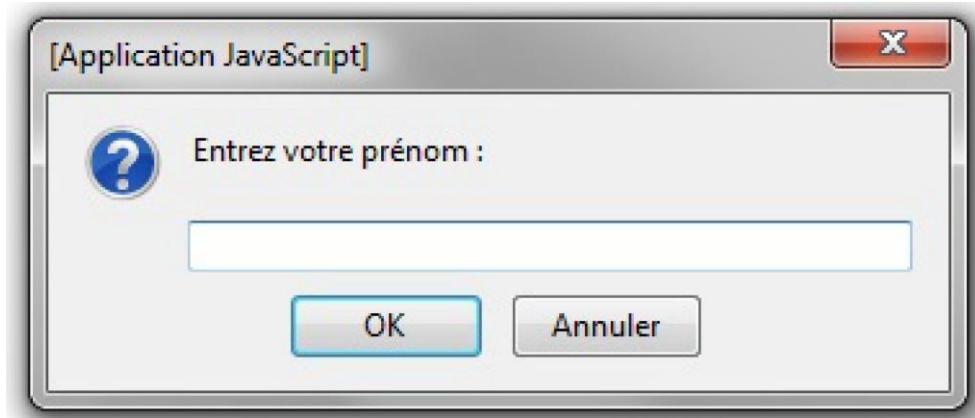
`heading = "Name\tAge\tLocation"`

2.3.1.8 Interagir avec l'utilisateur

La concaténation est le bon moment pour introduire votre toute première interaction avec l'utilisateur grâce à la fonction `prompt()`. Voici comment l'utiliser :

Code : JavaScript

```
var userName = prompt('Entrez votre prénom :');
alert(userName); // Affiche le prénom entré par l'utilisateur
```



Maintenant nous pouvons essayer de dire bonjour à nos visiteurs :

Code : JavaScript

```
var start = 'Bonjour ', name, end = ' !', result;
name = prompt('Quel est votre prénom ?');
result = start + name + end;
alert(result);
```

2.3.1.9 Convertir entre chaînes de caractères et nombres

Code : JavaScript

```
var text = '1337', number;  
  
number = parseInt(text);  
alert(typeof number); // Affiche : « number »  
alert(number); // Affiche : « 1337 »
```

Code : JavaScript

```
var text, number1 = 4, number2 = 2;  
text = number1 + '' + number2;  
alert(text); // Affiche : « 42 »
```

2.3.1.10 Les alternatives

Code : JavaScript

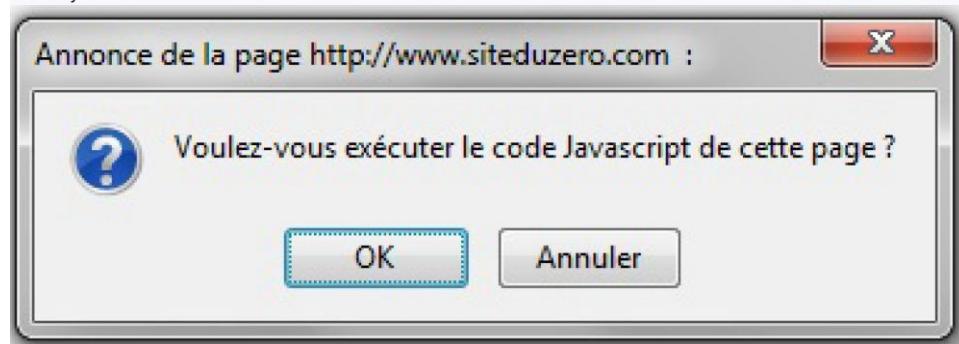
```
if (true) {  
    alert("Ce message s'est bien affiché.");  
}  
  
if (false) {  
    alert("Pas la peine d'insister, ce message ne s'affichera  
pas.");  
}
```

2.3.1.11 Petit intermède : la fonction confirm()

Afin d'aller un petit peu plus loin dans le cours, nous allons apprendre l'utilisation d'une fonction bien pratique : confirm() ! Son utilisation est simple : on lui passe en paramètre une chaîne de caractères qui sera affichée à l'écran et elle retourne un booléen en fonction de l'action de l'utilisateur ; vous allez comprendre en essayant :

Code : JavaScript

```
if (confirm('Voulez-vous exécuter le code Javascript de cette page ?')) {
    alert('Le code a bien été exécuté !');
}
```



2.3.1.12 La condition « switch »

Code : JavaScript

```
var drawer = parseInt(prompt('Choisissez le tiroir à ouvrir (1 à 4) :'));

switch (drawer) {
    case 1:
        alert('Contient divers outils pour dessiner : du papier, des crayons, etc.');
        break;

    case 2:
        alert('Contient du matériel informatique : des câbles, des composants, etc.');
        break;

    case 3:
        alert('Ah ? Ce tiroir est fermé à clé ! Dommage !');
        break;

    case 4:
        alert('Contient des vêtements : des chemises, des pantalons, etc.');
        break;

    default:
        alert("Info du jour : le meuble ne contient que 4 tiroirs et, jusqu'à preuve du contraire, les tiroirs négatifs n'existent pas.");
}
```

2.3.2 Les boucles

Code : JavaScript

```
var number = 1;

while (number < 10) {
    number++;
}

alert(number); // Affiche : « 10 »

<script>
counter=0

while (counter < 5)
{
    document.write("Counter: " + counter + "<br>")
    ++counter
}
</script>

<script>
count = 1

do
{
    document.write(count + " times 7 is " + count * 7 + "<br>")
} while (++count <= 7)
</script>

<script>
for (count = 1 ; count <= 7 ; ++count)
{
    document.write(count + "times 7 is " + count * 7 + "<br>");
}
</script>
```

```
<script>
haystack      = new Array()
haystack[17] = "Needle"

for (j = 0 ; j < 20 ; ++j)
{
  if (haystack[j] == "Needle")
  {
    document.write("<br>- Found at location " + j)
    break
  }
  else document.write(j + ", ")
}
</script>

<script>
haystack      = new Array()
haystack[4]   = "Needle"
haystack[11]  = "Needle"
haystack[17]  = "Needle"

for (j = 0 ; j < 20 ; ++j)
{
  if (haystack[j] == "Needle")
  {
    document.write("<br>- Found at location " + j + "<br>")

    continue
  }
  document.write(j + ", ")
}
</script>
```

2.3.2.1 Portée des variables de boucle

En Javascript, il est **déconseillé de déclarer des variables au sein d'une boucle** (entre les accolades), pour des soucis de performance (vitesse d'exécution) et de logique : il n'y a en effet pas besoin de déclarer une même variable à chaque passage dans la boucle ! Il est conseillé de **déclarer les variables directement dans le bloc d'initialisation**, comme montré dans les exemples de ce cours. **Mais attention : une fois que la boucle est exécutée, la variable existe toujours.** Ce comportement est différent de celui de nombreux autres langages, dans lesquels une variable déclarée dans une boucle est « détruite » une fois la boucle exécutée.

2.3.3 Fonctions, objets et tableaux

Code : JavaScript

```
function myFunction(arguments) {  
    // Le code que la fonction va devoir exécuter  
}
```

2.3.3.1 Variables globales et variables locales

Les variables globales sont **définies à l'extérieur des fonctions (ou à l'intérieur, mais sans le mots clé var)**.

Dans l'exemple suivant "ohai" est une variable globale.

Code : JavaScript

```
var ohai = 'Hello world !';  
  
function sayHello() {  
    alert(ohai);  
}  
  
sayHello();
```

Code : JavaScript

```
var message = 'Ici la variable globale !';  
  
function showMsg() {  
    var message = 'Ici la variable locale !';  
    alert(message);  
}  
  
showMsg();  
  
alert(message);
```

À noter que, dans l'ensemble, il est plutôt déconseillé de créer des variables globales et locales de même nom, cela est souvent source de confusion.

Maintenant que vous savez faire la différence entre les variables globales et locales, il vous faut savoir quand est-ce qu'il est bon d'utiliser l'une ou l'autre. Car malgré le sens pratique des variables globales (vu qu'elles sont accessibles partout) elles sont parfois à proscrire car **elles peuvent rapidement vous perdre dans votre code** (et engendrer des problèmes si vous souhaitez partager votre code, mais vous découvrirez cela par vous-même). Voici un exemple de ce qu'il ne faut pas faire :

Code : JavaScript

```
var var1 = 2, var2 = 3;

function calculate() {
    alert(var1 * var2); // Affiche : « 6 » (sans blague ?!)
}

calculate();
```

Dans ce code, vous pouvez voir que les variables var1 et var2 ne sont utilisées que pour la fonction calculate() et pour rien d'autre, or ce sont ici des variables globales. Par principe, cette façon de faire est stupide : vu que ces variables ne servent qu'à la fonction calculate(), autant les déclarer dans la fonction de la manière suivante :

Code : JavaScript

```
function calculate() {
    var var1 = 2, var2 = 3;
    alert(var1 * var2);
}

calculate();
```

2.3.3.2 Les arguments et les valeurs de retour

Code : JavaScript

```
function myFunction(arg) { // Notre argument est la variable « arg »  
    // Une fois que l'argument a été passé à la fonction, vous  
    // allez le retrouver dans la variable « arg »  
    alert('Votre argument : ' + arg);  
}  
  
myFunction('En voilà un beau test !');
```

Code : JavaScript

```
function moar(first, second) {  
    // On peut maintenant utiliser les variables « first » et «  
    // second » comme on le souhaite :  
    alert('Votre premier argument : ' + first);  
    alert('Votre deuxième argument : ' + second);  
}
```

Code : JavaScript

```
function sayHello() {  
    return 'Bonjour !'; // L'instruction « return » suivie d'une  
    // valeur, cette dernière est donc renvoyée par la fonction  
}  
  
alert(sayHello()); // Ici on affiche la valeur renournée par la  
// fonction sayHello()
```

2.3.3.3 Les fonctions anonymes

Après les fonctions, voici les fonctions anonymes ! Ces fonctions particulières sont extrêmement importantes en Javascript ! Elles vous serviront pour énormément de choses : les objets, les évènements, les variables statiques, les closures, etc.

Pour assigner une fonction anonyme à une variable, rien de plus simple :

Code : JavaScript

```
var sayHello = function() {  
    alert('Bonjour !');  
};
```

Code : JavaScript

```
sayHello(); // Affiche : « Bonjour ! »
```

Code : JavaScript

```
var sayHello = (function() {  
    return 'Yop !';  
})();  
  
alert(sayHello); // Affiche : « Yop ! »
```

Code : JavaScript

```
sayHello(); // Affiche : « Bonjour ! »
```

2.3.4 Les objets et les tableaux

Il a été dit précédemment que le **Javascript est un langage orienté objet**. Cela veut dire que le langage dispose d'objets.

Un objet est un concept, une idée ou une chose. Un objet possède une structure qui lui permet de pouvoir fonctionner et d'interagir avec d'autres objets. Le Javascript met à notre disposition des objets natifs, c'est-à-dire des objets directement utilisables. Vous avez déjà manipulé de tels objets sans le savoir : un nombre, une chaîne de caractères ou même un booléen.

Si, mais en réalité, une variable contient surtout un objet. Par exemple, si nous créons une chaîne de caractères, comme ceci :

Code : JavaScript

```
var myString = 'Ceci est une chaîne de caractères';
```

Code : JavaScript

```
var myString = 'Ceci est une chaîne de caractères'; // On crée un  
objet String  
  
alert(myString.length); // On affiche le nombre de caractères, au  
moyen de la propriété « length »  
  
alert(myString.toUpperCase()); // On récupère la chaîne en  
majuscules, avec la méthode toUpperCase()
```

La méthode push() permet d'ajouter un ou plusieurs items à un tableau :

Code : JavaScript

```
var myArray = ['Sébastien', 'Laurence'];  
  
myArray.push('Ludovic'); // Ajoute « Ludovic » à la fin du tableau  
myArray.push('Pauline', 'Guillaume'); // Ajoute « Pauline » et «  
Guillaume » à la fin du tableau
```

2.3.4.1 Les objets littéraux

S'il est possible d'accéder aux items d'un tableau via leur indice, il peut être pratique d'y accéder au moyen d'un identifiant. Par exemple, dans le tableau des prénoms, l'item appelé **sister** pourrait retourner la valeur « Laurence ». Pour ce faire, nous allons créer nous-mêmes un tableau sous la forme d'un objet littéral. Voici un exemple :

Code : JavaScript

```
var family = {  
    self:      'Sébastien',  
    sister:    'Laurence',  
    brother:   'Ludovic',  
    cousin_1:  'Pauline',  
    cousin_2:  'Guillaume'  
};
```

Code : JavaScript

```
family.sister;
```

Code : JavaScript

```
family['sister'];
```

Ici, pas de méthode push() car elle n'existe tout simplement pas dans un objet vide, il faudrait pour cela un tableau. En revanche, il est possible **d'ajouter un item en spécifiant un identifiant** qui n'est pas encore présent. Par exemple, si nous voulons ajouter un oncle dans le tableau :

Code : JavaScript

```
family['uncle'] = 'Didier'; // « Didier » est ajouté et est accessible via l'identifiant « uncle »
```

Parcourir un objet avec for in : il suffit de fournir une « variable clé » qui reçoit un identifiant (au lieu d'un index) et de spécifier l'objet à parcourir :

Code : JavaScript

```
for (var id in family) { // On stocke l'identifiant dans « id » pour parcourir l'objet « family »  
    alert(family[id]);  
}
```

Les objets littéraux ne sont pas souvent utilisés mais peuvent se révéler très utiles pour ordonner un code. On les utilise aussi dans les fonctions : les fonctions, avec return, ne savent retourner qu'une seule variable. Si on veut retourner plusieurs variables, il faut les placer dans un tableau et retourner ce dernier. Mais il est plus commode d'utiliser un objet littéral.

Code : JavaScript

```
function getCoords() {  
    /* Script incomplet, juste pour l'exemple */  
  
    return { x: 12, y: 21 };  
}  
  
var coords = getCoords();  
  
alert(coords.x); // 12  
alert(coords.y); // 21
```

2.3.5 Modeler vos pages Web (DHTML et DOM)

Le Javascript est un langage qui permet de créer ce que l'on appelle des pages DHTML. Ce terme désigne les pages Web qui modifient elles-mêmes leur propre contenu sans charger de nouvelle page. C'est cette modification de la structure d'une page Web que nous allons étudier dans cette partie.

Dans cette section consacrée à la manipulation du code HTML, nous allons voir le concept du DOM. Nous étudierons tout d'abord comment naviguer entre les différents nœuds qui composent une page HTML puis nous aborderons l'édition du contenu d'une page en ajoutant, modifiant et supprimant des nœuds.

2.3.5.1 Le Document Object Model

Le **Document Object Model** (abrégé **DOM**) est une interface de programmation pour les documents XML et HTML.

Le DOM est donc une API qui s'utilise avec les documents XML et HTML, et qui **va nous permettre, via le Javascript, d'accéder au code XML et/ou HTML d'un document**. C'est grâce au DOM que nous allons pouvoir modifier des éléments HTML (afficher ou masquer un <div> par exemple), en ajouter, en déplacer ou même en supprimer.

2.3.5.2 L'objet window

Avant de véritablement parler du document, c'est-à-dire de la page Web, nous allons parler de l'objet window. L'objet window est ce qu'on appelle un objet global qui représente la fenêtre du navigateur. C'est à partir de cet objet que le Javascript est exécuté.

Si nous reprenons notre « Hello World! » du début, nous avons :

Code : HTML

```
<!DOCTYPE html>

<html>
  <head>
    <title>Hello World!</title>
  </head>

  <body>

    <script>
      alert('Hello world!');
    </script>

  </body>
</html>
```

Contrairement à ce qui a été dit dans ce cours, **alert()** n'est pas vraiment une fonction. Il s'agit en réalité d'une méthode appartenant à l'objet window. Mais l'objet window est dit implicite, c'est-à-dire qu'il n'y a généralement pas besoin de le spécifier. Ainsi, ces deux instructions produisent le même effet, à savoir ouvrir une boîte de dialogue :

Code : JavaScript

```
window.alert('Hello world!');  
alert('Hello world!');
```

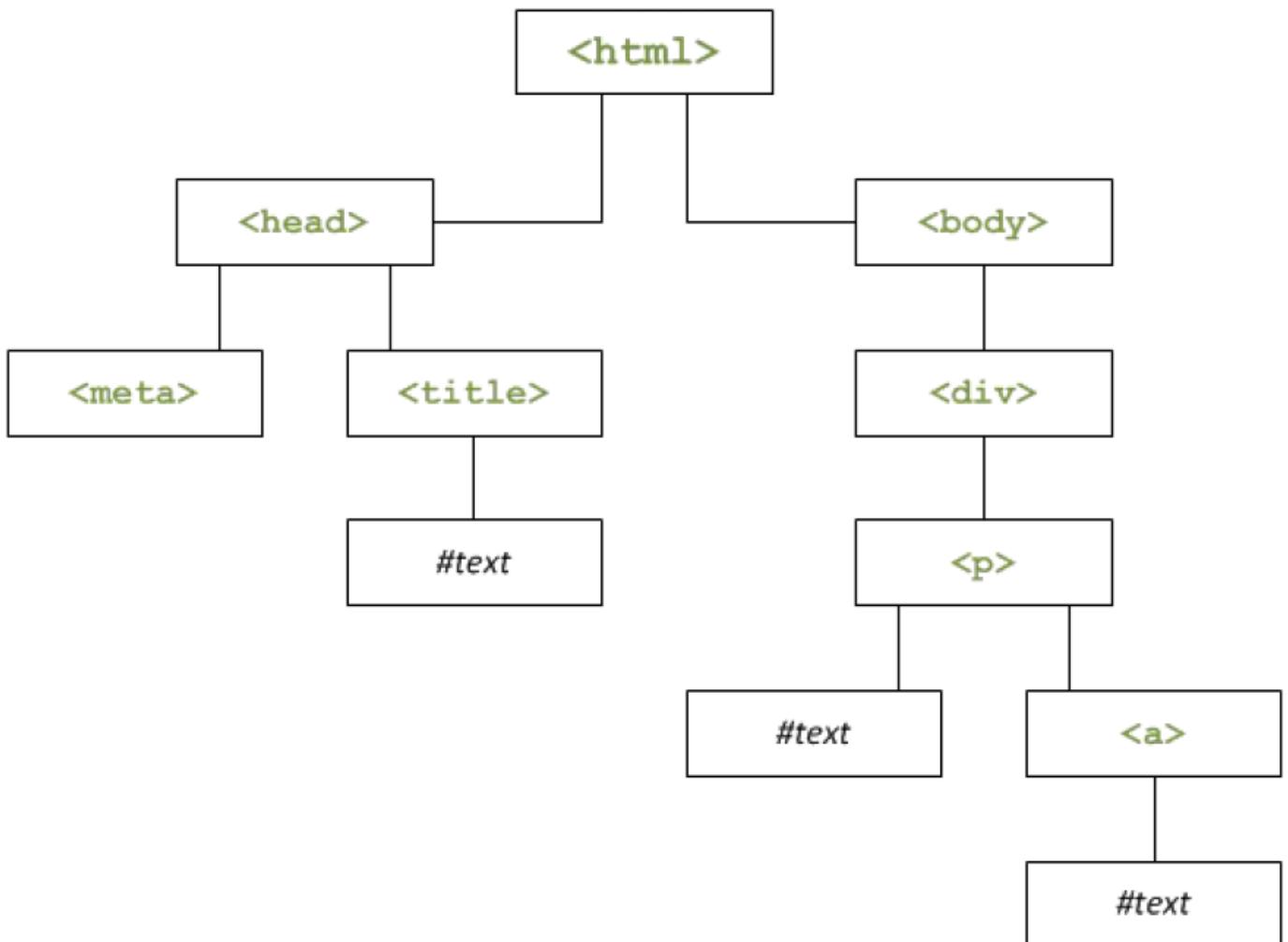
2.3.5.3 Le document

L'objet **document** est un sous-objet de **window**, l'un des plus utilisés. Et pour cause, il représente la page Web et plus précisément la balise <html>. C'est grâce à cet élément-là que nous allons pouvoir accéder aux éléments HTML et les modifier.

Voyons donc, dans la sous-partie suivante, comment naviguer dans le document.

2.3.5.4 Naviguer dans le document : La structure DOM

Comme il a été dit précédemment, le DOM pose comme concept que la page Web est vue comme un arbre, comme une hiérarchie d'éléments. On peut donc schématiser une page Web simple comme ceci :



Code : HTML

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
  </head>

  <body>
    <div>
      <p>Un peu de texte <a>et un lien</a></p>
    </div>
  </body>
</html>
```

Le schéma est plutôt simple : l'élément `<html>` contient deux éléments, appelés enfants : `<head>` et `<body>`. Pour ces deux enfants, `<html>` est l'élément parent. Chaque élément est appelé nœud (node en anglais). L'élément `<head>` contient lui aussi deux enfants : `<meta>` et `<title>`. `<meta>` ne contient pas d'enfant tandis que `<title>` en contient un, qui s'appelle `#text`. Comme son nom l'indique, `#text` est un élément qui contient du texte.

2.3.5.5 Accéder aux éléments : `getElementById()`

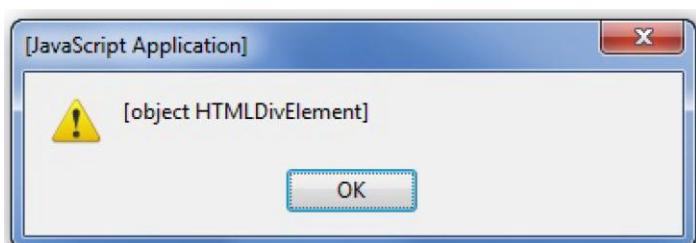
La méthode `getElementById()` permet d'accéder à un élément en connaissant son ID qui est simplement l'attribut `id` de l'élément. Cela fonctionne de cette manière :

Code : HTML

```
<div id="myDiv">
  <p>Un peu de texte <a>et un lien</a></p>
</div>

<script>
  var div = document.getElementById('myDiv');

  alert(div);
</script>
```



2.3.5.6 *getElementsByTagName()*

Cette méthode permet de récupérer, sous la forme d'un tableau, tous les éléments de la famille. Si, dans une page, on veut récupérer tous les <div>, il suffit de faire comme ceci :

```
var divs = document.getElementsByTagName('div');

for (var i = 0, c = divs.length ; i < c ; i++) {
    alert('Element n° ' + (i + 1) + ' : ' + divs[i]);
}
```

2.3.5.7 Accéder aux éléments grâce aux technologies récentes

Les deux méthodes querySelector() et querySelectorAll() ont pour particularité de grandement simplifier la sélection d'éléments dans l'arbre DOM grâce à leur mode de fonctionnement. Ces deux méthodes prennent pour paramètre un seul argument : une chaîne de caractères !

Cette chaîne de caractères doit être un sélecteur CSS comme ceux que vous utilisez dans vos feuilles de style. Exemple :

Code : CSS

```
#menu .item span
```

Ce sélecteur CSS stipule que l'on souhaite sélectionner les balises de type contenues dans les classes .item elles-mêmes contenues dans un élément dont l'identifiant est #menu.

Code : HTML

```
<div id="menu">

  <div class="item">
    <span>Élément 1</span>
    <span>Élément 2</span>
  </div>

  <div class="publicite">
    <span>Élément 3</span>
    <span>Élément 4</span>
  </div>

</div>

<div id="contenu">
  <span>Introduction au contenu de la page...</span>
</div>
```

Code : JavaScript

```
var query = document.querySelector('#menu .item span'),
queryAll = document.querySelectorAll('#menu .item span');

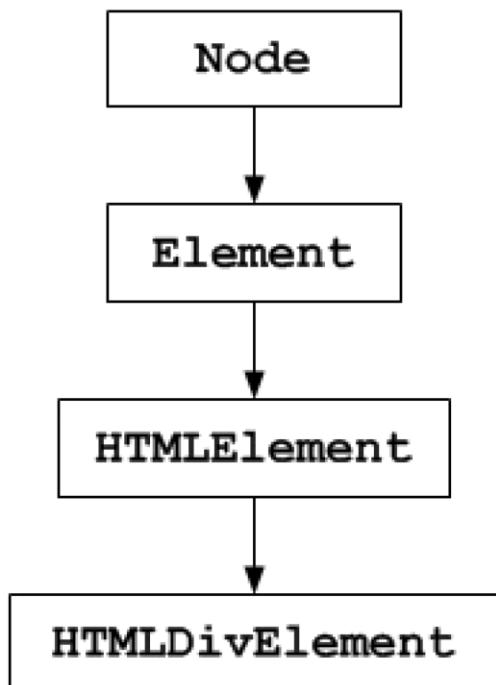
alert(query.innerHTML); // Affiche : "Élément 1"

alert(queryAll.length); // Affiche : "2"
alert(queryAll[0].innerHTML + ' - ' + queryAll[1].innerHTML); // 
Affiche : "Élément 1 - Élément 2"
```

2.3.5.8 L'héritage des propriétés et des méthodes

Le Javascript voit les éléments HTML comme étant des objets, cela veut donc dire que chaque élément HTML possède des propriétés et des méthodes. Cependant faites bien attention parce que tous ne possèdent pas les mêmes propriétés et méthodes. Certaines sont néanmoins communes à tous les éléments HTML, car tous les éléments HTML sont d'un même type : le type Node, qui signifie « nœud » en anglais.

Nous avons vu qu'un élément <div> est un objet HTMLDivElement, mais un objet, en Javascript, peut appartenir à différents groupes. Ainsi, notre <div> est un HTMLDivElement, qui est un sous-objet d'HTMLElement qui est lui-même un sous-objet d'Element. Element est enfin un sous-objet de Node. Ce schéma est plus parlant :



2.3.5.9 Éditer les éléments HTML

Maintenant que nous avons vu comment accéder à un élément, nous allons voir comment l'éditer. Les éléments HTML sont souvent composés d'attributs (l'attribut href d'un par exemple), et d'un contenu, qui est de type #text. Le contenu peut aussi être un autre élément HTML.

Pour interagir avec les attributs, l'objet Element nous fournit deux méthodes, getAttribute() et setAttribute() permettant respectivement de récupérer et d'éditer un attribut. Le premier paramètre est le nom de l'attribut, et le deuxième, dans le cas de setAttribute() uniquement, est la nouvelle valeur à donner à l'attribut. Petit exemple :

Code : HTML

```
<body>
  <a id="myLink" href="http://www.un_lien_quelconque.com">Un lien
  modifié dynamiquement</a>

  <script>
    var link = document.getElementById('myLink');
    var href = link.getAttribute('href'); // On récupère l'attribut
    « href »
    alert(href);

    link.setAttribute('href', 'http://www.siteduzero.com'); // On
    édite l'attribut « href »
  </script>
</body>
```

En fait, pour la plupart des éléments courants comme , il est possible d'accéder à un attribut via une propriété. Ainsi, si on veut modifier la destination d'un lien, on peut utiliser la propriété href, comme ceci :

Code : HTML

```
<body>
  <a id="myLink" href="http://www.un_lien_quelconque.com">Un lien
  modifié dynamiquement</a>

  <script>
    var link = document.getElementById('myLink');
    var href = link.href;

    alert(href);

    link.href = 'http://www.siteduzero.com';
  </script>
</body>
```

2.3.5.10 La classe

On peut penser que pour modifier l'attribut class d'un élément HTML, il suffit d'utiliser element.class. Ce n'est pas possible, car le mot-clé class est réservé en Javascript, bien qu'il n'ait aucune utilité. À la place de class, il faudra utiliser className.

Code : HTML

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
    <style type="text/css">
      .blue {
        background: blue;
        color: white;
      }
    </style>
  </head>

  <body>
    <div id="myColoredDiv">
      <p>Un peu de texte <a>et un lien</a></p>
    </div>

    <script>
      document.getElementById('myColoredDiv').className = 'blue';
    </script>
  </body>
</html>
```

2.3.5.11 Le contenu : innerHTML

innerHTML permet de récupérer le code HTML enfant d'un élément sous forme de texte. Ainsi, si des balises sont présentes, innerHTML les retournera sous forme de texte :

Code : HTML

```
<body>
  <div id="myDiv">
    <p>Un peu de texte <a>et un lien</a></p>
  </div>

  <script>
    var div = document.getElementById('myDiv');

    alert(div.innerHTML);
  </script>
</body>
```



Pour éditer ou ajouter du contenu HTML, il suffit de faire l'inverse, c'est-à-dire de définir un nouveau contenu :

Code : JavaScript

```
document.getElementById('myDiv').innerHTML = '<blockquote>Je mets  
une citation à la place du paragraphe</blockquote>';
```

Si vous voulez ajouter du contenu, et ne pas modifier le contenu déjà en place, il suffit d'utiliser += à la place de l'opérateur d'affectation :

Code : JavaScript

```
document.getElementById('myDiv').innerHTML += ' et <strong>une  
portion mise en emphase</strong>.';
```

Toutefois, une petite mise en garde : il ne faut pas utiliser le `+=` dans une boucle ! En effet, `innerHTML` ralentit considérablement l'exécution du code si l'on opère de cette manière, il vaut donc mieux concaténer son texte dans une variable pour ensuite ajouter le tout via `innerHTML`. Exemple :

Code : JavaScript

```
var text = '';  
  
while( /* condition */ ) {  
    text += 'votre_texte'; // On concatène dans la variable « text »  
}  
  
element.innerHTML = text; // Une fois la concaténation terminée, on  
ajoute le tout à « element » via innerHTML
```

2.3.5.12 Naviguer entre les nœuds : La propriété `parentNode`

Nous avons vu précédemment qu'on utilisait les méthodes `getElementById()` et `getElementsByName()` pour accéder aux éléments HTML. Une fois que l'on a atteint un élément, il est possible de se déplacer de façon un peu plus précise, avec toute une série de méthodes et de propriétés que nous allons étudier ici.

Code : HTML

```
<blockquote>  
    <p id="myP">Ceci est un paragraphe !</p>  
</blockquote>
```

Admettons qu'on doive accéder à `myP`, et que pour une autre raison on doive accéder à l'élément `<blockquote>`, qui est le parent de `myP`. Il suffit d'accéder à `myP` puis à son parent, avec `parentNode` :

Code : JavaScript

```
var paragraph = document.getElementById('myP');  
var blockquote = paragraph.parentNode;
```

2.3.5.13 *nodeType et nodeName*

nodeType et nodeName servent respectivement à vérifier le type d'un nœud et le nom d'un nœud. nodeType retourne un nombre, qui correspond à un type de nœud. Voici un tableau qui liste les types possibles, ainsi que leurs numéros (les types courants sont mis en gras) :

Numéro	Type de nœud
1	Nœud élément
2	Nœud attribut
3	Nœud texte
4	Nœud pour passage CDATA (relatif au XML)
5	Nœud pour référence d'entité
6	Nœud pour entité
7	Nœud pour instruction de traitement
8	Nœud pour commentaire
9	Nœud document
10	Nœud type de document
11	Nœud de fragment de document
12	Nœud pour notation

nodeName, quant à lui, retourne simplement le nom de l'élément, en majuscule. Il est toutefois conseillé d'utiliser toLowerCase() (ou toUpperCase()) pour forcer un format de casse et ainsi éviter les mauvaises surprises.

Code : JavaScript

```
var paragraph = document.getElementById('myP');
alert(paragraph.nodeType + '\n\n' +
paragraph.nodeName.toLowerCase());
```

2.3.5.14 Lister et parcourir des nœuds enfants

Comme leur nom le laisse présager, firstChild et lastChild servent respectivement à accéder au premier et au dernier enfant d'un nœud.

Code : HTML

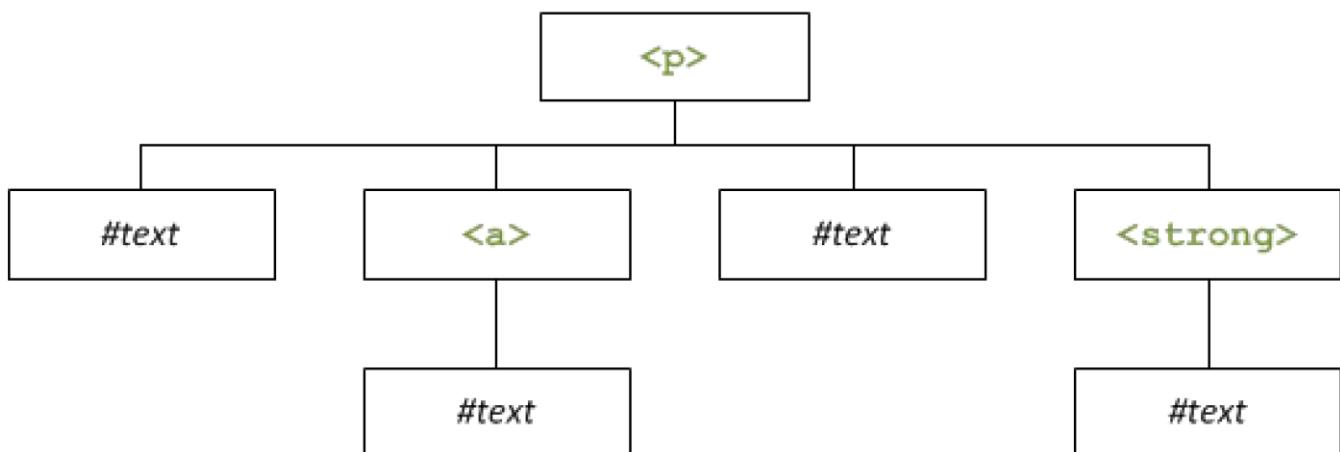
```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
  </head>

  <body>
    <div>
      <p id="myP">Un peu de texte, <a>un lien</a> et <strong>une
portion en emphase</strong></p>
    </div>

    <script>
      var paragraph = document.getElementById('myP');
      var first = paragraph.firstChild;
      var last = paragraph.lastChild;

      alert(first.nodeName.toLowerCase());
      alert(last.nodeName.toLowerCase());
    </script>
  </body>
</html>
```

En schématisant l'élément myP précédent, on obtient ceci :



Le premier enfant de <p> est un nœud textuel, alors que le dernier enfant est un élément .

2.3.5.15 *nodeValue et data*

Changeons de problème : il faut récupérer le texte du premier enfant, et le texte contenu dans l'élément , mais comment faire ? Il faut soit utiliser la propriété `nodeValue` soit la propriété `data`. Si on recode le script précédent, nous obtenons ceci :

Code : JavaScript

```
var paragraph = document.getElementById('myP');
var first = paragraph.firstChild;
var last = paragraph.lastChild;

alert(first.nodeValue);
alert(last.firstChild.data);
```

2.3.5.16 *childNodes*

La propriété `childNodes` retourne un tableau contenant la liste des enfants d'un élément. L'exemple suivant illustre le fonctionnement de cette propriété, de manière à récupérer le contenu des éléments enfants :

Code : HTML

```
<body>
  <div>
    <p id="myP">Un peu de texte <a>et un lien</a></p>
  </div>

<script>
  var paragraph = document.getElementById('myP');
  var children = paragraph.childNodes;

  for (var i = 0, c = children.length; i < c; i++) {

    if (children[i].nodeType === 1) { // C'est un élément HTML
      alert(children[i].firstChild.data);
    } else { // C'est certainement un nœud textuel
      alert(children[i].data);
    }
  }
</script>
</body>
```

2.3.5.17 *Créer et insérer des éléments : Ajouter des éléments HTML*

Avec le DOM, l'ajout d'un élément HTML se fait en trois temps :

- On crée l'élément ;
- On lui affecte des attributs ;
- On l'insère dans le document, et ce n'est qu'à ce moment-là qu'il sera « ajouté ».

La création d'un élément se fait avec la méthode createElement(), un sous-objet de l'objet racine, c'est-à-dire document dans la majorité des cas :

Code : JavaScript

```
var newLink = document.createElement('a');
```

Ici, c'est comme nous avons vu précédemment : on définit les attributs, soit avec setAttribute(), soit directement avec les propriétés adéquates.

Code : JavaScript

```
newLink.id      = 'sdz_link';
newLink.href    = 'http://www.siteduzero.com';
newLink.title   = 'Découvrez le Site du Zéro !';
newLink.setAttribute('tabindex', '10');
```

On utilise la méthode `appendChild()` pour insérer l'élément. Append child signifie « ajouter un enfant », ce qui signifie qu'il nous faut connaître l'élément auquel on va ajouter l'élément créé. Considérons donc le code suivant :

Code : HTML

```
<!doctype html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
  </head>

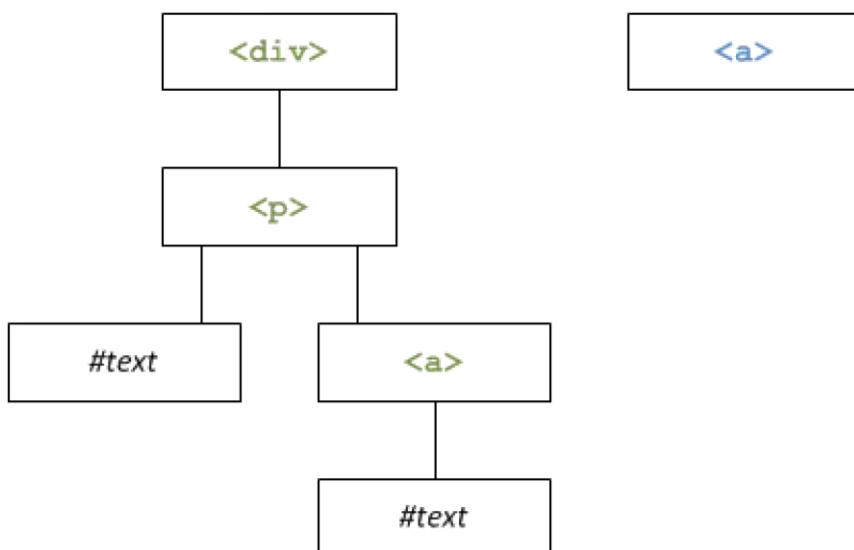
  <body>
    <div>
      <p id="myP">Un peu de texte <a>et un lien</a></p>
    </div>
  </body>
</html>
```

On va ajouter notre élément `<a>` dans l'élément `<p>` portant l'ID myP. Pour ce faire, il suffit de récupérer cet élément, et d'ajouter notre élément `<a>` via `appendChild()` :

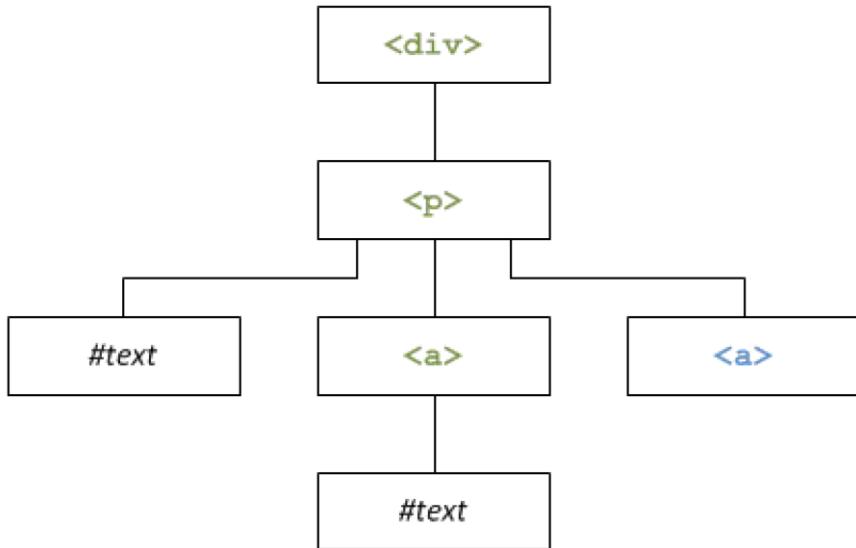
Code : JavaScript

```
document.getElementById('myP').appendChild(newLink);
```

Une petite explication s'impose ! Avant d'insérer notre élément `<a>`, la structure DOM du document ressemble à ceci :



On voit que l'élément `<a>` existe, mais n'est pas lié. Un peu comme s'il était libre dans le document : il n'est pas encore placé. Le but est de le placer comme enfant de l'élément `myP`. La méthode `appendChild()` va alors déplacer notre `<a>` pour le placer en tant que dernier enfant de `myP` :



2.3.5.18 Cloner, remplacer, supprimer...

Pour cloner un élément, rien de plus simple : `cloneNode()`. Cette méthode requiert un paramètre booléen (`true` ou `false`) : si vous désirez cloner le nœud avec (`true`) ou sans (`false`) ses enfants et ses différents attributs.

Petit exemple très simple : on crée un élément `<hr />`, et on en veut un deuxième, donc on clone le premier :

```

// On va cloner un élément créé :
var hr1 = document.createElement('hr');
var hr2 = hr1.cloneNode(false); // Il n'a pas d'enfants...

// Ici, on clone un élément existant :
var paragraph1 = document.getElementById('myP');
var paragraph2 = paragraph1.cloneNode(true);

// Et attention, l'élément est cloné, mais pas « inséré » tant que
// l'on n'a pas appelé appendChild() :
paragraph1.parentNode.appendChild(paragraph2);
  
```

2.3.5.19 Remplacer un élément par un autre

Dans l'exemple suivant, le contenu textuel (pour rappel, il s'agit du premier enfant de <a>) du lien va être remplacé par un autre. La méthode replaceChild() est exécutée sur l'élément <a>, c'est-à-dire le nœud parent du nœud à remplacer.

Code : HTML

```
<body>
  <div>
    <p id="myP">Un peu de texte <a>et un lien</a></p>
  </div>

  <script>
    var link = document.getElementsByTagName('a')[0];
    var newLabel= document.createTextNode('et un hyperlien');

    link.replaceChild(newLabel, link.firstChild);
  </script>
</body>
```

2.3.5.20 Supprimer un élément

Pour insérer un élément, on utilise appendChild(), et pour en supprimer un, on utilise removeChild(). Cette méthode prend en paramètre le nœud enfant à retirer. Si on se calque sur le code HTML de l'exemple précédent, le script ressemble à ceci :

Code : JavaScript

```
var link = document.getElementsByTagName('a')[0];

link.parentNode.removeChild(link);
```

À noter que la méthode removeChild() retourne l'élément supprimé, ce qui veut dire qu'il est parfaitement possible de supprimer un élément HTML pour ensuite le réintégrer où on le souhaite dans le DOM :

Code : JavaScript

```
var link = document.getElementsByTagName('a')[0];

var oldLink = link.parentNode.removeChild(link); // On supprime
// l'élément et on le garde en stock

document.body.appendChild(oldLink); // On réintègre ensuite
// l'élément supprimé où on veut et quand on veut
```

2.3.5.21 Insérer à la bonne place : `insertBefore()`

La méthode `insertBefore()` permet d'insérer un élément avant un autre. Elle reçoit deux paramètres : le premier est l'élément à insérer, tandis que le deuxième est l'élément avant lequel l'élément va être inséré. Exemple :

Code : HTML

```
<p id="myP">Un peu de texte <a>et un lien</a></p>

<script>
  var paragraph = document.getElementsByTagName('p')[0];
  var emphasis = document.createElement('em'),
      emphasisText = document.createTextNode(' en emphase légère ');
  emphasis.appendChild(emphasisText);

  paragraph.insertBefore(emphasis, paragraph.lastChild);
</script>
```


2.3.6 Interaction back-end via Ajax

2.3.7 La bibliothèque jquery

2.4 Framework Angular

2.4.1 Basiques de Angular et le langage typescript

2.4.2 Premier exemple

2.4.3 Composants maître/détails

2.4.4 Services

2.4.5 Routing

2.4.6 HTTP

2.5 Exemples d'application

2.5.1 Application 1

2.5.2 Application 2

2.5.3 ...

3 Programmation côté serveur

3.1 PHP

3.1.1 Basiques de PHP

3.1.2 Expressions et flots de contrôle

3.1.3 Fonctions et objets

3.1.4 Tableaux

3.1.5 Pratiques PHP

3.1.6 Manipulation des formes

3.1.7 Cookies, sessions, et authentification

3.2 Connexion BDD

3.2.1 MySQL

3.2.2 Postgres

3.2.3 Accès BDD via PHP

3.3 Framework Laravel

3.4 Exemples d'application

3.4.1 Application 1

3.4.2 Application 2

3.4.3 ...

4 Services Web

4.1 Introduction

4.2 SOA

4.3 Standards des services Web

4.4 Plateformes de développement

4.5 Plateformes .NET et Java

4.5.1 Application 1

4.5.2 Application 2

4.5.3 ...

5 Fiche TP n°1 : Prise en main HTML/CSS

[TP HTML, EMP, (<https://geometrica.saclay.inria.fr/team/Marc.Glisson>)]

[CSS : vos premiers pas, (<http://www.css-faciles.com/premiers-pas-css.php>)]

Nous allons commencer par créer un document html très petit, et nous l'enrichirons au fur et à mesure. Lancez d'abord notepad (c'est l'éditeur de texte par défaut sous windows), et recopiez le texte suivant :

```
<html>
<head>
<meta name="author" content="Mourad Bachir">
<!-- je suis bête, j'ai oublié de modifier le nom de l'auteur -->
<title>
    Premier essai
</title>
</head>
<body>
<p>
    Bonjour tout le monde...
    <!-- je ne sais pas trop quoi écrire
    alors je fais plein de commentaires
    qui ne vont pas s'afficher -->
    <hr>
</body>
</html>
```

À l'aide d'un navigateur (chrome, firefox, IE, par exemple), regardez votre page. Nous allons la modifier progressivement (n'hésitez pas à faire d'autres changements que ceux proposés, en vous en inspirant), et il faudra vérifier à chaque nouvelle étape à quoi ressemble maintenant votre page.

5.1 Des couleurs

Remplacez la ligne `<body>` par la suivante :

```
<body bgcolor="#00C0FF" text=red>
```

Prenez une image quelconque nommez la "mine.gif" (ou un nom similaire). En fait n'importe quelle image fera l'affaire. Copiez-là dans le répertoire de ce TP, et essayez à la place :

```
<body background="mine.gif">
```

Ne soyez pas surpris si cela devient complètement illisible, c'est un piège classique des fonds d'écran.

5.2 Paragraphes

Ecrivez deux paragraphes de texte, en sautant des lignes entre les deux paragraphes. Par exemple, reprenons à Bonjour tout le monde... :

Bonjour tout le monde... Ceci est un premier paragraphe. J'essaie de le faire suffisamment long pour qu'il y ait au moins un retour à la ligne, sinon

Et maintenant j'ai sauté plein de lignes, que se passe-t-il ?

Vous pouvez supprimer tous ces sauts de ligne, et mettre à la place `<p>`. Essayez aussi de mettre `
` à la place. `<p>` indique un changement de paragraphe, alors que `
` est un simple retour à la ligne. En général, vous devriez donc plus utiliser `<p>`.

5.3 Titres

Changeons un peu le texte de notre page, pour quelque chose comme :

```
<h1 align=center>Voici un titre</h1>
<h2>Maintenant un sous-titre</h2>
<h3>Et on peut continuer</h3>
<h4>La profondeur est limitée</h4>
<h5>Pénultième</h5>
<p align=right>Un petit texte justifié à droite pour introduire
cette partie, elle le mérite bien, et puis il serait dommage de se
priver.
<h6>Un vraiment petit paragraphe</h6>
Et un titre est un changement de paragraphe implicite.
```

5.4 Paragraphes spéciaux

On ajoute encore :

```
<blockquote>
Mignonne, allons voir si la rose<br>
Qui ce matin avoit desclose<br>
Sa robe de pourpre au Soleil,<br>
A point perdu ceste vesprée<br>
Les plis de sa robe pourprée,<br>
Et son teint au vostre pareil.
</blockquote>
<pre>
< &lt;html&gt;
  &lt;body&gt;
    &lt;/body&gt;
  &lt;/html&gt;
</pre>
```

On utilise `blockquote` pour des citations longues, et `pre` pour du texte préformaté, comme du code dans un langage de programmation. Quelles différences de rendu observez-vous ? Que se passe-t-il si au lieu d'écrire `<` et `>` on écrit directement `<` et `>` ?

5.5 Les barres horizontales

On a depuis le début une barre en bas de notre page, représentée par `<hr>`. On va la modifier un peu. Essayez le code suivant :

```
<p>Barre horizontale simple,
<hr>
```

```

<p>avec une longueur relative limitée,
<hr width=50%>

<p>avec une longueur absolue,
<hr width=100>

<p>avec d'autres positions dans la page,
<hr width=50% align=right>
<hr width=50% align=left>

<p>avec une épaisseur modifiée,
<hr size=3>

<p>sans l'ombré,
<hr noshade>

<p>avec une longueur dépendant de la taille des caractères,
<hr style="width: 10em;">

```

N'oubliez pas de changer la taille de votre fenêtre pour voir comment évoluent les traits. Changez aussi la taille des caractères (dans le menu affichage, ou en appuyant sur CONTROL et + ou -).

5.6 Styles, couleurs, tailles, fontes

```

<ul>
  <li> <b> Le texte peut être en gras, </b> </li>
  <li> <i> Les italiques mettent le texte en valeur !</i> </li>
  <li> <tt> Et enfin, on peut écrire à la machine du même nom, </tt></li>

  <li> <u> un style souligné à éviter : ici, on ne peut pas cliquer,</u> </li>
  <li> <s> et on peut même rayer du texte </s> </li>
  <li> <big> On peut écrire de gros caractères, </big> <small> ou de petits </small> </li>

  <li> Et enfin, on peut utiliser des exposants comme dans 1<sup>er</sup> ou des indices,
  u<sub>n+1</sub> = u<sub>n</sub>+1 </li>
</ul>
<ul>
  <li> <font color=purple>Du texte violet, </font></li>
  <li> du texte, <font size=+3> du texte plus gros (en relatif),</font> </li>
  <li> <font size=2> du texte, taille absolue, </font> </li>

  <li> <font face="Zapf Chancery"> et des jolies lettres.</font></li>
</ul>

```

5.7 Énumérations

Dans le paragraphe précédent, on vient de créer une liste. Que se passe-t-il si on remplace `ul` par `ol` ? Créez une liste, dont le premier élément commencera par « Les peintres », suivi d'une liste numérotée avec dans l'ordre vos trois peintres favoris. Le second élément sera la même chose avec « Les artisans ». Vous ferez de plus suivre le nom de votre musicien préféré d'une liste numérotée de ses trois meilleures chansons. Vous pourrez ensuite essayer de remplacer un `` par `<ol type="a">` (au lieu de "a", on peut choisir parmi : a, A, i, I, 1). Essayez aussi : `<ol start="42">`. Dans vos listes imbriquées, remplacez tous les `ol` par des `ul`, et regardez les points utilisés.

5.8 Tableaux

```
<table>
  <tr> <td> X </td> <td> 1 </td> <td> 2 </td> <td> 3 </td> </tr>
  <tr> <td> 1 </td> <td> 1 </td> <td> 2 </td> <td> 3 </td> </tr>
  <tr> <td> 2 </td> <td> 2 </td> <td> 4 </td> <td> 6 </td> </tr>
  <tr> <td> 3 </td> <td> 3 </td> <td> 6 </td> <td> 9 </td> </tr>
</table>
```

On ajoute maintenant au début de l'élément table :

```
<caption align=bottom> Table de multiplication </caption>
```

La table n'est pas encore très jolie. Ajoutez à l'élément table un attribut `border`, de valeur 1. Pour changer encore plus radicalement, essayez :

```
<table border=20 cellpadding=10 cellspacing=5>
```

Voici quelques exemples plus compliqués. Utilisez aussi `` pour changer la couleur du texte de façon appropriée.

L'alignement dans les cellules, verticalement, horizontalement,

```
<br><br>
<table border width=50%>
  <tr> <th> </th> <th ><font size=6> Peintre </font> <th> <font size=6>Écrivain </font>
  <th><font size=6> Artisan </font>
  <tr valign=top align=center> <th> <font size=6> 17ème</font> <td> Rembrandt
  <td> Molière <td> Bach
  <tr valign=middle align=left> <th><font size=6> 19ème</font> <td> Monet <td>
  Goethe <td> Brahms
  <tr valign=bottom align=right> <th><font size=6> 20ème</font> <td> <td> Ellington
  Modigliani <td> Jarry <td> Ellington
</table>

<hr>
On peut faire l'alignement, case par case,
<br><br>
<table border width=50%>
  <tr> <th> </th> <th ><font size=6> Peintre </font> <th> <font size=6>Écrivain </font>
  <th><font size=6> Artisan </font>
  <tr > <th> <font size=6> 17ème</font> <td> valign=top align=center> Rembrandt
  <td> Molière <td> Bach
  <tr > <th><font size=6> 19ème</font> <td> Monet <td> valign=middle
  align=right > Goethe <td> Brahms
  <tr > <th><font size=6> 20ème</font> <td> <td> valign=bottom align=left>
  Modigliani <td> Jarry <td> Ellington
</table>

<hr>
Les cases peuvent être fusionnées,
```

`<table border>`

```
  <tr> <td>un <td colspan=2> exemple simple </td>
  <tr> <td> avec <td> trois <td> colonnes
</table>
```

5.9 Liens

Votre document commence (enfin ce qu'il y a après `<body>`) sans doute par un titre `<h1>Titre</h1>`. Remplaçons-le par : `<h1 name="debut" id="debut">Titre</h1>` (on est censé utiliser `id` et pas `name`,

mais Internet Explorer tarde un peu, alors on fait avec. Ajoutez maintenant à la fin de votre document (avant </body>) : <p>Lien vers en haut. Cliquez dessus dans le navigateur. On peut aussi ajouter d'autres liens. Faites une copie de votre page sous un nom différent, par exemple `nouveau.html`. Ajoutez alors le code suivant :

nouvelle page. Que se passe-t-il si vous remplacez `href="nouveau.html"` par `href="nouveau.html#debut"` ? Faites dans la nouvelle page un lien vers la fin de la vieille page (notez que l'attribut `name` peut-être utilisé sur la plupart des éléments). Ajoutons encore, pour compléter la collection :

Un lien déjà visité.

Un lien pas encore visité.

Pour m'écrire.
Pour ouvrir cette même page dans une autre fenêtre.

Vous cliquerez sur tous ces liens, mais avant, regardez ce qui se passe si vous rajoutez les attributs `link="yellow"` et `vlink="orange"` à <body>.

5.10 Images

Si vous avez une image dans vos fichiers, tant mieux. Sinon, téléchargez-en une sur internet, n'importe laquelle. Créez un sous-répertoire `photo` dans `public_html`, et mettez-y cette image. Vous pouvez maintenant ajouter le code : en adaptant au nom de l'image. . . indique de remonter d'un répertoire. Déplacez le répertoire `photo` dans votre répertoire principal, et adaptez le lien : `src="../../photo/monimage.jpg"`. Que se passe-t-il ? Insérez maintenant votre image au milieu d'un paragraphe, et ajoutez-lui un attribut `align`. Essayez les valeurs : `top`, `bottom`, `center`, `left`, `right`. Arrangez-vous pour que cliquer sur votre image renvoie sur votre autre page. Pensez à ajouter un attribut `alt="Texte alternatif"` qui sera utilisé par les navigateurs ne pouvant afficher d'image.

5.11 CSS dans le corps du code

Vous pouvez définir des styles CSS directement dans la définition d'une balise (X)HTML. Dans l'exemple ci-dessous, nous utilisons une balise <div> qui permet de définir une "boîte" à l'intérieur d'un contenu :

```
<div style="background-color:orange; border:1px solid black; color:yellow; font-size:150%; padding:1em;">  
    Cette balise div a du style !  
</div>
```

Insérer le contenu de cette exemple dans la page HTML courante, et constatez l'effet.

5.12 Les CSS dans l'en-tête de la page

Plutôt que par la méthode précédente, il est préférable de définir vos styles CSS une fois pour toute dans une section particulière de votre page Web (on utilise normalement la section <head>).

```
<head>  
    <style type="text/css">  
        div  
        {  
            background-color:#339;  
            color:#fff;  
            padding:15px;  
            border-bottom:5px solid red;  
            margin-bottom:15px;  
        }  
    </style>  
</head>  
<body>  
    <div>  
        Cette phrase est présentée en fonction du style défini dans l'en-tête  
    </div>  
    <div>  
        Cette phrase aussi, est pourtant le style n'a été défini qu'une fois !  
    </div>  
</body>
```

Insérer le contenu de cette exemple dans la page HTML courante, et constatez l'effet.

5.13 Les CSS dans une feuille de style totalement séparée

La façon idéale de définir les CSS consiste à les enregistrer dans un document indépendant de vos pages (X)HTML. Grâce à cette méthode, toutes les pages qui font référence à cette feuille de style externe hériteront de toutes ses définitions.

Une page (X)HTML peut faire référence à plusieurs feuilles de styles en même temps. Dans ce cas, les définitions contenues dans ces différentes feuilles seront combinées entre elles.

Voici un exemple de styles définis dans un document séparé : "Document 'mes-styles.css' "

```
body
{
    background-color:#ccf;
    letter-spacing:.1em;
}

p
{
    font-style:italic;
    font-family:times,serif;
}
```

Document 'ma-page.html'

```
<head>
    <link href="mes-styles.css" media="all" rel="stylesheet" type="text/css" />
</head>
<body>
    <p>Voici un exemple de paragraphe.</p>
    <p>Et voici un deuxième paragraphe.</p>
</body>
```

Insérer le contenu de cette exemple dans la page HTML courante, et constatez l'effet.

5.14 Comment utiliser des classes pour appliquer un style

Pour créer une classe, vous devez simplement faire figurer son nom précédé d'un point. Pour éviter toute ambiguïté, votre nom de classe ne doit pas comporter d'espace.

Cette définition de style est à placer dans une feuille de styles ou dans la section <head> de votre page :

```
.mon-style
{
  color:red;
}
```

Pour appliquer le style défini dans votre classe à un élément, ajouter la mention class="nom-du-style" dans la définition de la balise :

```
<p class="mon-style">Le style s'applique à ce paragraphe</p>
<p>Mais pas à celui-là</p>
<p class="mon-style">Le style peut s'appliquer à ce paragraphe</p>
<div class="mon-style">Et aussi à cette balise !</div>
```

Chaque élément (X)HTML peut avoir aucune, une ou plusieurs classes. Pour appliquer plusieurs classes au même élément, précisez simplement la liste de classes en séparant leurs noms par un espace :

```
.mon-style1
{
  color:yellow;
}

.mon-style2
{
  background-color:#A0A0A0;
  font-weight:bold;
}
```

Code (X)HTML associé au CSS

```
<p class="mon-style1 mon-style2">Les styles des deux classes s'appliquent à ce
paragraphe</p>
<p class="mon-style2">Alors que ce paragraphe n'a qu'une seule classe </p>
```

Pour créer un *id*, vous devez simplement faire précéder son nom d'un dièse #. Pour éviter toute ambiguïté, votre nom d'*id* ne doit pas comporter d'espace.

Le principe de l'*id* est très similaire à celui de la classe à une exception près :

- Plusieurs éléments peuvent avoir la même classe ;
- Il ne doit y avoir qu'un seul élément ayant un *id* donné.

Cette définition de style est à placer dans une feuille de styles ou dans la section <head> de votre page :

```
#mon-style  
{ color:red;  
}
```

Pour appliquer le style défini dans votre *id* à un élément, ajouter la mention id="nom-du style" dans la définition de la balise :

```
<p id="mon-style">Le style s'applique à ce paragraphe</p>  
<p>Mais pas à celui-là</p>
```

6 Fiche TP n°2 : Prise en main JavaScript

6.1 Un petit exercice pour la forme !

Maintenant que vous avez appris à vous servir des conditions, il serait intéressant de faire un petit exercice pour que vous puissiez vous entraîner.

Fournir un commentaire selon l'âge de la personne. Vous devez fournir un commentaire sur quatre tranches d'âge différentes qui sont les suivantes :

Tranche d'âge	Exemple de commentaire
1 à 17 ans	« Vous n'êtes pas encore majeur. »
18 à 49 ans	« Vous êtes majeur mais pas encore senior. »
50 à 59 ans	« Vous êtes senior mais pas encore retraité. »
60 à 120 ans	« Vous êtes retraité, profitez de votre temps libre ! »

Le déroulement du code sera le suivant :

- L'utilisateur charge la page Web ;
- Il est ensuite invité à taper son âge dans une fenêtre d'interaction ;
- Une fois l'âge fourni l'utilisateur obtient un petit commentaire.

L'intérêt de cet exercice n'est pas spécialement de sortir un commentaire pour chaque tranche d'âge, mais surtout que vous cherchiez à utiliser la structure conditionnelle la plus adaptée et que vous puissiez préparer votre code à toutes les éventualités.

6.2 Un deuxième petit exercice

Nous utilisons un script pour demander à l'utilisateur les prénoms de ses frères et sœurs. Les prénoms sont alors stockés dans une chaîne de caractères. Voici ce code :

```
var prenoms = "", prenom;  
while (true) {  
    prenom = prompt('Entrez un prénom :');  
  
    if (prenom) {  
        prenoms += prenom + ' '; // Ajoute le nouveau prénom ainsi qu'une espace jus  
  
    } else {  
        break; // On quitte la boucle  
    }  
}  
alert(prenoms); // Affiche les prénoms à la suite
```

Ce que nous vous demandons ici, c'est de stocker les prénoms dans un tableau. Pensez à la méthode push(). À la fin, il faudra afficher le contenu du tableau, avec alert(), seulement si le tableau contient des prénoms ; en effet, ça ne sert à rien de l'afficher s'il ne contient rien. Pour l'affichage, séparez chaque prénom par une espace. Si le tableau ne contient rien, faites-le savoir à l'utilisateur, toujours avec alert().

6.3 Travail pratique

Ce TP sera consacré à un exercice bien particulier : la conversion d'un nombre en toutes lettres. Ainsi, si l'utilisateur entre le nombre « 41 », le script devra retourner ce nombre en toutes lettres : « quarante-et-un ». Ne vous inquiétez pas : vous en êtes parfaitement capables, et nous allons même vous aider un peu avant de vous donner le corrigé !

Pour mener à bien votre exercice, voici quelles sont les étapes que votre script devra suivre (vous n'êtes pas obligés de faire comme ça, mais c'est conseillé) :

- L'utilisateur est invité à entrer un nombre entre 0 et 999.
- Ce nombre doit être envoyé à une fonction qui se charge de le convertir en toutes lettres.
- Cette même fonction doit contenir un système permettant de séparer les centaines, les dizaines et les unités. Ainsi, la
- fonction doit être capable de voir que dans le nombre 365 il y a trois centaines, six dizaines et cinq unités. Pour obtenir ce
- résultat, pensez bien à utiliser le modulo. Exemple : $365 \% 10 = 5$.
- Une fois le nombre découpé en trois chiffres, il ne reste plus qu'à convertir ces derniers en toutes lettres.
- Lorsque la fonction a fini de s'exécuter, elle renvoie le nombre en toutes lettres.
- Une fois le résultat de la fonction obtenu, il est affiché à l'utilisateur.
- Lorsque l'affichage du nombre en toutes lettres est terminé, on redemande un nouveau nombre à l'utilisateur.

D'ailleurs, puisque l'écriture des nombres en français est assez tordue, [nous vous conseillons d'aller faire un petit tour ici](#) afin de vous remémorer les bases.

Afin que vous puissiez avancer sans trop de problèmes dans la lecture de votre code, il va falloir étudier l'utilisation des fonctions parseInt() et isNaN().

La fonction parseInt() supporte plusieurs bases arithmétiques :

Code : JavaScript

```
alert(parseInt('100', 2)); // Affiche : « 4 »
```

Code : JavaScript

```
alert(parseInt('010', 10)); // Affiche « 10 »
```

La fonction isNaN()

Code : JavaScript

```
var test = parseInt('test'); // Contient au final la valeur « NaN »  
alert(typeof test); // Affiche « number »
```

Code : JavaScript

```
var test = parseInt('test'); // Contient au final la valeur « NaN »  
alert(isNaN(test)); // Affiche « true »
```

Vous voilà maintenant prêts à vous lancer dans l'écriture de votre code. Nous précisons de nouveau que les nombres à convertir vont de 0 à 999, mais rien ne vous empêche de faire plus si le cœur vous en dit. Évitez de faire moins, vous manqueriez une belle occasion de vous entraîner correctement.

6.4 Mini-TP : recréer une structure DOM

Afin de s'entraîner à jouer avec le DOM, voici quatre petits exercices. Pour chacun d'eux, une structure DOM sous forme de code HTML vous est donnée, et il vous est demandé de recréer cette structure en utilisant le DOM.

... à compléter ...

7 Fiche TP n°3 : Prise en main PHP