

Développement d'Applications Web

Cours Licence L2 - 2019-2020

Version du 15/03/2020 16:51

Sommaire

1	Introduction à la programmation Web	7
1.1	Définitions	7
1.2	Fonctionnement du Web	11
1.3	Historique	14
1.4	Programmation web client/serveur :	15
2	Programmation coté client.....	16
2.1	HTML et CSS.....	16
2.1.1	Basiques d'un document HTML	18
2.1.2	Tables HTML, forms, lists, frames	37
2.1.3	Basiques de CSS.....	37
2.1.4	HTML5.....	44
2.1.4.1	Balise Canvas	45
2.1.4.2	Geolocation	46
2.1.4.3	Local Storage	47
2.2	XML	48
2.2.1	Structure	48
2.2.2	DTD, XML schema, XSLT	48
2.3	JavaScript	49
2.3.1	Basiques de JavaScript	50
2.3.1.1	Indentation.....	51
2.3.1.2	Le Javascript « dans la page ».....	51
2.3.1.3	Javascript externe	52
2.3.1.4	Les variables.....	54
2.3.1.5	Les types de variables	54
2.3.1.6	Tester l'existence de variables avec typeof	55
2.3.1.7	Opérateurs	56
2.3.1.8	Interagir avec l'utilisateur	58
2.3.1.9	Convertir entre chaînes de caractères et nombres	59
2.3.1.10	Les alternatives.....	59
2.3.1.11	Petit intermède : la fonction confirm().....	60
2.3.1.12	La condition « switch »	60
2.3.2	Les boucles	61
2.3.2.1	Portée des variables de boucle	62
2.3.3	Fonctions, objets et tableaux	63
2.3.3.1	Variables globales et variables locales	63
2.3.3.2	Les arguments et les valeurs de retour	65
2.3.3.3	Les fonctions anonymes	66

2.3.4	Les objets et les tableaux.....	67
2.3.4.1	Les objets littéraux	68
2.3.5	Modeler vos pages Web (DHTML et DOM).....	70
2.3.5.1	Le Document Object Model.....	70
2.3.5.2	L'objet window.....	70
2.3.5.3	Le document.....	71
2.3.5.4	Naviguer dans le document : La structure DOM	72
2.3.5.5	Accéder aux éléments : getElementById()	73
2.3.5.6	getElementsByName()	75
2.3.5.7	Accéder aux éléments grâce aux technologies récentes	76
2.3.5.8	L'héritage des propriétés et des méthodes.....	77
2.3.5.9	Éditer les éléments HTML	78
2.3.5.10	La classe	79
2.3.5.11	Le contenu : innerHTML	80
2.3.5.12	Naviguer entre les nœuds : La propriété parentNode	81
2.3.5.13	nodeType et nodeName	82
2.3.5.14	Lister et parcourir des nœuds enfants	83
2.3.5.15	nodeValue et data.....	84
2.3.5.16	childNodes.....	84
2.3.5.17	Créer et insérer des éléments : Ajouter des éléments HTML..	85
2.3.5.18	Cloner, remplacer, supprimer.....	87
2.3.5.19	Remplacer un élément par un autre.....	88
2.3.5.20	Supprimer un élément	88
2.3.5.21	Insérer à la bonne place : insertBefore()	89
2.3.6	Les évènements	90
2.3.6.1	Les événements au travers du DOM0	91
2.3.6.2	Les événements au travers du DOM2	92
2.3.6.3	L'objet Event	94
2.3.7	Les formulaires.....	95
2.3.8	Les expressions régulières.....	96
2.3.9	La bibliothèque jquery	106
2.3.9.1	Gestion des évènements	109
2.3.9.2	Attendre que le document soit chargé.....	111
2.3.9.3	Les évènements blur et focus	111
2.3.9.4	Le mot clé this	111
2.3.9.5	clique et double-click	112
2.3.9.6	L'évènement clavier	113
2.3.9.7	Evènement souris	114
2.3.9.8	Evénement submit.....	116
2.3.9.9	Autres effets	117
2.3.9.10	Animations	120
2.3.9.11	Manipuler le DOM	122
2.3.9.12	Communication asynchrone.....	124
2.3.9.13	jQuery User Interface	125

2.3.9.14	jQuery Mobile User Interface	128
2.4	Framework Angular	129
2.4.1	Basiques de Angular et le langage typescript	129
2.4.2	Premier exemple.....	129
2.4.3	Composants maitre/détails	129
2.4.4	Services	129
2.4.5	Routing	129
2.4.6	HTTP	129
2.5	Exemples d'application.....	129
2.5.1	Application 1 : formulaire	130
2.5.2	Application 2	133
2.5.3	133
3	Programmation coté serveur.....	134
3.1	PHP	134
3.1.1	Variables.....	134
3.1.2	Opérateurs	135
3.1.3	Fonctions	139
3.1.4	Portée des variables.....	140
3.1.5	Précédence des opérateurs.....	141
3.1.6	Alternatives	143
3.1.7	Boucles	144
3.1.8	Passage par référence	146
3.1.9	Inclusion d'un fichier.....	146
3.1.10	Programmation objet.....	147
3.1.10.1	Création d'un objet	147
3.1.10.2	Accès à un objet	147
3.1.10.3	Cloner un objet.....	148
3.1.10.4	Constructeur	149
3.1.10.5	Destructeur	149
3.1.10.6	Méthodes	149
3.1.10.7	Déclarer des constantes	150
3.1.10.8	Méthodes statiques	150
3.1.10.9	Héritage	151
3.1.10.10	Accéder à la méthode du parent.....	152
3.1.10.11	Méthodes Final.....	152
3.1.11	Les tableaux.....	153
3.1.11.1	Ajouter.....	153
3.1.11.2	Tableau littéraux.....	154
3.1.11.3	La commande foreach.....	154
3.1.11.4	Tableau multidimensionnel	155
3.1.11.5	Fonctions sur les tableaux	155
3.1.12	Pratiques PHP.....	156
3.1.12.1	La fonction printf	156
3.1.12.2	Fonctions sur les dates	159

3.1.12.3	Gestion des fichiers	161
3.1.12.4	Appels système.....	164
3.2	Connexion BDD	165
3.2.1	MySQL.....	165
3.2.1.1	Connexion à MySQL	165
3.2.1.2	Création d'une base de données	165
3.2.1.3	Création des tables	166
3.2.1.4	Champs autoincrement.....	169
3.2.1.5	Ajout de données dans une table.....	169
3.2.1.6	Modifications dans une table	169
3.2.1.7	Les indexes.....	170
3.2.1.8	Clés primaires et étrangères	171
3.2.1.9	Interrogation d'une BD : SELECT	171
3.2.1.10	Jointures	173
3.2.1.11	Suppression dans une table	174
3.2.1.12	Mise-à-jour d'une table.....	174
3.2.2	Postgres	175
3.2.3	Accès BDD via PHP	176
3.3	Interaction clients/serveur et communication asynchrone	180
3.3.1	Dialoguer avec le serveur.....	181
3.3.1.1	Le HTML	182
3.3.1.2	Le XML.....	182
3.3.1.3	Le JSON	183
3.3.2	Principes synchrones et asynchrones	183
3.3.3	L'objet XMLHttpRequest.....	185
3.3.4	Envoi d'une requête http	187
3.3.4.1	Passer des variables.....	187
3.3.4.2	Protéger les caractères.....	188
3.3.5	Traitement côté serveur (PHP).....	188
3.3.5.1	Récupérer les données	190
3.3.6	Une fonction de callback.....	190
3.3.7	Ajax avec jQuery.....	192
3.3.8	Exemple d'une API Serveur/Client.....	193
3.4	Framework Laravel	198
3.5	Framework Django.....	198
3.6	Exemples d'application.....	198
3.6.1	Application 1	198
3.6.2	Application 2	198
3.6.3	198
4	Services Web	199
4.1	XML	201
4.1.1	Les règles de base.....	202
4.1.2	Le prologue XML.....	203
4.1.3	Les instructions de traitement	203

4.1.4	Les sections CDATA	203
4.1.5	Les commentaires.....	204
4.1.6	La galaxie XML	204
4.1.7	La notion de <i>namespace</i>	204
4.1.8	Le style avec CSS.....	206
4.1.9	Le style avec Xslt/XPath.....	206
4.1.10	XQuery : XML comme une base de données	207
4.1.11	DTDs et schémas XML.....	207
4.1.12	DOM.....	207
4.1.13	XML sur le web.....	208
4.1.13.1	XHtml.....	208
4.1.13.2	SVG (pour Scalable Vector Graphics).....	208
4.1.13.3	SMIL (pour Synchronized Multimedia Integration Language)	208
4.1.13.4	XForms.....	208
4.1.13.5	XLink	208
4.1.13.6	XPointer	208
4.1.13.7	XMPP	209
4.1.13.8	SOAP et WSDL	209
4.2	SOAP	209
4.2.1	La structure d'un message SOAP	212
4.2.1.1	Exemple 1 message SOAP	214
4.2.1.2	Exemple 2 message SOAP	215
4.2.2	Construire un serveur SOAP	216
4.2.3	Construire un client SOAP	217
4.2.4	Implémentation de SOAP dans les autres langages	219
4.2.5	Les fichiers WSDL.....	219
4.2.6	Construire un fichier WSDL	221
4.2.7	Utilisation de fichiers WSDL avec le client.....	223
4.3	L'annuaire des services UDDI.....	224
4.4	Application 1 : service web <i>restful</i>	225
4.4.1	Création de la base de données et de la classe user	225
4.4.2	Les services web d'inscription et de login	230
4.5	Application 2 : service web SOAP (WSDL généré).....	232
4.6	Application 3 : service web SOAP (WSDL disponible).....	232
4.6.1	Côté serveur.....	232
4.6.2	Côté client	233
4.6.3	Fichier WSDL	233
4.7	Platform .NET	236
4.8	Platform Java.....	236
5	Sécurité	236
5.1	Protéger les données entrant dans la BDD	237
5.1.1	Détournement de clause WHERE	237
5.1.2	Détournement de la clause DELETE	238
5.1.3	Les magic quotes	238

5.1.4	Sécuriser	239
5.1.5	addcslashes()	240
6	Fiches TP	243
6.1	Fiche TP n°1 : Prise en main HTML/CSS	243
6.1.1	Des couleurs	243
6.1.2	Paragraphes	244
6.1.3	Titres.....	244
6.1.4	Paragraphes spéciaux	244
6.1.5	Les barres horizontales.....	245
6.1.6	Styles, couleurs, tailles, fontes	245
6.1.7	Énumérations.....	246
6.1.8	Tableaux	246
6.1.9	Liens	248
6.1.10	Images	248
6.1.11	CSS dans le corps du code	249
6.1.12	Les CSS dans l'en-tête de la page.....	249
6.1.13	Les CSS dans une feuille de style totalement séparée	250
6.1.14	Comment utiliser des classes pour appliquer un style	251
6.2	Fiche TP n°2 : Prise en main JavaScript	253
6.2.1	Un petit exercice pour la forme !	253
6.2.2	Un deuxième petit exercice	253
6.2.3	Travail pratique	254
6.2.4	Mini-TP : recréer une structure DOM.....	256
6.2.5	TP : un formulaire interactif (<i>travail personnel</i>)	260
6.2.6	Manipulation : formulaire avec jquery.....	261
6.3	Fiche TP n°3 : Prise en main PHP	264
6.3.1	Question / variables superglobales.....	265
6.3.2	Question / typage des variables.....	265
6.3.3	Exercice / les fonctions	266
6.3.4	Exercice / passage de paramètres dans l'URL	267
6.3.5	Exercice / traitement de formulaire.....	268
6.4	Fiche TP n°4 : Serveur+Client	270
6.5	Fiche TP n°5 : services web	271
6.5.1	Service web <i>restful</i>	271
6.5.2	Service web SOAP (WSDL généré)	271
6.5.3	Service web SOAP (WSDL disponible)	271
7	Contrôles continus	272
7.1	Fiche Contrôle 1 : réaliser des formulaires	272
7.2	Contrôle 2	278
7.2.1	Côté client	278
7.2.2	Côté serveur.....	278

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.

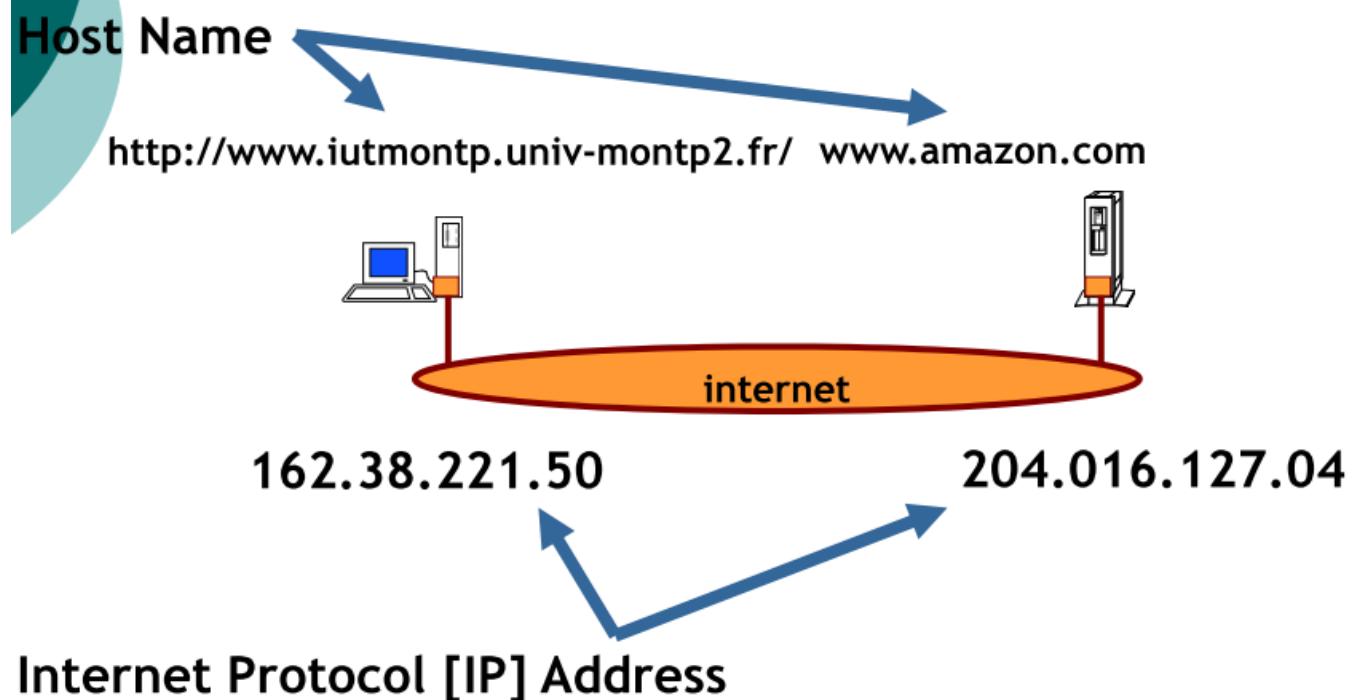
Utilisé pour la **communication entre les clients** Web et serveurs Web clients Web et serveurs Web.

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 référer 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

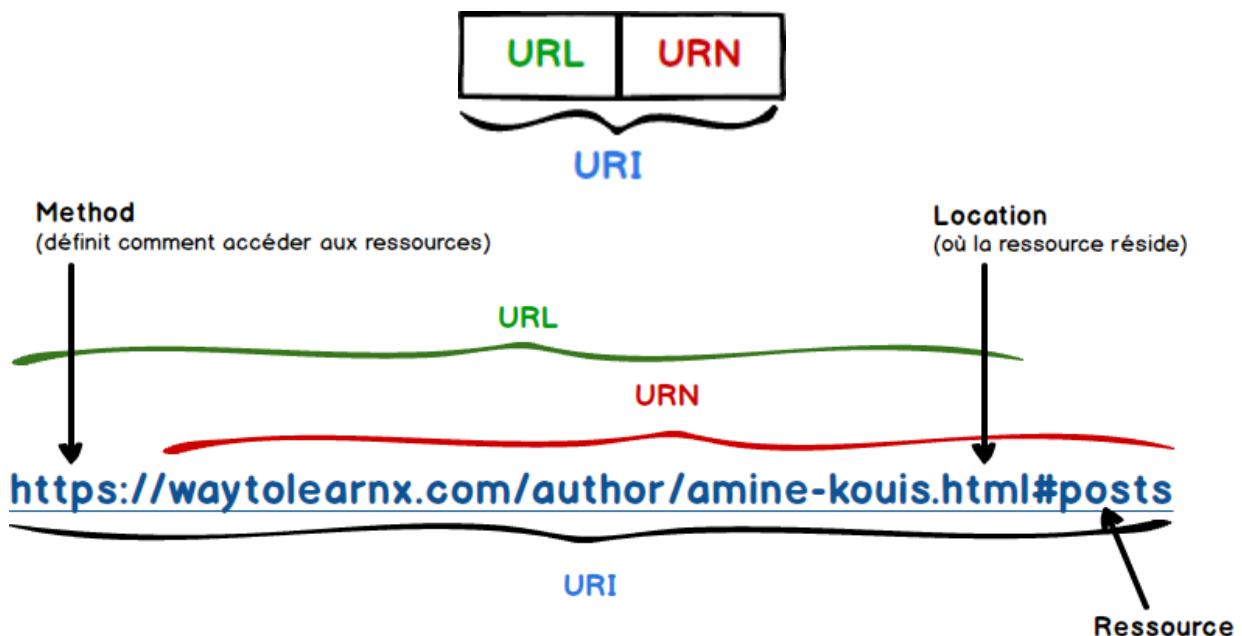
http://157.240.21.39/index.php

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



[<https://waytolearnx.com/2018/09/difference-entre-uri-et-url.html>]

1.2 Fonctionnement du Web

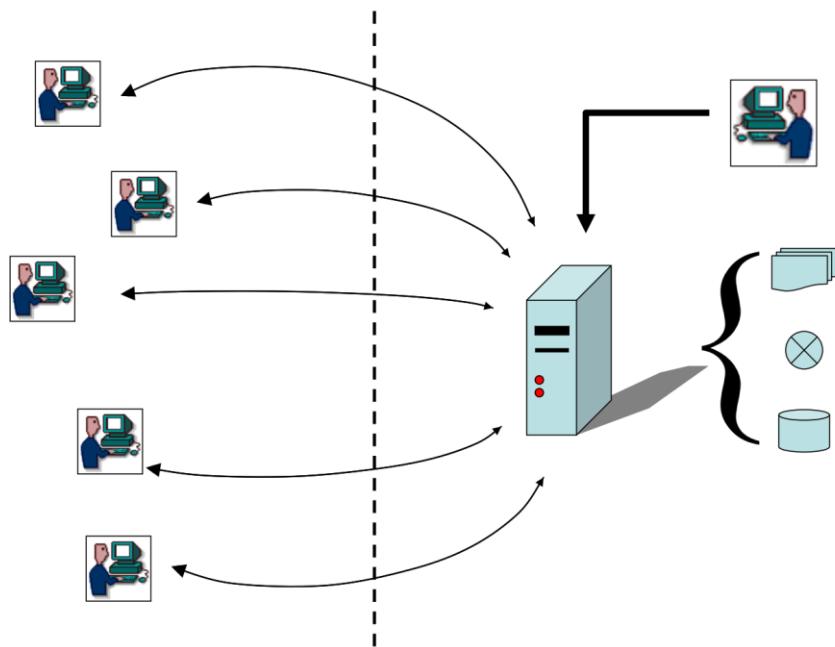


Figure - schéma de l'architecture client-serveur. Plusieurs clients partagent les fichiers, les applications et les bases de données placées en partage sur un serveur. Le trait pointillé entre le serveur et les client illustre le fait qu'un client et un serveur peuvent être très éloignés l'un de l'autre, 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.

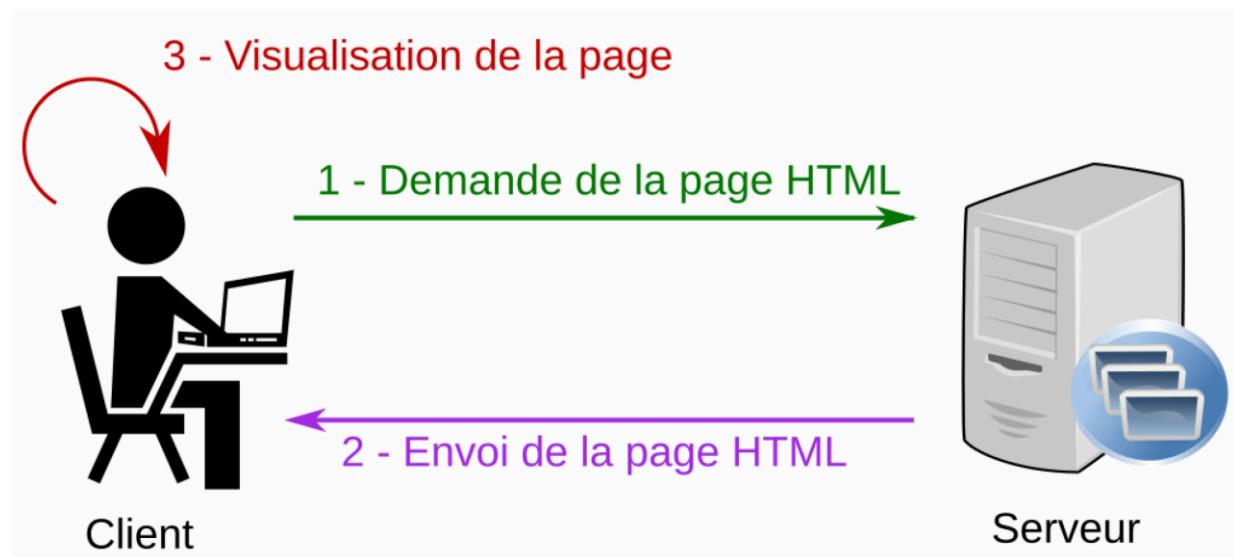
Lorsqu'on parle de « **client** » et de « **serveur** », il peut s'agir à la fois de l'**ordinateur** et du **programme** qui tourne sur cet ordinateur.

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

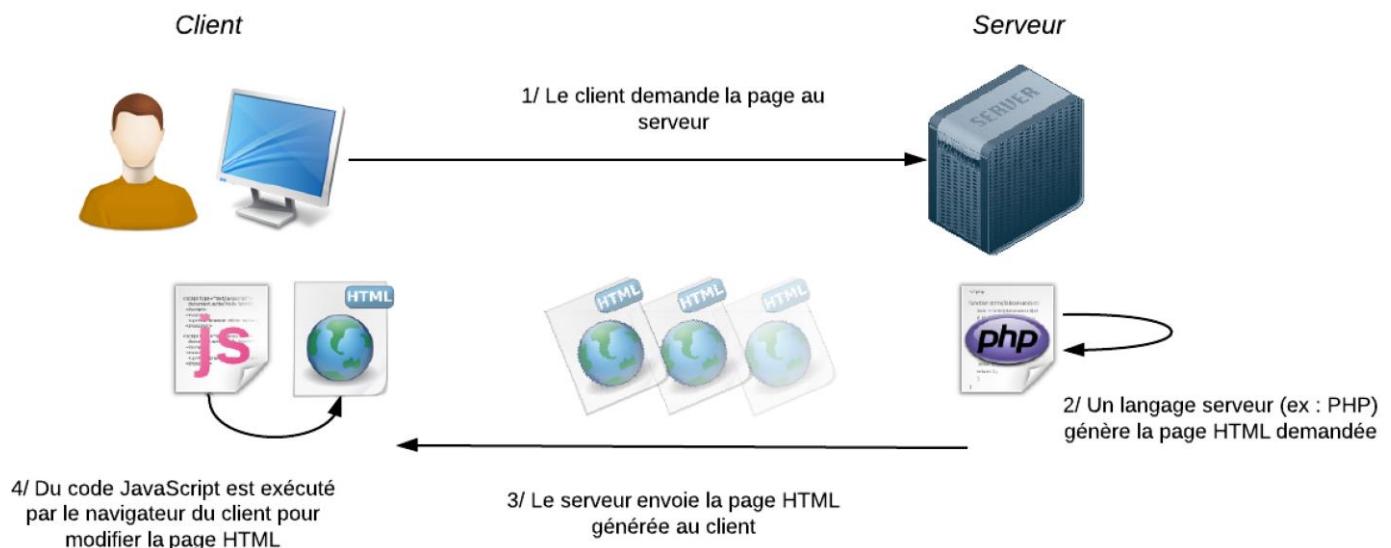
Le plus ancien HTML sur le web : 13 Novembre 1990

"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

Un texte en gras

Rendu par défaut

Un texte **en gras**

<a href="<http://www.ig2i.fr/>">Un lien

Un lien

```
<ul >
    <li>Point 1</li>
    <li>Point 2</li>
</ul>
```

- Point 1
- Point 2

On dit que <toto> est une balise ouvrante et </toto> une balise fermante.

On peut écrire <toto/> comme raccourci pour <toto></toto>.

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** !

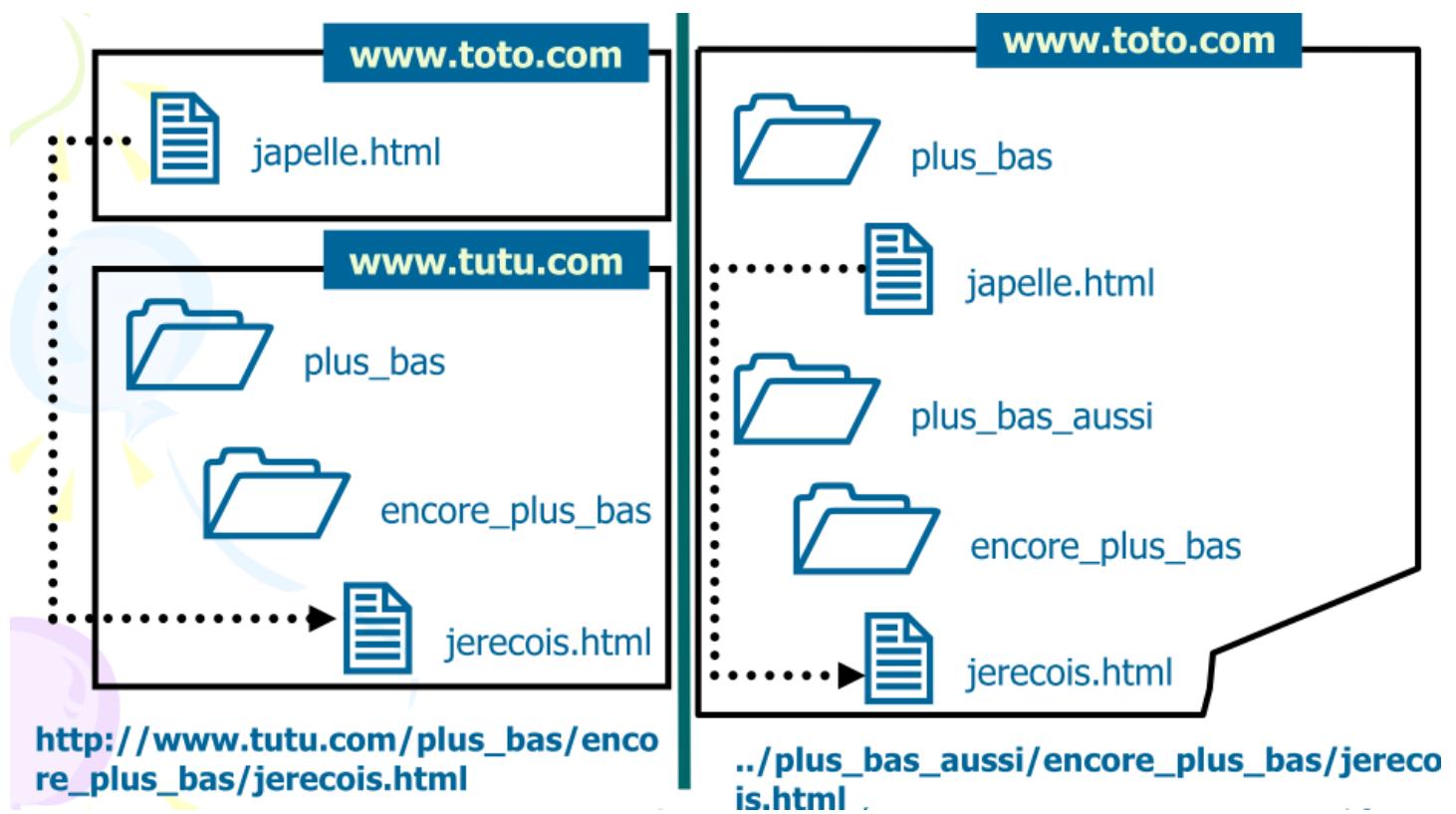
Un commentaire s'écrit entre <!-- et -->

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>
</head>
<body>
</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

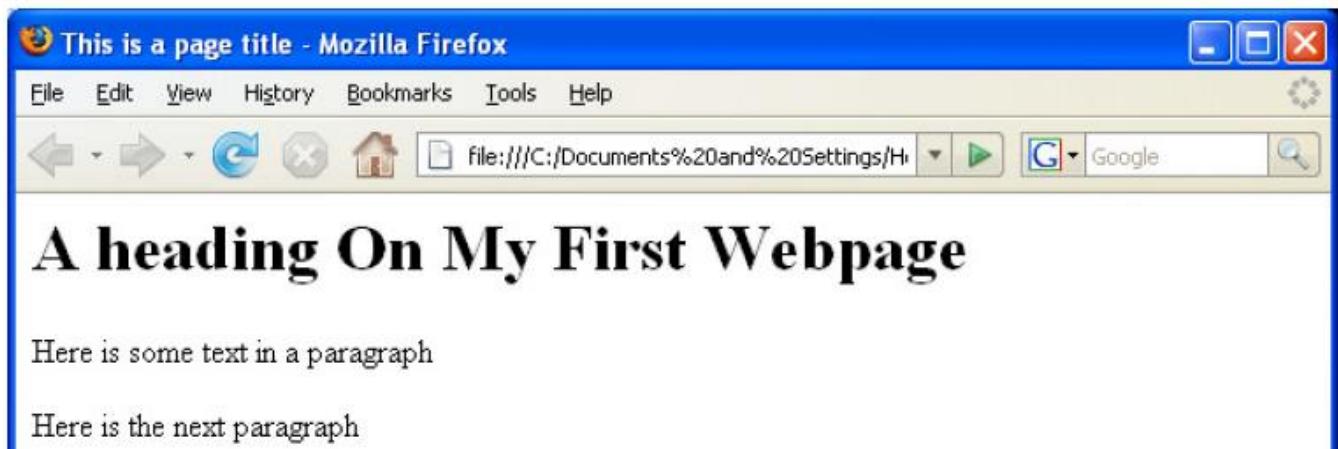
Titre niveau 4

Titre niveau 5

Titre niveau 6

- Un premier contenu : paragraphe

```
<html>
<head>
    <title>Example</title>
</head>
</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>
</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>
```



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>
    <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>
```

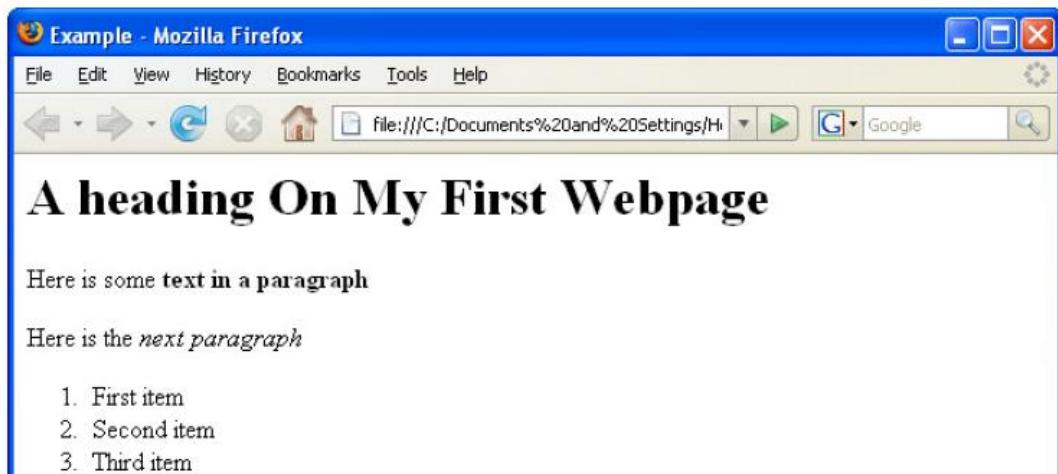


type="disc|circle|square" dans permet de choisir sa puce.

Exemple : <ul type="disc">.

Liste ordonnée :

```
<html>
<head>
    <title>Example</title>
</head>
<body>
    <h1>A heading On My First Webpage</h1>
    <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 a blue `<dl>` tag on the left. Inside it, there are three blue `<dt>niv1` tags, each followed by a blue `<dd>niv2` tag. The first `<dd>niv2` tag is followed by two more blue `<dd>niv2` tags, indicating a second level of nesting under the first `<dt>niv1`. To the right of the `<dd>niv2` tags, there are two blue `<dt>niv1` tags, which are then followed by two more blue `<dd>niv2` tags. Finally, the entire structure is closed by a blue `</dl>` tag on the far right.

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

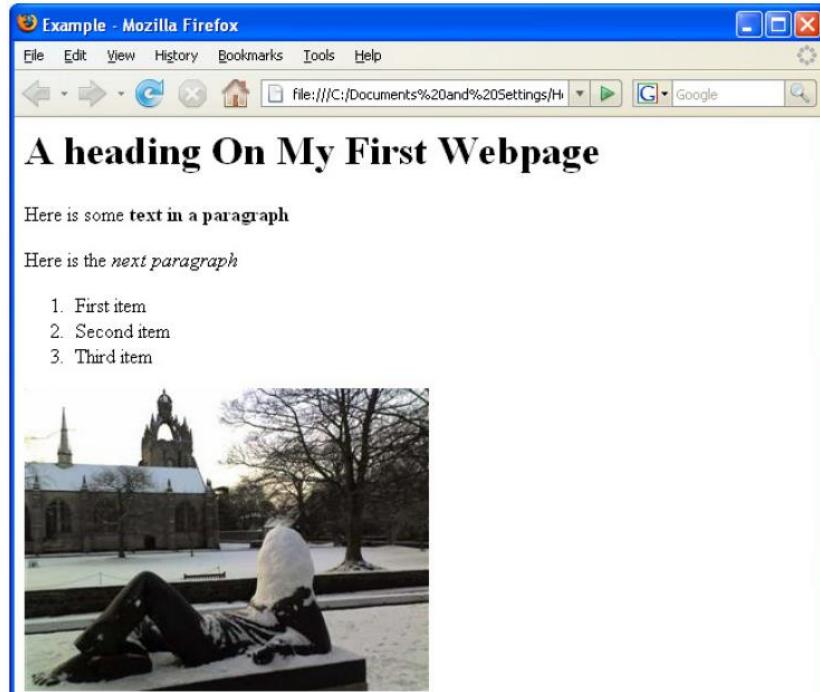
Création de lignes horizontales

<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.**

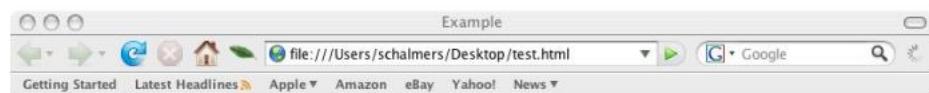
Images

```
<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 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>
    <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*

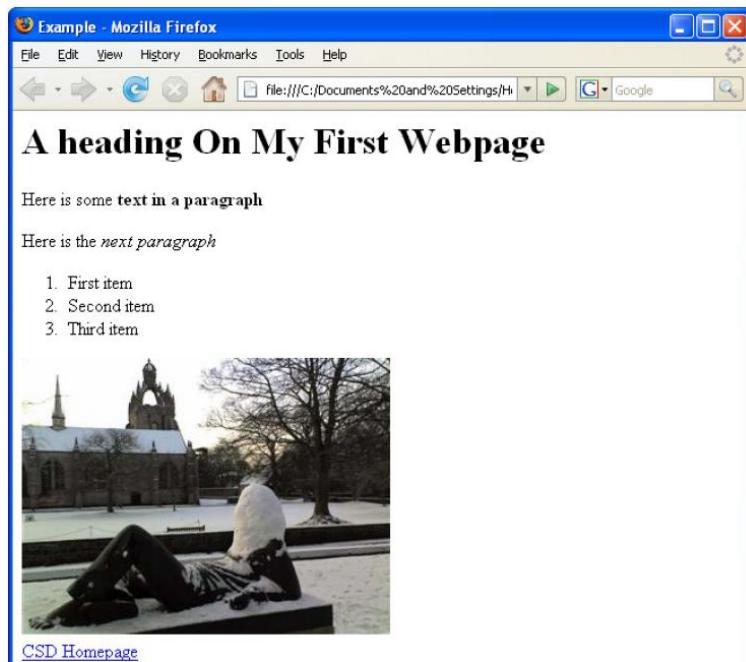
1. First item
2. Second item
3. Third item



[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 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 « mes-styles.css » séparé :

```
body {  
    background-color: #ccf;  
    letter-spacing: .1em;  
}  
  
p {  
    font-style: italic;  
    font-family: times, serif;  
}
```

```
<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>  
</html>
```

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 :

<link href="general.css" rel="stylesheet" type="text/css" media="all">	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.
<link href="ecran.css" rel="stylesheet" type="text/css" media="screen, projection">	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.
<link href="mobile.css" rel="stylesheet" type="text/css" media="handheld">	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.
<link href="impression.css" rel="stylesheet" type="text/css" media="print">	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.

```
.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 :

```
.mon-style1 {  
    color: yellow;  
}  
  
.mon-style2 {  
    background-color: #A0A0A0;  
    font-weight: bold;  
}
```

```
<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.

```
#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

[[Wikipedia HTML5](#)]

HTML5 (HyperText Markup Language 5) est la dernière révision majeure du HTML (format de données conçu pour représenter les pages web). Cette version a été **finalisée le 28 octobre 2014**. HTML5 spécifie deux syntaxes d'un modèle abstrait défini en termes de DOM : HTML5 et XHTML5. Le langage comprend également une couche application avec de nombreuses API, ainsi qu'un algorithme afin de pouvoir traiter les documents à la syntaxe non conforme. Le travail a été repris par le W3C en mars 2007 après avoir été lancé par le WHATWG. Les deux organisations travaillent en parallèle sur le même document afin de maintenir une version unique de la technologie. Le W3C clôt les ajouts de fonctionnalités le 22 mai 2011, annonçant une finalisation de la spécification en 20141, et encourage les développeurs Web à utiliser HTML 5 dès ce moment. Fin 2016, la version 5.1 est officiellement publiée et présente plusieurs nouveautés qui doivent faciliter le travail des développeurs d'applications Web2.

Dans le langage courant, HTML5 désigne souvent un ensemble de technologies Web (HTML5, CSS3 et JavaScript) permettant notamment le développement d'applications (cf. DHTML).

2.1.4.1 Balise Canvas

```
<!DOCTYPE html>
<html>

<head>
    <title>The HTML5 Canvas</title>
    <script>
        function O(i) { return typeof i == 'object' ? i :
document.getElementById(i) }
        function S(i) { return O(i).style }
        function C(i) { return document.getElementsByClassName(i) }
    </script>
</head>

<body>
    <canvas id='mycanvas' width='320' height='240'>
        This is a canvas element given the ID
        <i>mycanvas</i>
        This text is visible only in non-HTML5 browsers
    </canvas>
    <script>
        canvas = O('mycanvas')
        context = canvas.getContext('2d')
        context.fillStyle = 'red'
        S(canvas).border = '1px solid black'
        context.beginPath()
        context.moveTo(160, 120)
        context.arc(160, 120, 70, 0, Math.PI * 2, false)
        context.closePath()
        context.fill()
    </script>
</body>

</html>
```

2.1.4.2 Geolocation

```
<!DOCTYPE html>
<html>

<head>
    <title>Geolocation Example</title>
</head>

<body>
    <script>
        if (typeof navigator.geolocation == 'undefined')
            alert("Geolocation not supported.")
        else
            navigator.geolocation.getCurrentPosition(granted, denied)
        function granted(position) {
            var lat = position.coords.latitude
            var lon = position.coords.longitude

            alert("Permission Granted. You are at location:\n\n"
                + lat + ", " + lon +
                "\n\nClick 'OK' to load Google Maps with your location")
            window.location.replace("https://www.google.com/maps/@"
                + lat + "," + lon + ",14z")
        }
        function denied(error) {
            var message
            switch (error.code) {
                case 1: message = 'Permission Denied'; break;
                case 2: message = 'Position Unavailable'; break;
                case 3: message = 'Operation Timed Out'; break;
                case 4: message = 'Unknown Error'; break;
            }
            alert("Geolocation Error: " + message)
        }
    </script>
</body>

</html>
```

Audio et Video

```
<!DOCTYPE html>
<html>

<head>
  <title>HTML5 Video</title>
</head>

<body>
  <video width='560' height='320' controls>
    <source src='movie.mp4' type='video/mp4'>
    <source src='movie.webm' type='video/webm'>
    <source src='movie.ogv' type='video/ogg'>
  </video>
</body>

</html>
```

2.1.4.3 Local Storage

API permettant de stocker des données de type clé/valeur dans une BD locale au browser.

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.

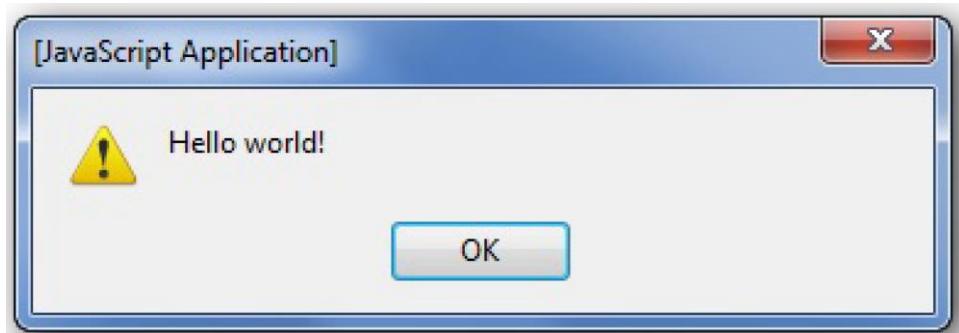
Le Javascript peut aussi être utilisé pour réaliser des applications. Mozilla Firefox est l'exemple le plus connu : **l'interface du navigateur est créée avec une sorte de HTML** appelé XUL et c'est le Javascript qui est utilisé pour animer l'interface.

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.

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

```
<!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 :

```
<!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 :

```
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 :

```
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 :

```
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

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

Table 13-6. JavaScript's escape characters

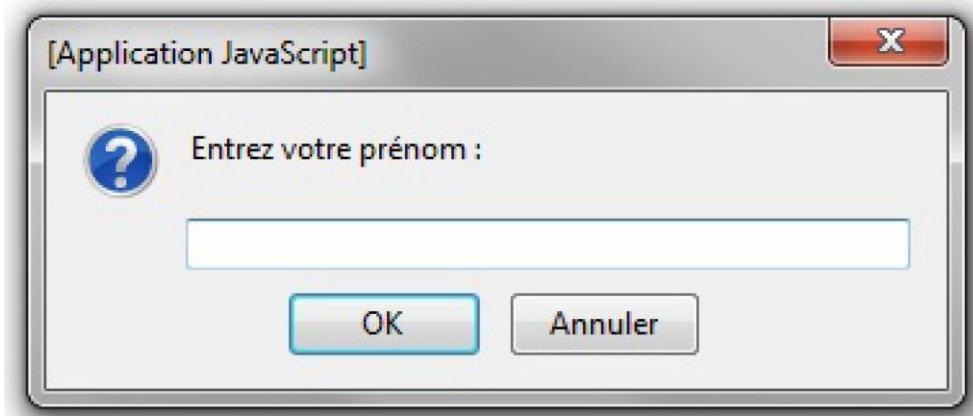
Character	Meaning
\b	Backspace
\f	Form feed
\n	Newline
\r	Carriage return
\t	Tab
\'	Single quote (or apostrophe)
\"	Double quote
\\\	Backslash
\xxx	An octal number between 000 and 377 that represents the Latin-1 character equivalent (such as \251 for the © symbol)
\xxx	A hexadecimal number between 00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

```
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 :

```
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

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

var first, second, result;
first = prompt('Entrez le premier chiffre :');
second = prompt('Entrez le second chiffre :');
result = parseInt(first) + parseInt(second);
alert(result);
```

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

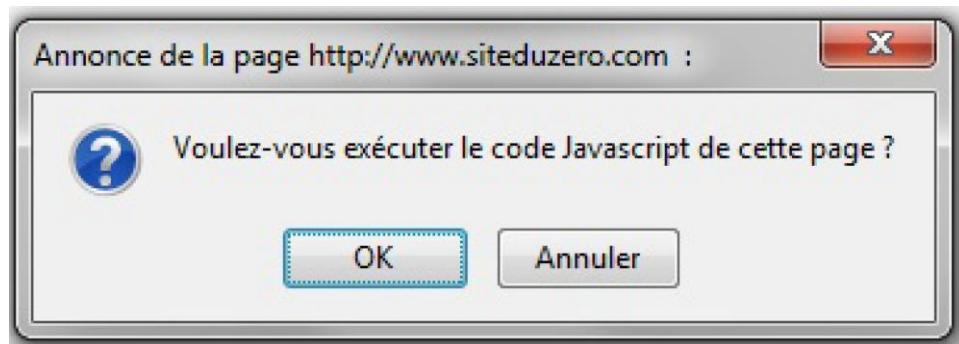
2.3.1.10 Les alternatives

```
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 :

```
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 »

```
var drawer = parseInt(prompt('Choisissez le tiroir à ouvrir (1 à 4): '));
if (drawer == 1) {
    alert('Contient divers outils pour dessiner : du papier, des crayons, etc.');
} else if (drawer == 2) {
    alert('Contient du matériel informatique : des câbles, des composants, etc.');
} else if (drawer == 3) {
    alert('Ah ? Ce tiroir est fermé à clé ! Dommage !');
} else if (drawer == 4) {
    alert('Contient des vêtements : des chemises, des pantalons, etc.');
} else {
    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

```
var number = 1;
while (number < 10) {
    number++;
}
alert(number); // Affiche : « 10 »
```

```
counter = 0
while (counter < 5) {
    document.write("Counter: " + counter + "<br>")
    ++counter
}
```

```
count = 1
do {
    document.write(count + " times 7 is " + count * 7 + "<br>")
} while (++count <= 7)
```

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

```
<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();
```

```
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 :

```
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

```
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 !');
```

```
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 :

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

```
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 :

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

```
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 :

```
var myArray = ['Ali', 'Nassima'];
myArray.push('Kada'); // Ajoute à la fin du tableau
myArray.push('Abdi', 'Ghoulam'); // Ajoute 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 :

```
var family = {  
    self:      'Saber',  
    sister:    'Latifa',  
    brother:   'Lotfi',  
    cousin_1:  'Boulifa',  
    cousin_2:  'Ghoulam'  
};
```

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 :

```
family['uncle'] = 'Dahmane'; // « ... » 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 :

```
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.

```
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!');
```

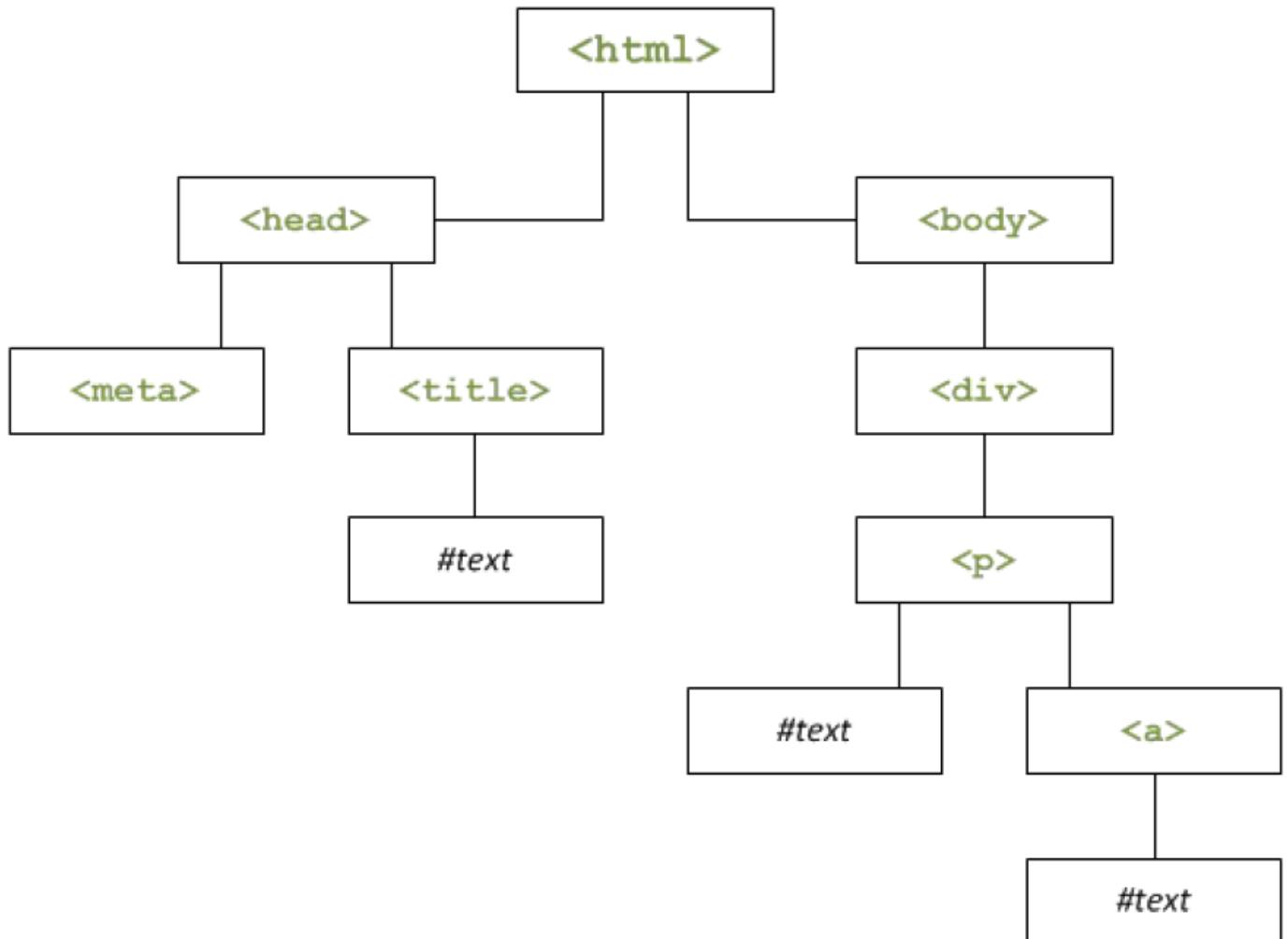
2.3.5.3 Le document

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 :



```
<!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 noeud (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 :

```
<!doctype html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Le titre de la page</title>
</head>
<body>
    <div id="myDiv">
        <p>Un peu de texte <a>et un lien</a></p>
    </div>
    <script>
        var div = document.getElementById('myDiv');
        alert(div);
    </script>
</body>
</html>
```



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

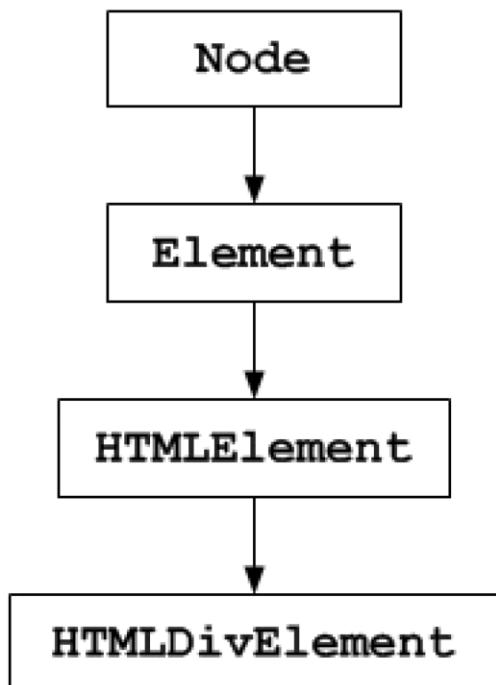
```
<div id="menu">
    <div class="item">
        <span>Élément 1</span>
        <span>Élément 2</span>
    </div>
    <div class="publicite">
        <span>Élément 3</span>
        <span>Élément 4</span>
    </div>
</div>
<div id="contenu">
    <span>Introduction au contenu de la page...</span>
</div>
```

```
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 <a> 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 :

```
<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 <a>, 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 :

```
<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.

```
<!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 :

```
<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 :

```
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 :

```
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 :

```
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.

```
<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` :

```
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.

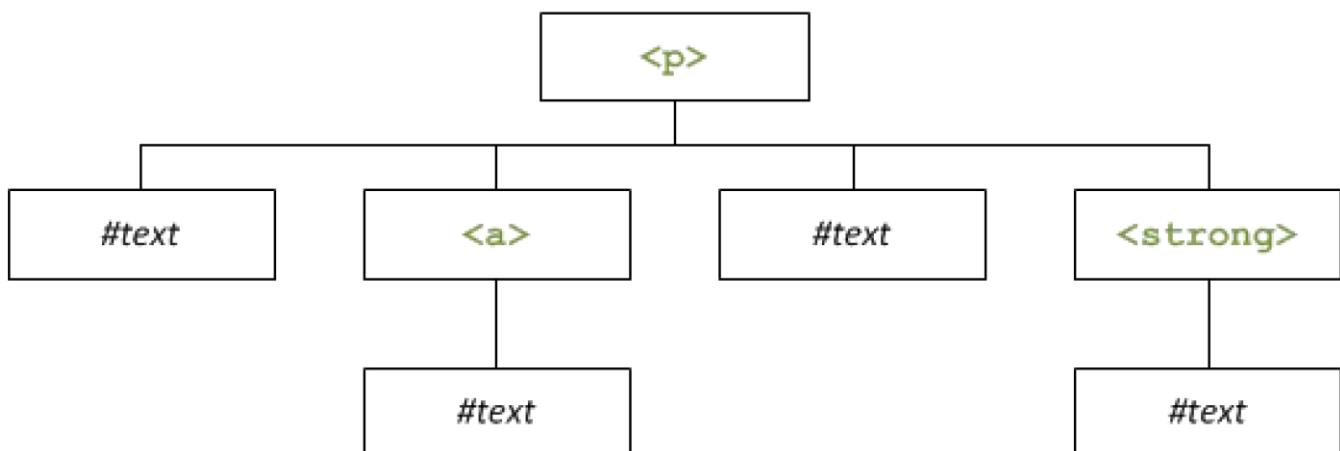
```
var paragraph = document.getElementById('myP');
alert(paragraph.nodeType + '\n\n' +
paragraph.nodeName.toLowerCase());
```

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.

```
<!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 :

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
        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 :

```
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.

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

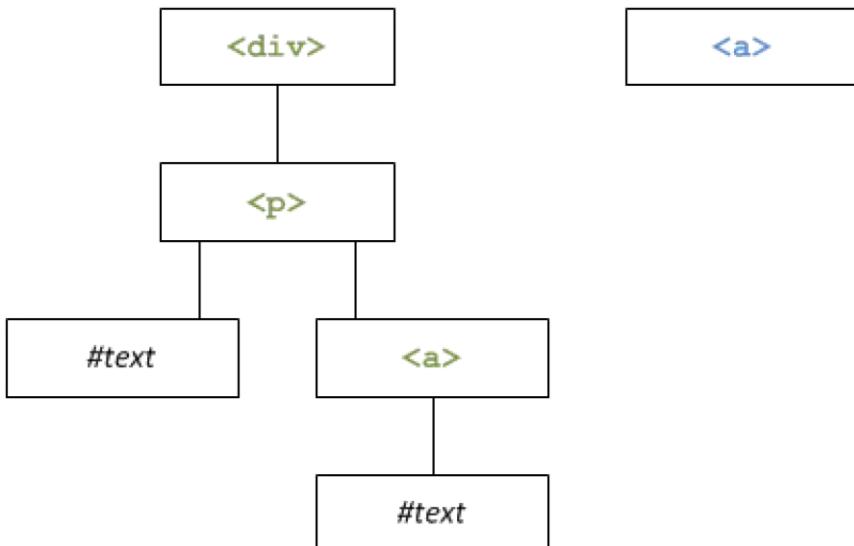
On utilise la méthode **appendChild()** pour insérer l'élément. Append child signifie « ajouter un enfant », ce qui signifie qu'il nous faut connaître l'élément auquel on va ajouter l'élément créé. Considérons donc le code suivant :

```
<!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>
```

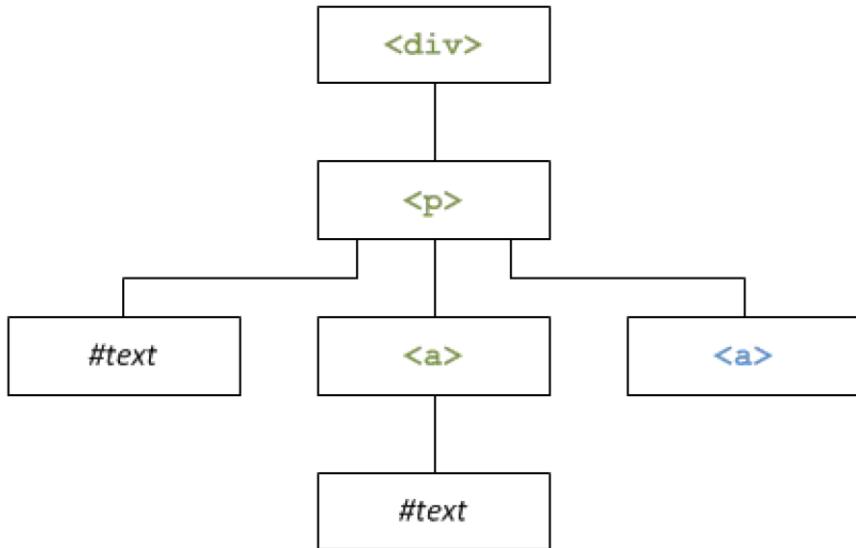
On va ajouter notre élément `<a>` dans l'élément `<p>` portant l'ID myP. Pour ce faire, il suffit de récupérer cet élément, et d'ajouter notre élément `<a>` via `appendChild()` :

```
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.

```
<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 :

```
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 :

```
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 :

```
<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 Les évènements

Nous étudierons l'utilisation des événements sans le DOM, avec le **DOM-0** (inventé par Netscape) puis avec le **DOM-2**. Nous verrons comment mettre en place ces événements, les utiliser, modifier leur comportement, etc.

Il existe de **nombreux événements**, tous plus ou moins utiles. Voici la liste des événements principaux, ainsi que les actions à effectuer pour qu'ils se déclenchent :

Nom de l'événement	Action pour le déclencher
click	Cliquer (appuyer puis relâcher) sur l'élément
dblclick	Double-cliquer sur l'élément
mouseover	Faire entrer le curseur sur l'élément
mouseout	Faire sortir le curseur de l'élément
mousedown	Appuyer (sans relâcher) sur le bouton gauche de la souris sur l'élément
mouseup	Relâcher le bouton gauche de la souris sur l'élément
mousemove	Faire déplacer le curseur sur l'élément
keydown	Appuyer (sans relâcher) sur une touche de clavier sur l'élément
keyup	Relâcher une touche de clavier sur l'élément
keypress	Frapper (appuyer puis relâcher) une touche de clavier sur l'élément
focus	« Cibler » l'élément
blur	Annuler le « ciblage » de l'élément
change	Changer la valeur d'un élément spécifique aux formulaires (<code>input</code> , <code>checkbox</code> , etc.)
select	Sélectionner le contenu d'un champ de texte (<code>input</code> , <code>textarea</code> , etc.)

Le focus définit ce qui peut être appelé le « ciblage » d'un élément. Lorsqu'un élément est ciblé, il va recevoir tous les événements de votre clavier. Un exemple simple est d'utiliser un `<input>` de type `text` : si vous cliquez dessus alors l'`input` possède le focus. Autrement dit : il est ciblé, et si vous tapez des caractères sur votre clavier vous allez les voir s'afficher dans l'`input` en question.

Commençons par l'événement click sur un simple `` :

```
<span onclick="alert('Hello !');">Cliquez-moi !</span>
```

Le mot-clé **this** n'est, normalement, pas censé vous servir dès maintenant, cependant il est toujours bon de le connaître pour les événements. Il s'agit d'une propriété pointant sur l'objet actuellement en cours d'utilisation. Donc, si vous faites appel à ce mot-clé lorsqu'un événement est déclenché, l'objet pointé sera l'élément qui a déclenché l'événement. Exemple :

```
<span onclick="alert('Voici le contenu de l\'élément que vous  
avez cliqué :\n\n' + this.innerHTML);">Cliquez-moi !</span>
```

2.3.6.1 Les événements au travers du DOM

Commençons par créer un simple code avec un événement click :

```
<span id="clickme">Cliquez-moi !</span>  
<script>  
    var element = document.getElementById('clickme');  
    element.onclick = function () {  
        alert("Vous m'avez cliqué !");  
    };  
</script>
```

Alors, voyons par étapes ce que nous avons fait dans ce code :

1. On récupère tout d'abord l'élément HTML dont l'ID est clickme ;
2. On accède ensuite à la propriété onclick à laquelle on assigne une fonction anonyme ;
3. Dans cette même fonction, on fait un appel à la fonction alert() avec un texte en paramètre.

2.3.6.2 Les événements au travers du DOM2

Le **DOM-2**, lui, permet la création multiple d'un même événement et gère aussi l'objet Event. Autrement dit, le DOM-2 c'est bien !

```
<span id="clickme">Cliquez-moi !</span>
<script>
    var element = document.getElementById('clickme');
    element.addEventListener('click', function () {
        alert("Vous m'avez cliqué !");
    }, false);
</script>
```

Concrètement, qu'est-ce qui change par rapport au DOM-0 ? Alors tout d'abord nous n'utilisons plus une propriété mais une méthode nommée `addEventListener()`.

Comme indiqué plus haut, le DOM-2 permet la **création multiple d'événements identiques** pour un même élément, ainsi, vous pouvez très bien faire ceci :

```
<span id="clickme">Cliquez-moi !</span>
<script>
    var element = document.getElementById('clickme');
    // Premier événement click
    element.addEventListener('click', function () {
        alert("Et de un !");
    }, false);

    // Deuxième événement click
    element.addEventListener('click', function () {
        alert("Et de deux !");
    }, false);
</script>
```

Venons-en maintenant à la suppression des événements ! Celle-ci s'opère avec la méthode **removeEventListener()** et se fait de manière très simple :

```
element.addEventListener('click', myFunction, false); // On  
crée l'événement  
element.removeEventListener('click', myFunction, false); //  
On supprime l'événement en lui repassant les mêmes paramètres
```

Le troisième argument *useCapture* : si *true* le gestionnaire est exécuté lors de la capture, sinon *false* après traitement des nœuds parents. Soit l'exemple suivant, où les trois premiers gestionnaires enclenchent d'abord le plus haut nœud (body), puis ses descendants, alors que les trois suivants commencent d'abord par le nœud le plus bas. (<https://riptutorial.com/fr/dom/example/1344/événement-bouillonnant-et-capturant>)

```
<!DOCTYPE html>  
<html>  
  
<head>  
    <meta charset="utf-8" />  
</head>  
  
<body id="body">  
    <p id="paragraph">  
        <span id="text">Hello World</span>  
    </p>  
  
<script>  
    document.body.addEventListener('click', function(event) {  
        console.log("Body clicked!");  
    }, false);  
    window.paragraph.addEventListener('click', function(event) {  
        console.log("Paragraph clicked!");  
    }, false);  
    window.text.addEventListener('click', function(event) {  
        console.log("Text clicked!");  
    }, false);  
  
    document.body.addEventListener('click', function(event) {  
        console.log("Body clicked!");  
    }, true);  
    window.paragraph.addEventListener('click', function(event) {  
        console.log("Paragraph clicked!");  
    }, true);
```

```
        window.text.addEventListener('click', function(event) {  
            console.log("Text clicked!");  
        }, true);  
  
        window.text.click();  
    </script>  
  
</body>  
  
</html>
```

2.3.6.3 L'objet Event

Tout d'abord, à quoi sert cet objet ? À vous fournir une multitude d'informations sur l'événement actuellement déclenché. Par exemple, vous pouvez récupérer quelles sont les touches actuellement enfoncées, les coordonnées du curseur, l'élément qui a déclenché l'événement... Les possibilités sont nombreuses !

```
        element.addEventListener('click', function (e) { //  
L'argument « e » va récupérer une référence vers l'objet « Event »  
            alert(e.type); // Ceci affiche le type de l'événement  
(click, mouseover, etc.)  
        }, false);
```

2.3.7 Les formulaires

Les formulaires sont simples à utiliser, cependant il faut d'abord mémoriser quelques propriétés de base. Comme vous le savez déjà, il est possible d'accéder à n'importe quelle propriété d'un élément HTML juste en tapant son nom, il en va donc de même pour des propriétés spécifiques aux éléments d'un formulaire comme value, disabled, checked, etc. Nous allons voir ici comment utiliser ces propriétés spécifiques aux formulaires.

Maintenant revenons sur deux événements : **submit** et **reset**, encore les mêmes noms ! Il n'y a sûrement pas besoin de vous expliquer quand l'un et l'autre se déclenchent, cela paraît évident. Cependant, il est important de préciser une chose : envoyer un formulaire avec la méthode submit() du Javascript ne déclenchera jamais l'événement submit ! Mais dans le doute, voici un exemple complet dans le cas où vous n'auriez pas tout compris :

```
<form id="myForm">
    <input type="text" value="Entrez un texte" />
    <br /><br />
    <input type="submit" value="Submit !" />
    <input type="reset" value="Reset !" />
</form>
<script>
    var myForm = document.getElementById('myForm');
    myForm.addEventListener('submit', function(e) {
        alert('Vous avez envoyé le formulaire !\n\nMais
celui-ci a été bloqué pour que vous ne changiez pas de page.');
        e.preventDefault();
    }, true);
    myForm.addEventListener('reset', function(e) {
        alert('Vous avez réinitialisé le formulaire !');
    }, true);
</script>
```

2.3.8 Les expressions régulières

Dans ce chapitre, nous allons aborder quelque chose d'assez complexe : les **expressions régulières**. C'est complexe, mais aussi très puissant ! Ce n'est pas un concept lié au Javascript, car les expressions régulières, souvent surnommées « regex », trouvent leur place dans bon nombre de langages, comme le Perl, le Python ou encore le PHP.

Les **regex** sont une sorte de langage « à part » qui sert à manipuler les chaînes de caractères. Voici quelques exemples de ce que les regex sont capables de faire :

- 1) **Vérifier** si une URL entrée par l'utilisateur ressemble effectivement à une URL. On peut faire pareil pour les adresses e-mail, les numéros de téléphone et toute autre syntaxe structurée ;
- 2) **Rechercher et extraire** des informations hors d'une chaîne de caractères (bien plus puissant que de jouer avec `indexOf()` et `substring()`) ;
- 3) **Supprimer** certains caractères, et au besoin les remplacer par d'autres ;
- 4) Pour les forums, **convertir** des langages en HTML lors des prévisualisations pendant la frappe ;
- 5) Et bien d'autres choses...

La syntaxe des regex en Javascript découle de la syntaxe des regex du langage Perl. C'est un langage assez utilisé pour l'analyse et le traitement des données textuelles (des chaînes de caractères, donc), en raison de la puissance de ses expressions régulières. Le Javascript hérite donc d'une grande partie de la puissance des expressions régulières de Perl.

Les regex ne s'utilisent pas seules, et il y a deux manières de s'en servir : soit par le biais de `RegExp` qui est l'objet qui gère les expressions régulières, soit par le biais de certaines méthodes de l'objet `String` :

- **match()** : retourne un tableau contenant toutes les occurrences recherchées ;
- **search()** : retourne la position d'une portion de texte (semblable à `indexOf()` mais avec une regex) ;
- **split()** : la fameuse méthode `split()`, mais avec une regex en paramètre ;
- **replace()** : effectue un rechercher/remplacer.

Nous n'allons pas commencer par ces quatre méthodes car nous allons d'abord nous entraîner à écrire et tester des regex. Pour ce faire, nous utiliserons la méthode `test()` fournie par l'objet `RegExp`. L'instanciation d'un objet `RegExp` se fait comme ceci :

```
var myRegex = /contenu_à_rechercher/modifiers;
```

Cela ressemble à une chaîne de caractères à l'exception près qu'elle est encadrée par deux slashes / au lieu des apostrophes ou guillemets traditionnels.

Si votre regex contient elle-même des slashes, n'oubliez pas de les échapper en utilisant un anti-slash comme suit :

```
var regex_1 = /contenu/_contenu /; // La syntaxe est fausse
car le slash n'est pas échappé
var regex_2 = /contenu\_/_contenu/; // La syntaxe est bonne
car le slash est échappé avec un anti-slash
```

L'utilisation de la méthode test() est très simple. En cas de réussite du test, elle renvoie true ; dans le cas contraire, elle renvoie false.

```
if (myRegex.test('Chaîne de caractères dans laquelle
effectuer la recherche')) {
    // Retourne true si le test est réussi
} else {
    // Retourne false dans le cas contraire
}
```

ou encore

```
if (/contenu_à_rechercher/.test('Chaîne de caractères bla bla bla'))
```

Commençons par une recherche simple :

```
if (/raclette/.test('Je mangerais bien une raclette
savoyarde !')) {
    alert('Ça semble parler de raclette');
} else {
    alert('Pas de raclette à l\'horizon');
}
```

nous disposons de l'opérateur OU, représenté par la barre verticale pipe |. Son utilisation est très simple puisqu'il suffit de la placer entre chaque mot recherché, comme ceci :

```
if (/Raclette/i.test('Je mangerais bien une raclette  
savoyarde !')) {  
    alert('Ça semble parler de raclette');  
} else {  
    alert('Pas de raclette à l\'horizon');  
}
```

Les symboles ^ et \$ permettent respectivement de représenter le début et la fin d'une chaîne de caractères. Si un de ces symboles est présent, il indique à la regex que ce qui est recherché commence ou termine la chaîne. Cela délimite la chaîne en quelque sorte :

Chaîne	Regex	Résultat
Raclette savoyarde	/^Raclette savoyarde\$/	true
Une raclette savoyarde	/^Raclette/	false
Une raclette savoyarde	/savoyarde\$/	true
Une raclette savoyarde !	/raclette savoyarde\$/	false

Une classe de caractères est écrite entre crochets et sa signification est simple : une des lettres qui se trouve entre les crochets peut convenir.

Chaîne	Regex	Résultat
Cette tartiflette est grosse	/Cette tartiflette est gr[ao]sse/	true
Cette tartiflette est grasse	/Cette tartiflette est gr[ao]sse/	true

Si nous voulons trouver les lettres allant de a à o, on écrira [a-o].

L'exclusion d'un intervalle est possible aussi : [^b-y] qui exclura les lettres allant de b à y.

Le point permet de trouver n'importe quel caractère, à l'exception des sauts de ligne (les retours à la ligne). Le point symbolise donc un caractère quelconque :

Code : JavaScript

```
/gr.s/
```

- ? : ce symbole indique que le caractère qui le précède peut apparaître 0 ou 1 fois ;
- + : ce symbole indique que le caractère qui le précède peut apparaître 1 ou plusieurs fois ;
- * : ce symbole indique que le caractère qui le précède peut apparaître 0, 1 ou plusieurs fois.

- {n} : le caractère est répété n fois ;
- {n,m} : le caractère est répété de n à m fois. Par exemple, si on a {0, 5}, le caractère peut être présent de 0 à 5 fois ;
- {n,} : le caractère est répété de n fois à l'infini.

Si la tartiflette possède un, deux ou trois t, la regex peut s'écrire :

Code : JavaScript

```
/raclet{1,3}e/
```

Chaîne	Regex	Résultat
Hellowwwwwwwww	/Hellow+/?	true
Goooooogle	/Go{2,}gle/	true
Le 1er septembre	/Le [1-9] [a-z]{2,3} septembre/	true
Le 1er septembre	/Le [1-9] [a-z]{2,3}[a-z]+/	false

Ainsi, si on veut rechercher un caractère de a à z ou les métacaractères ! et ?, il faudra écrire ceci :

Code : JavaScript

```
/[a-z!?]/
```

Type	Description
\d	Trouve un caractère décimal (un chiffre)
\D	Trouve un caractère qui n'est pas décimal (donc pas un chiffre)
\s	Trouve un caractère blanc
\S	Trouve un caractère qui n'est pas un caractère blanc
\w	Trouve un caractère « de mot » : une lettre, accentuée ou non, ainsi que l'underscore
\W	Trouve un caractère qui n'est pas un caractère « de mot »

Type	Description
\n	Trouve un retour à la ligne
\t	Trouve une tabulation

Regular Expression Modifiers

Modifiers can be used to perform case-insensitive more global searches:

Modifier	Description
i	Perform case-insensitive matching
g	Perform a global match (find all matches rather than stopping after the first match)
m	Perform multiline matching

Regular Expression Patterns

Brackets are used to find a range of characters:

Expression	Description
[abc]	Find any of the characters between the brackets
[0-9]	Find any of the digits between the brackets
(x y)	Find any of the alternatives separated with

Metacharacters are characters with a special meaning:

Metacharacter	Description
\d	Find a digit
\s	Find a whitespace character
\b	Find a match at the beginning of a word like this: \bWORD, or at the end of a word like this: WORD\b
\uxxxx	Find the Unicode character specified by the hexadecimal number xxxx

Quantifiers define quantities:

Quantifier	Description
n+	Matches any string that contains at least one n
n*	Matches any string that contains zero or more occurrences of n
n?	Matches any string that contains zero or one occurrences of n

Exemple : Une adresse e-mail contient trois parties distinctes

- La partie locale, avant l'arobase ;
- L'arobase @ ;
- Le domaine, lui-même et de l'extension « com ».

```
/^[\w\.-]+@[^\w\.-]+\.\w{2,6}$/
```

```
var email = prompt("Entrez votre adresse e-mail :",
"javascript@siteduzero.com");
if (/^[\w\.-]+@[^\w\.-]+\.\w{2,6}$.test(email)) {
    alert("Adresse e-mail valide !");
} else {
    alert("Adresse e-mail invalide !");
}
```

L'objet RegExp **contient neuf propriétés, appelées \$1, \$2, \$3... jusqu'à \$9**. Comme nous allons le voir dans la sous-partie suivante, il est possible d'utiliser une regex pour extraire des portions de texte, et ces portions sont accessibles via les propriétés \$1 à \$9.

```
var birth = 'Je suis né en mars';
/^Je suis né en (\S+)$/.exec(birth);
alert(RegExp.$1); // Affiche : « mars »
```

La regex est exécutée via exec(). Et ici une autre explication s'impose. Quand on exécute test() ou exec(), le contenu des parenthèses capturantes est enregistré temporairement au sein de l'objet RegExp. Le premier couple de parenthèses sera enregistré dans la propriété \$1, le deuxième dans \$2 et ainsi de suite, jusqu'au neuvième, dans \$9. Cela veut donc dire qu'il ne peut y avoir qu'un maximum de neuf couples de parenthèses. Les couples sont numérotés suivant le sens de lecture, de gauche à droite.

Et pour accéder aux propriétés, il suffit de faire RegExp.\$1, RegExp.\$2, etc.

```
var email = prompt("Entrez votre adresse e-mail :",
    "javascript@siteduzero.com");
if (/^([a-zA-Z0-9._-]+)@([a-zA-Z0-9._-]+)\.([a-zA-Z]{2,6})$/.test(email)) {
    alert('Partie locale : ' + RegExp.$1 + '\nDomaine : ' +
        RegExp.$2 + '\nExtension : ' + RegExp.$3);
} else {
    alert('Adresse e-mail invalide !');
}
```

Mais on n'a pas besoin que ce soit une parenthèse capturante et qu'elle soit accessible via RegExp.\$1. Pour la rendre non capturante, on va ajouter ?: au début de la parenthèse, comme ceci :

```
/(https|http|ftp|steam):\/\/\//
```

Un rechercher-replacer se fait au moyen de la méthode replace() de l'objet String. Elle reçoit deux arguments : la regex et une chaîne de caractères qui sera le texte de remplacement. Petit exemple :

```
var sentence = 'Je m\'appelle Ali';
var result = sentence.replace(/Ali/, 'Said');
alert(result); // Affiche : « Je m'appelle Said »
```

Nous avions vu l'option i qui permet aux regex d'être insensibles à la casse des caractères. Il existe une autre option, g, qui signifie « rechercher plusieurs fois ». Par défaut, la regex donnée précédemment ne sera exécutée qu'une fois : dès que « Sébastien » sera trouvé, il sera remplacé...

La méthode **search()**, toujours de l'objet String, ressemble à `indexOf()` mis à part le fait que le paramètre est une expression régulière. `search()` retourne la position de la première occurrence trouvée. Si aucune occurrence n'est trouvée, -1 est retourné. Exactement comme `indexOf()` :

```
var sentence = 'Si ton tonton';
var result = sentence.search(/\bton\b/);
if (result > -1) { // On vérifie que quelque chose a été
trouvé
    alert('La position est ' + result); // 3
}
```

```
var sentence = 'Si ton tonton tond ton tonton, ton tonton
tondu sera tondu';
var result = sentence.match(/\btonton\b/g);
alert('Il y a ' + result.length + ' "tonton" :\n\n' +
result);
```

Nous avions vu que la méthode `split()` recevait une chaîne de caractères en paramètre. Mais il est également possible de transmettre une regex. C'est très pratique pour découper une chaîne à l'aide, par exemple, de plusieurs caractères distincts :

```
var family = 'Ali,Mourad;Said:Imad;Nour;Kada';
var result = family.split(/[,:;]/);
alert(result);
```

L'`alert()` affiche donc un tableau contenant tous les prénoms, car il a été demandé à `split()` de couper la chaîne dès qu'une virgule, un deux-points ou un point-virgule est rencontré.

2.3.9 La bibliothèque jquery

- [[jQuery, Le framework JavaScript du Web 2.0 \(2ième édition\), Luc VAN LANCKER](#)]
- [[Learning PHP, MySQL, & JavaScript, WITH JQUERY, CSS & HTML5, Robin Nixon, 2018, 5ème Edition](#)]

L'approche de jQuery ne consiste pas seulement en un codage des scripts plus intuitif et concis, mais sa **philosophie première est concentrée sur l'ensemble des éléments pris en compte par le DOM**. Le JavaScript traditionnel, dans son évolution historique, a dû s'accommorder du Document Object Model John Resig, avec jQuery, a en quelque sorte reconstruit le JavaScript en le percevant comme un véritable **langage de programmation axé en priorité sur le DOM**. Ce qui modifie totalement la façon d'appréhender et le JavaScript.

Tous les éléments du DOM sont aisément accessibles avec jQuery. Les méthodes getElementById , getElementsByName et getElementsByTagName du JavaScript **montrent très rapidement leurs limites**, spécialement lorsque le concepteur souhaite accéder aux attributs et autres propriétés de style. Avec jQuery, **tous les éléments du document peuvent être accédés facilement** et surtout intuitivement.

L'approche de jQuery est complète. Les méthodes et fonctions de jQuery ne se limitent pas à quelques animations d'ordre esthétique. **Par quelques lignes de code, jQuery peut modifier du texte, insérer des images, trier des tableaux ou réorganiser l'entièreté de la structure du document Html.** Ce framework pose un regard nouveau sur le JavaScript et, après un bref apprentissage, simplifie grandement la conception et l'écriture des scripts. Nous ne manquerons pas d'attirer votre attention, dans la suite de cet ouvrage, sur la concision du code ainsi produit.

Le code jQuery est compatible avec les différents navigateurs.

Une **communauté dynamique** de développeurs soutient jQuery. Cette communauté, initiée selon les principes historiques de passion et de partage d'Internet, fournit une **multitude de plug-ins**, soit des extensions de jQuery, dédiées à des tâches très spécifiques. Ces plug-ins, souvent des merveilles de programmation, sont disponibles librement sur la toile et sont très prisés par les concepteurs de site. Un carrousel d'images ou un tableau triable implémenté en quelques minutes et en quelques lignes de code simplifie grandement leur travail.

Dans la pratique, jquery.js sera souvent inclus dans un sous répertoire, par exemple js . On y accède alors par

```
<script type="text/javascript" src="js/jquery.js"></script>.
```

Il est possible d'utiliser directement les versions de jQuery hébergées par le site officiel de jquery. Le lien vers le fichier jQuery devient alors :

```
<script src='http://code.jquery.com/jquery-3.2.1.min.js'></script>
```

Mais aussi sur le site de microsoft et google :

```
<script src='http://ajax.aspnetcdn.com/ajax/jQuery/jquery-3.2.1.min.js'></script>
<script src='http://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js'> </script>
```

Le framework mis en place et prêt à être utilisé, il faut au préalable parcourir quelques notions théoriques. Toutes les instructions ou pour être plus précis les scripts jQuery s'articulent autour d'un modèle unique. Soit :

```
jQuery(function(){
    //contenu exécuté lorsque le document sera chargé
});
```

Pour **économiser** des frappes de clavier, le **signe dollar (\$)** qui fonctionne comme alias de jQuery est presque toujours utilisé.

```
$(function(){
    //contenu exécuté lorsque le document sera chargé
});
```

Un exemple simple :

```
<!DOCTYPE html>
<html>
<head>
    <title>First jQuery Example</title>
    <script src='jquery-3.4.1.min.js'></script>
</head>

<body>
    The jQuery library uses either the <code>$()</code> or
    <code>jQuery()</code> function names.
    <script>
        $('code').css('border', '1px solid #aaa')
    </script>
</body>
</html>
```

Pour **changer le style d'une balise**, il suffit d'invoquer la fonction css sur la balise, comme par exemple :

```
$(‘p’).css(‘text-align’, ‘justify’)
```

De même, pour **récupérer un élément** d'un style, comme par exemple :

```
color = $('#elem').css('color')
```

On peut **changer les styles** sur un élément tag, comme par exemple :

```
$(‘blockquote’).css(‘background’, ‘lime’)
```

sur un tag ayant un identificateur :

```
$(‘#advert’).css(‘border’, ‘3px dashed red’)
```

sur **une classe** :

```
$(‘.new’).css(‘text-decoration’, ‘underline’)
```

On peut aussi combiner :

```
$(‘blockquote, #advert, .new’).css(‘font-weight’, ‘bold’)
```

Exemple :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Second jQuery Example</title>
    <script src='jquery-3.2.1.min.js'></script>
  </head>
  <body>
    <blockquote>Powerful and flexible as JavaScript is, with a
plethora of
      built-in functions, it is still necessary to use additional
code for
      simple things that cannot be achieved natively or with CSS,
such as
      animations, event handling, and asynchronous
communication.</blockquote>
```

```
<div id='advert'>This is an ad</div>
<p>This is my <span class='new'>new</span> website</p>
<script>
  $('blockquote').css('background', 'lime')
  $('#advert').css('border', '3px dashed red')
  $('.new').css('text-decoration', 'underline')
  $('blockquote, #advert, .new').css('font-weight', 'bold')
</script>
</body>
</html>
```

2.3.9.1 Gestion des événements

Dans jquery, il est simple d'attacher un code réagissant à un évènement :

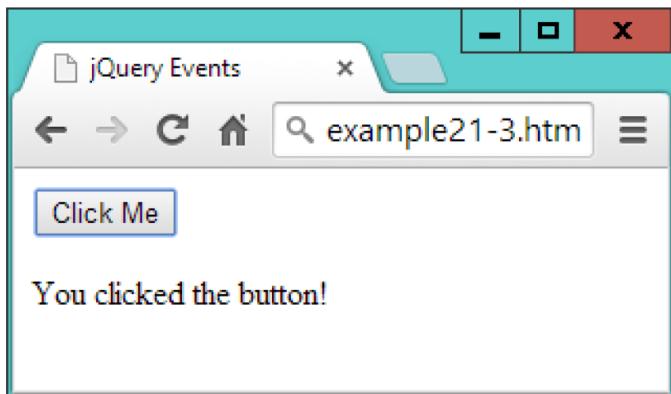
```
$('#clickme').click(function()
{
  $('#result').html('You clicked the button!')
})
```

```
<!DOCTYPE html>
<html>

<head>
  <title>jQuery Events</title>
  <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
  <button id='clickme'>Click Me</button>
  <p id='result'>I am a paragraph</p>
  <script>
    $('#clickme').click(function () {
      $('#result').html('You clicked the button!')
    })
  </script>
</body>

</html>
```



2.3.9.2 Attendre que le document soit chargé

Pour exécuter un code **suite au chargement complet d'un document**, il suffit de faire appel à la fonction suivante :

```
$(document).ready(function()
{
    // Your code goes here
})
```

2.3.9.3 Les évènements blur et focus

focus est un évènement déclenché quand un élément est pointé par l'utilisateur.

blur est un évènement déclenché quand le focus sorte d'un élément.

L'appel focus sans arguments force le focus sur l'élément en question.

```
<!DOCTYPE html>
<html>

<head>
    <title>Events: blur</title>
    <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
    <h2>Click in and out of these fields</h2>
    <input id='first'>
    <input>
    <input>
    <input>
    <script>
        $('#first').focus()
        $('input').focus(function () { $(this).css('background',
'#ff0') })
        $('input').blur(function () { $(this).css('background',
'#aaa') })
    </script>
</body>

</html>
```

2.3.9.4 Le mot clé this

Quand un évènement est enclenché, l'élément concerné prend le statut de this.

2.3.9.5 clique et double-click

```
<!DOCTYPE html>
<html>
  <head>
    <title>Events: click & dblclick</title>
    <script src='jquery-3.3.1.min.js'></script>
  </head>
  <body>
    <h2>Click and double click the buttons</h2>
    <button class='myclass'>Button 1</button>
    <button class='myclass'>Button 2</button>
    <button class='myclass'>Button 3</button>
    <button class='myclass'>Button 4</button>
    <button class='myclass'>Button 5</button>
    <script>
      $('.myclass').click( function() { $(this).slideUp() })
      $('.myclass').dblclick( function() { $(this).hide() })
    </script>
  </body>
</html>
```

2.3.9.6 L'évènement clavier

```
<!DOCTYPE html>
<html>
<head>
    <title>Events: keypress</title>
    <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
    <h2>Press some keys</h2>
    <div id='result'></div>
    <script>
        $(document).keypress(function (event) {
            key = String.fromCharCode(event.which)
            if (key >= 'a' && key <= 'z' ||
                key >= 'A' && key <= 'Z' ||
                key >= '0' && key <= '9') {
                $('#result').html('You pressed: ' + key)
                event.preventDefault()
            }
        })
    </script>
</body>

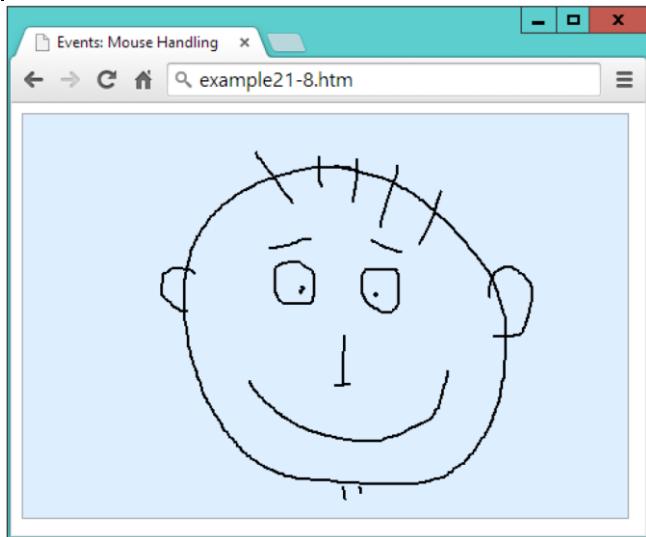
</html>
```

2.3.9.7 Evènement souris

```
<!DOCTYPE html>
<html>

<head>
    <title>Events: Mouse Handling</title>
    <script src='jquery-3.3.1.min.js'></script>
    <style>
        #pad {
            background: #def;
            border: 1px solid #aaa;
        }
    </style>
</head>

<body>
    <canvas id='pad' width='480' height='320'></canvas>
    <script>
        canvas = $('#pad')[0]
        context = canvas.getContext("2d")
        pendown = false
        $('#pad').mousemove(function (event) {
            var xpos = event.pageX - canvas.offsetLeft
            var ypos = event.pageY - canvas.offsetTop
            if (pendown) context.lineTo(xpos, ypos)
            else context.moveTo(xpos, ypos)
            context.stroke()
        })
        $('#pad').mousedown(function () { pendown = true })
        $('#pad').mouseup(function () { pendown = false })
    </script>
</body>
</html>
```



```
<!DOCTYPE html>
<html>

<head>
    <title>Events: Further Mouse Handling</title>
    <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
    <h2 id='test'>Pass the mouse over me</h2>
    <script>
        $('#test').mouseenter(function () { $(this).html('Hey, stop
tickling!') })
        $('#test').mouseleave(function () { $(this).html('Where did
you go?') })
    </script>
</body>

</html>
```

2.3.9.8 Evénement submit

Quand une forme est validée, on a envie de faire une vérification d'erreur :

```
<!DOCTYPE html>
<html>

<head>
    <title>Events: submit</title>
    <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
    <form id='form'>
        First name:
        <input id='fname' type='text' name='fname'>
        <br> Last name:
        <input id='lname' type='text' name='lname'>
        <br>
        <input type='submit'>
    </form>
    <script>
        $('#form').submit(function () {
            if ($('#fname').val() == '' ||
                $('#lname').val() == '') {
                alert('Please enter both names')
                return false
            }
        })
    </script>
</body>

</html>
```

2.3.9.9 Autres effets

Une panoplie d'effets peuvent être appliqués : hide, show, toggle, slideUp, slideDown, ...

Tous ces effets sont accompagné de 3 options possibles : durée, Easing (swing, linear), Callback (appelé à la fin de l'effet).

```
<!DOCTYPE html>
<html>

<head>
    <title>Effects: hide & show</title>
    <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
    <button id='hide'>Hide</button>
    <button id='show'>Show</button>
    <p id='text'>Click the Hide and Show buttons</p>
    <script>
        $('#hide').click(function () { $('#text').hide('slow',
'linear') })
        $('#show').click(function () { $('#text').show('slow',
'linear') })
    </script>
</body>

</html>
```

fade (décolorer ...)

```
<!DOCTYPE html>
<html>

<head>
    <title>Effects: Fading</title>
    <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
    <button id='fadeout'>fadeOut</button>
    <button id='fadein'>fadeIn</button>
    <button id='fadetoggle'>fadeToggle</button>
    <button id='fadeto'>fadeTo</button>
    <p id='text'>Click the buttons above</p>
    <script>
        $('#fadeout').click(function () { $('#text').fadeOut('slow')
    })
        $('#fadein').click(function () { $('#text').fadeIn('slow') })
        $('#fadetoggle').click(function () {
            $('#text').fadeToggle('slow') })
        $('#fadeto').click(function () { $('#text').fadeTo('slow',
0.5) })
    </script>
</body>

</html>
```

```
<!DOCTYPE html>
<html>

<head>
    <title>Effects: Sliding</title>
    <script src='jquery-3.3.1.min.js'></script>
</head>

<body>
    <button id='slideup'>slideUp</button>
    <button id='slidedown'>slideDown</button>
    <button id='slidetoggle'>slideToggle</button>
    <div id='para' style='background:#def'>
        <h2>From A Tale of Two Cities - By Charles Dickens</h2>
        <p>It was the best of times, it was the worst of times, it  
was the age of wisdom, it was the age of foolishness, it  
    was the epoch of belief, it was the epoch of incredulity,  
it was the season of Light, it was the season of Darkness,  
    it was the spring of hope, it was the winter of despair,  
we had everything before us, we had nothing before us,  
    we were all going direct to Heaven, we were all going  
direct the other way - in short, the period was so far  
        like the present period, that some of its noisiest  
authorities insisted on its being received, for good or for  
            evil, in the superlative degree of comparison only</p>
    </div>
    <script>
        $('#slideup').click(function () { $('#para').slideUp('slow')
})
        $('#slidedown').click(function () {
$('#para').slideDown('slow') })
        $('#slidetoggle').click(function () {
$('#para').slideToggle('slow') })
    </script>
</body>

</html>
```

2.3.9.10 Animations

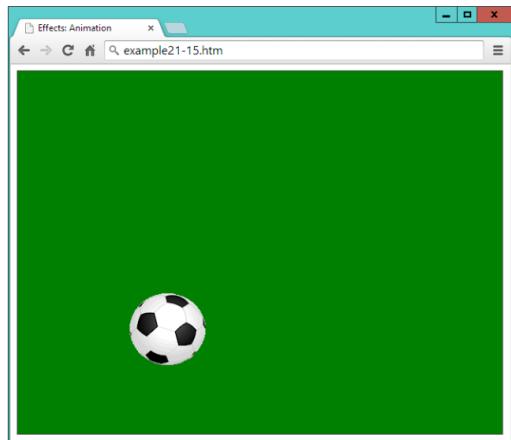
```
<!DOCTYPE html>
<html>

<head>
    <title>Effects: Animation</title>
    <script src='jquery-3.3.1.min.js'></script>
    <style>
        #ball {
            position: relative;
        }

        #box {
            width: 640px;
            height: 480px;
            background: green;
            border: 1px solid #444;
        }
    </style>
</head>

<body>
    <div id='box'>
        <img id='ball' src='ball.png'>
    </div>
    <script>
        bounce()
        function bounce() {
            $('#ball')
                .animate({ left: '270px', top: '380px' }, 'slow',
'slinear')
                .animate({ left: '540px', top: '190px' }, 'slow',
'slinear')
                .animate({ left: '270px', top: '0px' }, 'slow',
'slinear')
                .animate({ left: '0px', top: '190px' }, 'slow',
'slinear')
        }
    </script>
</body>

</html>
```



2.3.9.11 Manipuler le DOM

L'invocation se fait plus simplement qu'avec les appels standards du DOM :

```
<!DOCTYPE html>
<html>
  <head>
    <title>The DOM: html & text</title>
    <script src='jquery-3.3.1.min.js'></script>
  </head>
  <body>
    <h2>Example Document</h2>
    <p id='intro'>This is an example document</p>
    <script>
      alert($('#intro').html())
    </script>
  </body>
</html>
```

Toutes les méthodes vues sur le DOM existent d'une façon plus compacte. Exemples :

```
my_parent = $('#elem').parent()
$('li').parent('.memo').css('list-style-type', 'circle')
$('#elem').parents('div').css('background', 'yellow')
$('#elem').parentsUntil('div').css('background', 'yellow')
my_children = $('#elem').children()
$('#two').siblings('.new').css('font-weight', 'bold')
$('#new').next().css('font-weight', 'bold')
$('#new').nextAll().css('font-weight', 'bold') // all the siblings
$('#new').nextAll('.info').css('font-weight', 'bold')
$('#new').nextUntil('#old', '.info').css('font-weight', 'bold')
$('ul>li').first()           .css('text-decoration', 'underline')
($('ul>li').last()          .css('font-style', 'italic')
$('#ul>li').eq(1)            .css('font-weight', 'bold')
($('ul>li').filter(':even').css('background', 'cyan')
$('#ul>li').not('#new')     .css('color', 'blue')
($('ul>li').has('ol')       .css('text-decoration', 'line-through')
$.each(pets, function(name, type) {})
...
```

```
if ($(this).parent().is('div')) elem = 'div'
else                           elem = 'span'
```


2.3.9.12 Communication asynchrone

Soumission des données via post d'une façon asynchrone :

```
<!DOCTYPE html>
<html> <!-- jqueryasyncpost.htm -->
  <head>
    <title>jQuery Asynchronous Post</title>
    <script src='jquery-3.3.1.min.js'></script>
  </head>
  <body style='text-align:center'>
    <h1>Loading a web page into a DIV</h1>
    <div id='info'>This sentence will be replaced</div>
    <script>
      $.post('urlpost.php', { url : 'amazon.com/gp/aw' },
function(data)
{
  $('#info').html(data)
}
    </script>
  </body>
</html>
```

Récupération des données via get d'une façon asynchrone :

```
<!DOCTYPE html>
<html> <!-- jqueryasyncget.htm -->
  <head>
    <title>jQuery Asynchronous GET</title>
    <script src='jquery-3.3.1.min.js'></script>
  </head>
  <body style='text-align:center'>
    <h1>Loading a web page into a DIV</h1>
    <div id='info'>This sentence will be replaced</div>
    <script>
      $.get('urlget.php?url=amazon.com/gp/aw', function(data)
{
  $('#info').html(data)
}
    </script>
  </body>
</html>
```

2.3.9.13 *jQuery User Interface*

Au dessus de la librairie jquery, un ensemble impressionnant de composants ont été développés. Le site

<https://jqueryui.com>

contient une panoplie de composants riches.

Ces composants sont illustrés dans des démos :

<https://jqueryui.com/demos/>

Une panoplie de composants riches sont aussi disponible ici :

<https://jqueryui.pbworks.com>

Ci-dessous quelques exemples de composants visuels.

Voici une démo pour un champs de saisie avec autocomplete :

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>jQuery UI Autocomplete functionality</title>
  <link href="js/jquery-ui-1.12.1/jquery-ui.css" rel="stylesheet">
  <script src="js/jquery-3.3.1.js"></script>
  <script src="js/jquery-ui.js"></script>

  <!-- Javascript -->
  <script>
    $(function () {
      var availableTutorials = [
        "ActionScript",
        "Bootstrap",
        "C",
        "C++",
      ];
      $("#autocomplete-1").autocomplete({
        source: availableTutorials
      });
    });
  </script>
</head>

<body>
  <!-- HTML -->
  <div class="ui-widget">
    <p>Type "a" or "s"</p>
    <label for="autocomplete-1">Tags: </label>
    <input id="autocomplete-1">
  </div>
</body>

</html>
```

Pour déclarer un champs date :

```
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <title>jQuery UI Datepicker functionality</title>
    <link href="js/jquery-ui-1.12.1/jquery-ui.css" rel="stylesheet">
    <script src="js/jquery-3.3.1.js"></script>
    <script src="js/jquery-ui.js"></script>

    <!-- Javascript -->
    <script>
        $(function () {
            $("#datepicker-13").datepicker();
            $("#datepicker-13").datepicker("show");
        });
    </script>
</head>

<body>
    <!-- HTML -->
    <p>Enter Date:
        <input type="text" id="datepicker-13">
    </p>
</body>

</html>
```

2.3.9.14 *jQuery Mobile User Interface*

jQuery Mobile est une interface HTML5 pour concevoir des sites web responsifs accessibles à partir des smartphones, tablettes, terminaux desktop.

Il est téléchargeable d'ici :

<https://jquerymobile.com/>

Des démos sont disponibles ici :

<http://demos.jquerymobile.com/>

2.4 Framework Angular

2.4.1 Basiques de Angular et le langage typescript

2.4.2 Premier exemple

2.4.3 Composants maitre/détails

2.4.4 Services

2.4.5 Routing

2.4.6 HTTP

2.5 Exemples d'application

2.5.1 Application 1 : formulaire

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>jQuery</title>
  <link rel="stylesheet" href="js/jquery-ui-1.12.1/jquery-ui.css">
  <script src="js/jquery-3.3.1.js"></script>
  <script src="js/jquery-ui.js"></script>

  <style>
    form label {
      display: inline-block;
      width: 100px;
    }

    form div {
      margin-bottom: 10px;
    }

    .error {
      color: red;
      margin-left: 5px;
    }

    label.error {
      display: inline;
    }
  </style>

</head>

<body>

  <script>
    $(document).ready(function () {
      $('#first_form').submit(function (e) {
        e.preventDefault();
        var first_name = $('#first_name').val();
        var last_name = $('#last_name').val();
        var email = $('#email').val();
        var password = $('#password').val();
      })
    })
  </script>

```

```
        $(".error").remove();
        if (first_name.length < 1) {
            $('#first_name').after('<span class="error">This field is
required</span>');
        }
        if (last_name.length < 1) {
            $('#last_name').after('<span class="error">This field is
required</span>');
        }
        if (email.length < 1) {
            $('#email').after('<span class="error">This field is
required</span>');
        } else {
            var regEx = /^[A-Z0-9][A-Z0-9._%+-]{0,63}@(?:[A-Z0-9-
]{1,63}\.){1,125}[A-Z]{2,63}$/;
            var validEmail = regEx.test(email);
            if (!validEmail) {
                $('#email').after('<span class="error">Enter a valid
email</span>');
            }
        }
        if (password.length < 8) {
            $('#password').after('<span class="error">Password must be
at least 8 characters long</span>');
        }
    });
});
```

```
</script>
```

```
<script>
    $('form[id="second_form"]').validate({
        rules: {
            fname: 'required',
            lname: 'required',
            user_email: {
                required: true,
                email: true,
            },
            pswrd: {
                required: true,
                minlength: 8,
            }
        }
    });

```

```
},
messages: {
    fname: 'This field is required',
    lname: 'This field is required',
    user_email: 'Enter a valid email',
    psword: {
        minlength: 'Password must be at least 8 characters long'
    }
},
submitHandler: function (form) {
    form.submit();
}
});
</script>

<h2>Example 2:</h2>
<form id="second_form" method="post" action="">
    <div>
        <label for="fname">First Name:</label>
        <input type="text" id="fname" name="fname"></input>
    </div>
    <div>
        <label for="lname">Last Name:</label>
        <input type="text" id="lname" name="lname"></input>
    </div>
    <div>
        <label for="user_email">Email:</label>
        <input type="email" id="user_email" name="user_email"></input>
    </div>
    <div>
        <label for="psword">Password:</label>
        <input type="password" id="psword" name="psword"></input>
    </div>
    <div>
        <input type="submit" value="Submit" />
    </div>
</form>
</body>

</html>
```

2.5.2 Application 2

2.5.3 ...

3 Programmation côté serveur

3.1 PHP

- [[Learning PHP, MySQL, & JavaScript, WITH JQUERY, CSS & HTML5, Robin Nixon, 2018, 5ème Edition](#)]

Les documents PHP prennent comme extension ".php".

Pour exécuter une commande PHP il suffit de créer le tag ouvrant :

```
<?php
```

et de le ferme avec

```
?>
```

Soit un programme simple affichant un message :

```
<?php  
echo "Hello world";  
?>
```

Les commentaires :

```
// This is a comment
```

Commentaires multiples :

```
/* This is a section  
of multiline comments  
which will not be  
interpreted */
```

Les instructions terminent toujours un point-virgule :

```
$x += 10;
```

Les variables doivent toujours commencer par un \$.

```
$mycounter = 1;  
$mystring = "Hello";  
$myarray = array("One", "Two", "Three");
```

3.1.1 Variables

Variables de type chaîne "string" :

```
$username = "Mohammed";  
echo $username;
```

PHP envoie les données au navigateur et doit adopter le langage HTML pour formater la donnée :

```
<?php // test1.php
$username = "Mohammed";
echo $username;
echo "<br>";
$current_user = $username;
echo $current_user;
?>
```

Variables numériques

```
$count = 17;
$count = 17.5;
```

Tableaux

```
$team = array('Mohammed', 'Oram', 'Ali', 'Sofiane', 'Kada');
```

Tableau bi-dimensionnel :

```
$oxo = array(array('x', ' ', 'o'),
             array('o', 'o', 'y'),
             array('x', 'o', ' '));
echo $oxo[1][2]; // affiche y
```

3.1.2 Opérateurs

Table 3-1. Arithmetic operators

Operator	Description	Example
+	Addition	<code>\$j + 1</code>
-	Subtraction	<code>\$j - 6</code>
*	Multiplication	<code>\$j * 11</code>
/	Division	<code>\$j / 4</code>
%	Modulus (the remainder after a division is performed)	<code>\$j % 9</code>
++	Increment	<code>++\$j</code>
--	Decrement	<code>--\$j</code>
**	Exponentiation (or power)	<code>\$j**2</code>

Table 3-2. Assignment operators

Operator	Example	Equivalent to
=	\$j = 15	\$j = 15
+=	\$j += 5	\$j = \$j + 5
-=	\$j -= 3	\$j = \$j - 3
*=	\$j *= 8	\$j = \$j * 8
/=	\$j /= 16	\$j = \$j / 16
.=	\$j .= \$k	\$j = \$j . \$k
%=	\$j %= 4	\$j = \$j % 4

```
$count = $count + 1;
```

Table 3-3. Comparison operators

Operator	Description	Example
==	Is equal to	\$j == 4
!=	Is not equal to	\$j != 21
>	Is greater than	\$j > 3
<	Is less than	\$j < 100
>=	Is greater than or equal to	\$j >= 15
<=	Is less than or equal to	\$j <= 8
<>	Is not equal to to	\$j <> 23
===	Is identical to to	\$j === "987"
!==	Is not identical to to	\$j !== "1.2e3"

Table 4-6. PHP's cast types

Cast type	Description
(int) (integer)	Cast to an integer by dropping the decimal portion.
(bool) (boolean)	Cast to a Boolean.
(float) (double) (real)	Cast to a floating-point number.
(string)	Cast to a string.
(array)	Cast to an array.
(object)	Cast to an object.

Table 3-4. Logical operators

Operator	Description	Example
&&	<i>And</i>	<code>\$j == 3 && \$k == 2</code>
and	Low-precedence <i>and</i>	<code>\$j == 3 and \$k == 2</code>
	<i>Or</i>	<code>\$j < 5 \$j > 10</code>
or	Low-precedence <i>or</i>	<code>\$j < 5 or \$j > 10</code>
!	<i>Not</i>	<code>! (\$j == \$k)</code>
xor	<i>Exclusive or</i>	<code>\$j xor \$k</code>

```
if ($hour > 12 && $hour < 14) dolunch();
$html = file_get_contents($site) or die("Cannot download from
$site");
```

Incrémantation et décrémentation des variables

```
++$x;
--$y;
```

Ci-dessous, on décrémente la variable \$y puis la commande est exécutée :

```
if ($y-- == 0) echo $y;
```

Concaténation des chaines :

```
echo "You have " . $msgs . " messages.";
$bulletin .= $newsflash;
```

Pour déclarer une chaîne avec un contenu muet, il suffit de mettre le texte entre cotes :

```
$info = 'Preface variables with a $ like this: $variable';
```

Si on veut y mettre des variables qui seront remplacées par leurs contenus, on met tout entre "

```
echo "This week $count people have viewed your profile";
```

Faire attention :

```
$text = 'My spelling's atroshus'; // Erroneous syntax
```

A corriger en :

```
$text = 'My spelling\'s still atroshus';
```

```
$text = "She wrote upon it, \"Return to sender\".";
```

On peut intégrer la tabulation \t, le retour chariot \r, et la nouvelle ligne \n :

```
$heading = "Date\tName\tPayment";
```

On peut mettre une chaîne sur plusieurs lignes :

```
<?php
$author = "Ali Mohammed";
echo "Developers, developers, developers, developers, developers,
developers, developers, developers, developers!
- $author.";
?>
```

On peut aussi autrement :

```
<?php
$out = <<<_END
Normal people believe that if it ain't broke, don't fix it.
Engineers believe that if it ain't broke, it doesn't have enough
features yet.
- $author.
_END;
echo $out;
?>
```

PHP est un langage essentiellement non-typé. C'est-à-dire qu'on peut déclarer les variables sans spécifier leurs types, qui seront inférés suivant l'usage.

Les variables sont automatiquement converties suivant usage, comme par exemple :

```
<?php
$number = 12345 * 67890;
echo substr($number, 3, 1);
?>
```

ou encore

```
<?php
$pi      = "3.1415927";
$radius = 5;
echo $pi * ($radius * $radius);
?>
```

Les constantes sont déclarées comme suit :

```
define("ROOT_LOCATION", "/usr/local/www/");
```

Nous disposons d'un ensemble de constantes prédéfinies :

Table 3-5. PHP's magic constants

Magic constant	Description
<code>__LINE__</code>	The current line number of the file.
<code>__FILE__</code>	The full path and filename of the file. If used inside an <code>include</code> , the name of the included file is returned. Some operating systems allow aliases for directories, called <i>symbolic links</i> ; in <code>__FILE__</code> these are always changed to the actual directories.
<code>__DIR__</code>	The directory of the file. (Added in PHP 5.3.0.) If used inside an <code>include</code> , the directory of the included file is returned. This is equivalent to <code>dirname(__FILE__)</code> . This directory name does not have a trailing slash unless it is the root directory.
<code>__FUNCTION__</code>	The function name. (Added in PHP 4.3.0.) As of PHP 5, returns the function name as it was declared (case-sensitive). In PHP 4, its value is always lowercase.
<code>__CLASS__</code>	The class name. (Added in PHP 4.3.0.) As of PHP 5, returns the class name as it was declared (case-sensitive). In PHP 4, its value is always lowercased.
<code>__METHOD__</code>	The class method name. (Added in PHP 5.0.0.) The method name is returned as it was declared (case-sensitive).
<code>__NAMESPACE__</code>	The name of the current namespace. (Added in PHP 5.3.0.) This constant is defined at compile time (case-sensitive).

Exemple :

```
echo "This is line " . __LINE__ . " of file " . __FILE__;
```

On peut envoyer les messages au navigateur avec echo ou print. C'est pareil. L'avantage de print est que c'est une fonction qui retourne une chaîne, alors que echo n'est pas une fonction. Cette spécificité fait que dans certaines situations print offre plus de flexibilité :

```
$b ? print "TRUE" : print "FALSE";
```

3.1.3 Fonctions

La fonction suivante qui montre la formation de déclaration de fonction, permet de formater la date au format Jour Mois JourDuMois Année, comme par exemple Sunday May 2nd 2021 :

```
<?php
function longdate($timestamp)
{
    return date("l F jS Y", $timestamp);
}
?>
```

3.1.4 Portée des variables

Variables locales

```
<?php
    function longdate($timestamp)
    {
        $temp = date("l F jS Y", $timestamp);
        return "The date is $temp";
    }
?>
```

Pour rendre une variable globale :

```
global $is_logged_in;
```

Variables statiques :

```
<?php
    function test()
    {
        static $count = 0;
        echo $count;
        $count++;
    }
?>
```

Dans ce cas, la variable \$count sera incrémentée à chaque appel.

Variables super-globales :

Superglobal name	Contents
\$GLOBALS	All variables that are currently defined in the global scope of the script. The variable names are the keys of the array.
\$_SERVER	Information such as headers, paths, and locations of scripts. The entries in this array are created by the web server, and there is no guarantee that every web server will provide any or all of these.
<hr/>	
Superglobal name	Contents
\$_GET	Variables passed to the current script via the HTTP GET method.
\$_POST	Variables passed to the current script via the HTTP POST method.
\$_FILES	Items uploaded to the current script via the HTTP POST method.
\$_COOKIE	Variables passed to the current script via HTTP cookies.
\$_SESSION	Session variables available to the current script.
\$_REQUEST	Contents of information passed from the browser; by default, \$_GET, \$_POST, and \$_COOKIE.
\$_ENV	Variables passed to the current script via the environment method.

Ainsi, pour avoir l'URL de la page :

```
$came_from = $_SERVER['HTTP_REFERER'];
```

Cet accès direct peut provoquer des problèmes de sécurité. Il est plus prudent de protéger l'accès comme suit :

```
$came_from = htmlentities($_SERVER['HTTP_REFERER']);
```

3.1.5 Précédence des opérateurs

Table 4-2. The precedence of PHP operators (high to low)

Operator(s)	Type
()	Parentheses
++ --	Increment/decrement
!	Logical
* / %	Arithmetic
+ - .	Arithmetic and string
<< >>	Bitwise
< <= > >= <>	Comparison
== != === !==	Comparison
&	Bitwise (and references)
^	Bitwise

Operator(s)	Type
	Bitwise
&&	Logical
	Logical
? :	Ternary
= += -= *= /= .= %= &= != ^= <<= >>=	Assignment
and	Logical
xor	Logical
or	Logical

Associativité

Table 4-3. Operator associativity

Operator	Description	Associativity
< <= >= == != === !== <>	Comparison	None
!	Logical NOT	Right
~	Bitwise NOT	Right
++ --	Increment and decrement	Right
(int)	Cast to an integer	Right
(double) (float) (real)	Cast to a floating-point number	Right
(string)	Cast to a string	Right
(array)	Cast to an array	Right
(object)	Cast to an object	Right
@	Inhibit error reporting	Right
= += -= *= /=	Assignment	Right
.= %= &= = ^= <<= >>=	Assignment	Right
+	Addition and unary plus	Left
-	Subtraction and negation	Left

Operator	Description	Associativity
*	Multiplication	Left
/	Division	Left
%	Modulus	Left
.	String concatenation	Left
<< >> & ^	Bitwise	Left
? :	Ternary	Left
&& and or xor	Logical	Left
,	Separator	Left

3.1.6 Alternatives

```
<?php
if ($bank_balance < 100)
{
    $money      = 1000;
    $bank_balance += $money;
}
else
{
    $savings    += 50;
    $bank_balance -= 50;
}
?>
```

cascade d'alternatives :

```
<?php
if ($bank_balance < 100)
{
    $money      = 1000;
    $bank_balance += $money;
}
elseif ($bank_balance > 200)
{
    $savings    += 100;
    $bank_balance -= 100;
}
else
{
    $savings    += 50;
    $bank_balance -= 50;
}
?>
```

Alternative compacte :

```
<?php
echo $fuel <= 1 ? "Fill tank now" : "There's enough fuel";
?>
```

L'instruction switch :

```
<?php
switch ($page)
{
    case "Home":
        echo "You selected Home";
        break;
    case "About":
        echo "You selected About";
        break;
    case "News":
        echo "You selected News";
        break;
    case "Login":
        echo "You selected Login";
        break;
    case "Links":
        echo "You selected Links";
        break;
}
?>
```

3.1.7 Boucles

```
<?php
$fuel = 10;
while ($fuel > 1)
{
    // Keep driving...
    echo "There's enough fuel";
}
?>
```

```
<?php
$count = 1;
while ($count <= 12)
{
    echo "$count times 12 is " . $count * 12 . "<br>";
    ++$count;
}
?>
```

```
<?php
$count = 1;
do
    echo "$count times 12 is " . $count * 12 . "<br>";
while (++$count <= 12);
?>
```

```
<?php
for ($count = 1 ; $count <= 12 ; ++$count)
    echo "$count times 12 is " . $count * 12 . "<br>";
?>
```

```
<?php
$fp = fopen("text.txt", 'wb');
for ($j = 0 ; $j < 100 ; ++$j)
{
    $written = fwrite($fp, "data");
    if ($written == FALSE) break;
}
fclose($fp);
?>
```

On évitera la division par zéro avec la commande continue :

```
<?php
$j = 10;
while ($j > -10)
{
    $j--;
    if ($j == 0) continue;
    echo (10 / $j) . "<br>";
}
?>
```

3.1.8 Passage par référence

```
<?php
    $a1 = "WILLIAM";
    $a2 = "henry";
    $a3 = "gatES";
    echo $a1 . " " . $a2 . " " . $a3 . "<br>";
    fix_names($a1, $a2, $a3);
    echo $a1 . " " . $a2 . " " . $a3;
    function fix_names(&$n1, &$n2, &$n3)
    {
        $n1 = ucfirst(strtolower($n1));
        $n2 = ucfirst(strtolower($n2));
        $n3 = ucfirst(strtolower($n3));
    }
?>
```

3.1.9 Inclusion d'un fichier

```
<?php
    include "library.php";
    // Your code goes here
?>
```

Pour l'inclure une fois en cas il est aussi inclus dans les fichiers inclus :

```
<?php
    include_once "library.php";
    // Your code goes here
?>
```

Pour exiger l'inclusion :

```
<?php
    require_once "library.php";
    // Your code goes here
?>
```

3.1.10 Programmation objet

```
<?php
$object = new User;
print_r($object);
class User {
    public $name, $password;
    function save_user()
    {
        echo "Save User code goes here";
    }
}
?>
```

Il affiche :

User Object ([name] => [password] =>)

3.1.10.1 Crédit d'un objet

```
$object = new User;
$temp   = new User('name', 'password');
```

3.1.10.2 Accès à un objet

```
<?php
$object = new User;
print_r($object); echo "<br>";
$object->name      = "Joe";
$object->password  = "mypass";
print_r($object); echo "<br>";
$object->save_user();
class User {
    public $name, $password;
    function save_user()
    {
        echo "Save User code goes here";
    }
}
?>

User Object
(
    [name]      =>
    [password] =>
)
User Object
(
    [name]      => Joe
    [password] => mypass
)
Save User code goes here
```

3.1.10.3 Cloner un objet

Si on affecte un objet à un autre, l'objet reste le même :

```
<?php
$object1      = new User();
$object1->name = "Ali";
$object2      = $object1;
$object2->name = "Ami";
echo "object1 name = " . $object1->name . "<br>";
echo "object2 name = " . $object2->name;
class User
{
    public $name;
}
?>
```

On obtient :

```
object1 name = Ami
object2 name = Ami
```

Pour créer une nouvelle copie :

```
<?php
$object1      = new User();
$object1->name = "Ali";
$object2      = clone $object1;
$object2->name = "Ami";
echo "object1 name = " . $object1->name . "<br>";
echo "object2 name = " . $object2->name;
class User
{
    public $name;
}
?>
```

On obtient ainsi :

```
object1 name = Ali
object2 name = Ami
```

3.1.10.4 Constructeur

```
<?php
    class User
    {
        function __construct($param1, $param2)
        {
            // Constructor statements go here
            public $username = "Guest";
        }
    }
?>
```

3.1.10.5 Destructeur

```
<?php
    class User
    {
        function __destruct()
        {
            // Destructor code goes here
        }
    }
?>
```

3.1.10.6 Méthodes

```
<?php
    class User
    {
        public $name, $password;
        function get_password()
        {
            return $this->password;
        }
    }
?>
```

Ainsi

```
$object      = new User;
)object->password = "secret";
echo $object->get_password();
```

3.1.10.7 Déclarer des constantes

```
<?php
    Translate:::lookup();
    class Translate
    {
        const ENGLISH = 0;
        const SPANISH = 1;
        const FRENCH = 2;
        const GERMAN = 3;
        // ...
        static function lookup()
        {
            echo self::SPANISH;
        }
    }
?>
```

3.1.10.8 Méthodes statiques

```
<?php
    User:::pwd_string();
    class User
    {
        static function pwd_string()
        {
            echo "Please enter your password";
        }
    }
?>
```

Une méthode statique n'a pas accès aux propriétés de l'objet.

3.1.10.9 Héritage

```
<?php
    $object      = new Subscriber;
    $object->name      = "Fred";
    $object->password = "pword";
    $object->phone     = "012 345 6789";
    $object->email     = "fred@bloggs.com";
    $object->display();
    class User
    {
        public $name, $password;
        function save_user()
        {
            echo "Save User code goes here";
        }
    }
    class Subscriber extends User
    {
        public $phone, $email;
        function display()
        {
            echo "Name: " . $this->name . "<br>";
            echo "Pass: " . $this->password . "<br>";
            echo "Phone: " . $this->phone . "<br>";
            echo "Email: " . $this->email;
        }
    }
?>
```

3.1.10.10 Accéder à la méthode du parent

```
<?php
$object = new Son;
$object->test();
$object->test2();
class Dad
{
    function test()
    {
        echo "[Class Dad] I am your Father<br>";
    }
}
class Son extends Dad
{
    function test()
    {
        echo "[Class Son] I am Luke<br>";
    }
    function test2()
    {
        parent::test();
    }
}
?>
```

3.1.10.11 Méthodes Final

Si on veut empêcher une classe héritée de surcharger une méthode :

```
<?php
class User
{
    final function copyright()
    {
        echo "This class was written by Joe Smith";
    }
}
?>
```

3.1.11 Les tableaux

3.1.11.1 Ajouter

```
<?php  
$paper[] = "Copier";  
$paper[] = "Inkjet";  
$paper[] = "Laser";  
$paper[] = "Photo";  
print_r($paper);  
?>
```

Ca donne :

```
Array  
(  
    [0] => Copier  
    [1] => Inkjet  
    [2] => Laser  
    [3] => Photo  
)
```

ou encore :

```
<?php  
$paper[0] = "Copier";  
$paper[1] = "Inkjet";  
$paper[2] = "Laser";  
$paper[3] = "Photo";  
print_r($paper);  
?>
```

Pour les afficher un à un :

```
<?php  
$paper[] = "Copier";  
$paper[] = "Inkjet";  
$paper[] = "Laser";  
$paper[] = "Photo";  
for ($j = 0 ; $j < 4 ; ++$j)  
    echo "$j: $paper[$j]<br>";  
?>
```

3.1.11.2 Tableau littéraux

```
<?php
$paper['copier'] = "Copier & Multipurpose";
$paper['inkjet'] = "Inkjet Printer";
$paper['laser'] = "Laser Printer";
$paper['photo'] = "Photographic Paper";
echo $paper['laser'];
?>
```

ou encore

```
<?php
$p1 = array("Copier", "Inkjet", "Laser", "Photo");
echo "p1 element: " . $p1[2] . "<br>";
$p2 = array('copier' => "Copier & Multipurpose",
            'inkjet' => "Inkjet Printer",
            'laser' => "Laser Printer",
            'photo' => "Photographic Paper");
echo "p2 element: " . $p2['inkjet'] . "<br>";
?>
```

3.1.11.3 La commande foreach

```
<?php
$paper = array("Copier", "Inkjet", "Laser", "Photo");
$j = 0;
foreach($paper as $item)
{
    echo "$j: $item<br>";
    ++$j;
}
?>
```

ou encore

```
<?php
$paper = array('copier' => "Copier & Multipurpose",
            'inkjet' => "Inkjet Printer",
            'laser' => "Laser Printer",
            'photo' => "Photographic Paper");
foreach($paper as $item => $description)
    echo "$item: $description<br>";
?>
```

3.1.11.4 Tableau multidimensionnel

```
<?php
$products = array(
    'paper' => array(
        'copier' => "Copier & Multipurpose",
        'inkjet' => "Inkjet Printer",
        'laser' => "Laser Printer",
        'photo' => "Photographic Paper"),
    'pens' => array(
        'ball' => "Ball Point",
        'hilite' => "Highlighters",
        'marker' => "Markers"),
    'misc' => array(
        'tape' => "Sticky Tape",
        'glue' => "Adhesives",
        'clips' => "Paperclips"
    )
);
echo "<pre>";
foreach($products as $section => $items)
    foreach($items as $key => $value)
        echo "$section:\t$key\t($value)<br>";
echo "</pre>";
?>
```

3.1.11.5 Fonctions sur les tableaux

Quelques fonctions de base :

```
echo (is_array($fred)) ? "Is an array" : "Is not an array";
echo count($fred);
sort($fred);
shuffle($cards);
```

Pour éclater une chaîne contenant plusieurs mots : le premier argument est le séparateur, puis le texte dont on veut extraire les mots

```
<?php
$temp = explode(' ', "This is a sentence with seven words");
print_r($temp);
?>
```

3.1.12 Pratiques PHP

3.1.12.1 La fonction printf

```
printf("There are %d items in your basket", 3);
```

Table 7-1. The printf conversion specifiers

Specifier	Conversion action on argument arg	Example (for an arg of 123)
%	Display a % character (no arg required)	%
b	Display arg as a binary integer	1111011
c	Display ASCII character for arg	{
d	Display arg as a signed decimal integer	123
e	Display arg using scientific notation	1.23000e+2

Specifier	Conversion action on argument arg	Example (for an arg of 123)
f	Display arg as floating point	123.000000
o	Display arg as an octal integer	173
s	Display arg as a string	123
u	Display arg as an unsigned decimal	123
x	Display arg in lowercase hexadecimal	7b
X	Display arg in uppercase hexadecimal	7B

Pour afficher :

```
<span style='color:#417FF5'>Hello</span>
```

```
printf("<span style='color:#%X%X%X'>Hello</span>", 65, 127, 245);
```

Soit

```
printf("The result is: %.2f", 123.42 / 12);
```

Ca donne

The result is \$10.29

Soit par exemple

```
<?php
echo "<pre>"; // Enables viewing of the spaces
// Pad to 15 spaces
printf("The result is $%15f\n", 123.42 / 12);
// Pad to 15 spaces, fill with zeros
printf("The result is $%015f\n", 123.42 / 12);
// Pad to 15 spaces, 2 decimal places precision
printf("The result is $%15.2f\n", 123.42 / 12);
// Pad to 15 spaces, 2 decimal places precision, fill with zeros
printf("The result is $%015.2f\n", 123.42 / 12);
// Pad to 15 spaces, 2 decimal places precision, fill with # symbol
printf("The result is $%'#15.2f\n", 123.42 / 12);
?>
```

Ca donne :

The result is \$ 10.285000
The result is \$00000010.285000
The result is \$ 10.29
The result is \$000000000010.29
The result is \$#####10.29

Ou encore sur les chaînes :

```
<?php
    echo "<pre>"; // Enables viewing of the spaces
    $h = 'Free';
    printf("[%s]\n",           $h); // Standard string output
    printf("[%12s]\n",          $h); // Right justify with spaces to width
12
    printf(["%-12s]\n",         $h); // Left justify with spaces
    printf(["%012s]\n",          $h); // Pad with zeros
    printf(["%'#12s]\n\n",       $h); // Use the custom padding character
'#
    $d = 'Free More';          // The original creator of PHP
    printf("%12.8s]\n",          $d); // Right justify, cutoff of 8
characters
    printf(["%-12.12s]\n",        $d); // Left justify, cutoff of 12
characters
    printf(["%-@12.10s]\n",       $d); // Left justify, pad with '@', cutoff
10 chars
?>
```

Ca donne :

```
[Free]
[      Free]
[Free      ]
[00000000Free]
[#####Free]

[   Free Mor]
[Free More   ]
[Free More@@@]
```

3.1.12.2 Fonctions sur les dates

```
echo time();
echo mktime(0, 0, 0, 1, 1, 2000);
```

Les paramètres de mktime :

- The number of the hour (0–23)
- The number of the minute (0–59)
- The number of seconds (0–59)
- The number of the month (1–12)
- The number of the day (1–31)
- The year (1970–2038, or 1901–2038 with PHP 5.1.0+ on 32-bit signed systems)

Si on veut imposer un format de dates :

```
date($format, $timestamp);
```

Par exemple :

```
echo date("l F jS, Y - g:ia", time());
```

donnant ici par exemple "Thursday July 6th, 2017 - 1:38pm"

Quelques constantes de formats de dates :

DATE_ATOM

This is the format for Atom feeds. The PHP format is "Y-m-d\TH:i:sP" and example output is "2022-10-22T12:00:00+00:00".

DATE_COOKIE

This is the format for cookies set from a web server or JavaScript. The PHP format is "l, d-M-y H:i:s T" and example output is "Wednesday, 26-Oct-22 12:00:00 UTC".

DATE_RSS

This is the format for RSS feeds. The PHP format is "D, d M Y H:i:s O" and example output is "Wed, 26 Oct 2022 12:00:00 UTC".

DATE_W3C

This is the format for the World Wide Web Consortium. The PHP format is "Y-m-d\TH:i:sP" and example output is "2022-10-26T12:00:00+00:00".

Pour vérifier :

```
<?php
$month = 9;      // September (only has 30 days)
$day   = 31;      // 31st
$year  = 2022;    // 2022
if (checkdate($month, $day, $year)) echo "Date is valid";
else echo "Date is invalid";
```


3.1.12.3 Gestion des fichiers

```
if (file_exists("testfile.txt")) echo "File exists";
```

```
<?php // testfile.php
$fh = fopen("testfile.txt", 'w') or die("Failed to create file");
$text = <<<_END
Line 1
Line 2
Line 3
-END;
fwrite($fh, $text) or die("Could not write to file");
fclose($fh);
echo "File 'testfile.txt' written successfully";
?>
```

```
<?php // copyfile.php
copy('testfile.txt', 'testfile2.txt') or die("Could not copy
file");
echo "File successfully copied to 'testfile2.txt'";
?>
```

```
<?php // movefile.php
if (!rename('testfile2.txt', 'testfile2.new'))
    echo "Could not rename file";
else echo "File successfully renamed to 'testfile2.new'";
?>
```

```
<?php // deletefile.php
if (!unlink('testfile2.new')) echo "Could not delete file";
else echo "File 'testfile2.new' successfully deleted";
?>
```

Vérouiller un fichier :

```
<?php
$fh    = fopen("testfile.txt", 'r+') or die("Failed to open file");
$text = fgets($fh);
if (flock($fh, LOCK_EX))
{
    fseek($fh, 0, SEEK_END);
    fwrite($fh, "$text") or die("Could not write to file");
    flock($fh, LOCK_UN);
}
fclose($fh);
echo "File 'testfile.txt' successfully updated";
?>
```

Chargement d'un fichier image :

```
<?php // upload2.php
echo <<<_END
<html><head><title>PHP Form Upload</title></head><body>
<form method='post' action='upload2.php' enctype='multipart/form-
data'>
Select a JPG, GIF, PNG or TIF File:
<input type='file' name='filename' size='10'>
<input type='submit' value='Upload'>
</form>
_END;

if ($_FILES)
{
$name = $_FILES['filename']['name'];
switch($_FILES['filename']['type'])
{
    case 'image/jpeg': $ext = 'jpg'; break;
    case 'image/gif': $ext = 'gif'; break;
    case 'image/png': $ext = 'png'; break;
    case 'image/tiff': $ext = 'tif'; break;
    default:           $ext = ''; break;
}
if ($ext)
{
$n = "image.$ext";
move_uploaded_file($_FILES['filename']['tmp_name'], $n);
echo "Uploaded image '$name' as '$n':<br>";
echo "<img src='$n'>";
}
else echo "'$name' is not an accepted image file";
}
else echo "No image has been uploaded";
echo "</body></html>";
?>
```

Table 7-6. The contents of the `$_FILES` array

Array element	Contents
<code>\$_FILES['file']['name']</code>	The name of the uploaded file (e.g., <code>smiley.jpg</code>)
<code>\$_FILES['file']['type']</code>	The content type of the file (e.g., <code>image/jpeg</code>)
<code>\$_FILES['file']['size']</code>	The file's size in bytes
<code>\$_FILES['file']['tmp_name']</code>	The name of the temporary file stored on the server
<code>\$_FILES['file']['error']</code>	The error code resulting from the file upload

Table 7-7. Some common internet media content types

application/pdf	image/gif	multipart/form-data	text/xml
application/zip	image/jpeg	text/css	video/mpeg
audio/mpeg	image/png	text/html	video/mp4
audio/x-wav	image/tiff	text/plain	video/quicktime

3.1.12.4 Appels système

```
<?php // exec.php
$cmd = "dir"; // Windows
// $cmd = "ls"; // Linux, Unix & Mac
exec(escapeshellcmd($cmd), $output, $status);
if ($status) echo "Exec command failed";
else
{
    echo "<pre>";
    foreach($output as $line) echo htmlspecialchars("$line\n");
    echo "</pre>";
}
?>
```

3.2 Connexion BDD

3.2.1 MySQL

Références : [Tutoriel : Administrez vos bases de données avec MySQL,
<http://sdz.tdct.org/sdz/administrez-vos-bases-de-donnees-avec-mysql.html>;
https://www.w3schools.com/sql/sql_join.asp; [Learning PHP, MySQL, & JavaScript, WITH JQUERY, CSS & HTML5](#), Robin Nixon, 2018, 5ème Edition]

Une base de données informatique est un **ensemble de données** qui ont été stockées sur un support informatique, et **organisées et structurées** de manière à pouvoir facilement consulter et modifier leur contenu.

Un Système de Gestion de Base de Données (SGBD) est un logiciel (ou un ensemble de logiciels) permettant de manipuler les données d'une base de données. Manipuler, c'est-à-dire sélectionner et afficher des informations tirées de cette base, modifier des données, en ajouter ou en supprimer (ce groupe de quatre opérations étant souvent appelé "CRUD", pour Create, Read, Update, Delete).

MySQL est un système de gestion de bases de données.

3.2.1.1 Connexion à MySQL

La commande pour lancer le client est tout simplement son nom :

```
mysql -h localhost -u root -p
```

3.2.1.2 Création d'une base de données

La commande SQL pour créer une base de données est la suivante (définir l'encodage utilisé (l'UTF-8 dans notre cas)) :

```
CREATE DATABASE elevage CHARACTER SET 'utf8';
```

La suppression se fait via :

```
DROP DATABASE IF EXISTS elevage;
```

Pour utiliser une base de données, il suffit d'invoquer :

```
USE elevage
```

Pour afficher les bases de données :

```
SHOW databases;
```

Nous renvoyons le lecteur aux références spécialisées sur la conception des bases de données et notamment : (1) leur normalisation; (2) la gestion des transactions; (2) sauvegarde et réPLICATION;

Les commandes les plus utilisées pour manipuler les BDs :

Command	Action	Command	Action
ALTER	Alter a database or table		
BACKUP	Back up a table		
\c	Cancel input		
CREATE	Create a database		
DELETE	Delete a row from a table	RENAME	Rename a table
DESCRIBE	Describe a table's columns	SHOW	List details about an object
DROP	Delete a database or table	SOURCE	Execute a file
EXIT (Ctrl-C)	Exit	STATUS (\s)	Display the current status
GRANT	Change user privileges	TRUNCATE	Empty a table
HELP (\h, \?)	Display help	UNLOCK	Unlock table(s)
INSERT	Insert data	UPDATE	Update an existing record
LOCK	Lock table(s)		
QUIT (\q)	Same as EXIT	USE	Use a database

Pour créer un utilisateur, il suffit d'invoquer la commande :

```
GRANT PRIVILEGES ON database.object TO 'username'@'hostname'
IDENTIFIED BY 'password';
```

Par exemple :

```
GRANT ALL ON publications.* TO 'jim'@'localhost' IDENTIFIED BY
'mypasswd';
```

3.2.1.3 Création des tables

Soit une BD publications :

```
USE publications;
```

Soit un exemple d'une commande de création d'une table :

```
CREATE TABLE classics (
author VARCHAR(128),
title VARCHAR(128),
type VARCHAR(16),
year CHAR(4)) ENGINE InnoDB;
```

InnoDB est un moteur de bases de données relationnelles et transactionnelles, à l'image de celui utilisé par son concurrent Open Source PostgreSQL.

Pour vérifier que la création de la table :

```
DESCRIBE classics;
```

Types de données

Table 8-6. MySQL's CHAR data types

Data type	Bytes used	Examples
CHAR(<i>n</i>)	Exactly <i>n</i> (≤ 255)	CHAR(5) "Hello" uses 5 bytes CHAR(57) "Goodbye" uses 57 bytes
VARCHAR(<i>n</i>)	Up to <i>n</i> (≤ 65535)	VARCHAR(7) "Hello" uses 5 bytes VARCHAR(100) "Goodbye" uses 7 bytes

Table 8-7. MySQL's BINARY data types

Data type	Bytes used	Examples
BINARY(<i>n</i>)	Exactly <i>n</i> (≤ 255)	As CHAR but contains binary data
VARBINARY(<i>n</i>)	Up to <i>n</i> (≤ 65535)	As VARCHAR but contains binary data

Table 8-8. MySQL's TEXT data types

Data type	Bytes used	Attributes
TINYTEXT(<i>n</i>)	Up to <i>n</i> (≤ 255)	Treated as a string with a character set
TEXT(<i>n</i>)	Up to <i>n</i> (≤ 65535)	Treated as a string with a character set
MEDIUMTEXT(<i>n</i>)	Up to <i>n</i> ($\leq 1.67e+7$)	Treated as a string with a character set
LONGTEXT(<i>n</i>)	Up to <i>n</i> ($\leq 4.29e+9$)	Treated as a string with a character set

Table 8-9. MySQL's BLOB data types

Data type	Bytes used	Attributes
TINYBLOB(n)	Up to n (≤ 255)	Treated as binary data—no character set
BLOB(n)	Up to n (≤ 65535)	Treated as binary data—no character set
MEDIUMBLOB(n)	Up to n ($\leq 1.67e+7$)	Treated as binary data—no character set
LONGBLOB(n)	Up to n ($\leq 4.29e+9$)	Treated as binary data—no character set

Table 8-10. MySQL's numeric data types

Data type	Bytes used	Minimum value		Maximum value	
		Signed	Unsigned	Signed	Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8.38e+6	0	8.38e+6	1.67e+7
INT / INTEGER	4	-2.15e+9	0	2.15e+9	4.29e+9
BIGINT	8	-9.22e+18	0	9.22e+18	1.84e+19
FLOAT	4	-3.40e+38	n/a	3.4e+38	n/a
DOUBLE / REAL	8	-1.80e+308	n/a	1.80e+308	n/a

Table 8-11. MySQL's DATE and TIME data types

Data type	Time/date format
DATETIME	'0000-00-00 00:00:00'
DATE	'0000-00-00'
TIMESTAMP	'0000-00-00 00:00:00'
TIME	'00:00:00'
YEAR	0000 (Only years 0000 and 1901–2155)

3.2.1.4 Champs autoincrement

Un champs autoincrément est un champs qui prend la valeur de la valeur de la dernière insertion plus un.

```
ALTER TABLE classics ADD id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY;
```

Un autre exemple :

```
CREATE TABLE classics (
    author VARCHAR(128),
    title VARCHAR(128),
    type VARCHAR(16),
    year CHAR(4),
    id INT UNSIGNED NOT NULL AUTO_INCREMENT KEY) ENGINE InnoDB;
```

3.2.1.5 Ajout de données dans une table

```
INSERT INTO classics(author, title, type, year)
VALUES('Mark Twain', 'The Adventures of Tom
Sawyer', 'Fiction', '1876');
```

```
INSERT INTO classics(author, title, type, year)
VALUES('Jane Austen', 'Pride and Prejudice', 'Fiction', '1811');
```

```
INSERT INTO classics(author, title, type, year)
VALUES('Charles Darwin', 'The Origin of Species', 'Non-
Fiction', '1856');
```

```
INSERT INTO classics(author, title, type, year)
VALUES('Charles Dickens', 'The Old Curiosity Shop', 'Fiction', '1841');
```

```
INSERT INTO classics(author, title, type, year)
VALUES('William Shakespeare', 'Romeo and Juliet', 'Play', '1594');
```

3.2.1.6 Modifications dans une table

Changer le nom de la table :

```
ALTER TABLE classics RENAME pre1900;
```

Changer le type de données :

```
ALTER TABLE classics MODIFY year SMALLINT;
```

Ajouter une nouvelle colonne :

```
ALTER TABLE classics ADD pages SMALLINT UNSIGNED;
```

Supprimer une colonne :

```
ALTER TABLE classics DROP pages;
```

Supprimer une table :

```
DROP TABLE disposable;
```

3.2.1.7 Les indexes

Tout l'intérêt des index est **d'accélérer les requêtes** qui utilisent des colonnes indexées comme critères de recherche. Par conséquent, si vous savez que dans votre application, vous ferez énormément de recherches sur la colonne *X*, ajoutez donc un index sur cette colonne, vous ne vous en porterez que mieux. Les index permettent aussi d'assurer l'intégrité des données de la base. Pour cela, il existe en fait plusieurs types d'index différents, et deux types de "clés". Lorsque je parle de garantir l'intégrité de vos données, cela signifie en gros garantir la qualité de vos données, vous assurer que vos données ont du sens. Par exemple, soyez sûrs que vous ne faites pas référence à un client dans la table *Commande*, alors qu'en réalité, ce client n'existe absolument pas dans la table *Client*.

Exemple :

```
ALTER TABLE classics ADD INDEX(author(20));
ALTER TABLE classics ADD INDEX(title(20));
ALTER TABLE classics ADD INDEX(category(4));
ALTER TABLE classics ADD INDEX(year);
DESCRIBE classics;
```

Ajout d'indexes au moment de la création :

```
CREATE TABLE classics (
    author VARCHAR(128),
    title VARCHAR(128),
    category VARCHAR(16),
    year SMALLINT,
    INDEX(author(20)),
    INDEX(title(20)),
    INDEX(category(4)),
    INDEX(year)) ENGINE InnoDB;
```

3.2.1.8 Clés primaires et étrangères

Nous allons passer à une autre notion très importante : les clés qui sont liées aux index. Et tout comme NOT NULL et les index UNIQUE, les clés font partie de ce qu'on appelle les contraintes.

Il existe deux types de clés :

- les clés primaires, qui ont déjà été survolées lors du chapitre sur la création d'une table, et qui servent à identifier une ligne de manière unique ;
- les clés étrangères, qui permettent de gérer des relations entre plusieurs tables, et garantissent la cohérence des données.

Exemple :

```
ALTER TABLE classics ADD isbn CHAR(13) PRIMARY KEY;
```

Exemple :

```
ALTER TABLE classics ADD isbn CHAR(13);
UPDATE classics SET isbn='9781598184891' WHERE year='1876';
UPDATE classics SET isbn='9780582506206' WHERE year='1811';
UPDATE classics SET isbn='9780517123201' WHERE year='1856';
UPDATE classics SET isbn='9780099533474' WHERE year='1841';
UPDATE classics SET isbn='9780192814968' WHERE year='1594';
ALTER TABLE classics ADD PRIMARY KEY(isbn);
DESCRIBE classics;
```

3.2.1.9 Interrogation d'une BD : SELECT

La syntaxe basique de SELECT est :

```
SELECT something FROM tablename;
```

La requête SELECT est très riche en syntaxe. Nous renvoyons le lecteur au manuel de référence du langage SQL et des sites de référence sur mysql. Voici quelques usages :

Exemple :

```
SELECT author,title FROM classics;
```

```
SELECT title,isbn FROM classics;
```

```
SELECT COUNT(*) FROM classics;
```

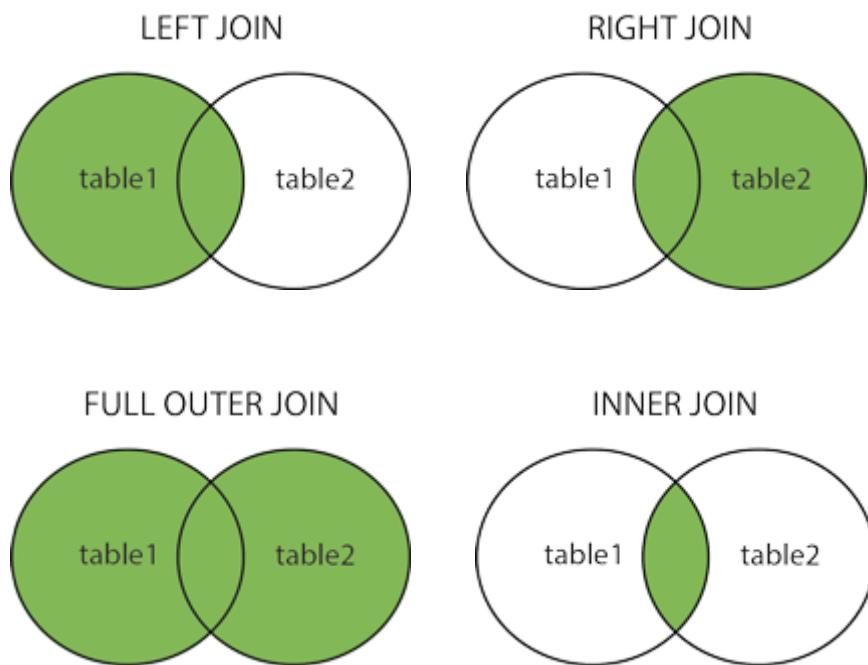
```
SELECT DISTINCT author FROM classics;
```

```
SELECT author,title FROM classics ORDER BY author;
```

```
SELECT category,COUNT(author) FROM classics GROUP BY category;
```


3.2.1.10 Jointures

Pour la jointure entre plusieurs tables :



Voici les différents types de jointure dans SQL:

- **(INNER) JOIN:** (par défaut) retourne les tuples qui vérifient les deux tables à la fois
(Returns records that have matching values in both tables)
- **LEFT (OUTER) JOIN:** retourne les tuples de la table gauche qui vérifient la table de droite
(Return all records from the left table, and the matched records from the right table)
- **RIGHT (OUTER) JOIN:** retourne les tuples de la table de droite qui vérifient la table gauche
(Return all records from the right table, and the matched records from the left table)
- **FULL (OUTER) JOIN:** retourne les tuples qui vérifient une des deux tables
(Return all records when there is a match in either left or right table)

Exemples :

```
CREATE TABLE customers (
    name VARCHAR(128),
    isbn VARCHAR(13),
    PRIMARY KEY (isbn)) ENGINE InnoDB;
INSERT INTO customers(name,isbn)
    VALUES('Joe Bloggs','9780099533474');
INSERT INTO customers(name,isbn)
    VALUES('Mary Smith','9780582506206');
INSERT INTO customers(name,isbn)
    VALUES('Jack Wilson','9780517123201');
SELECT * FROM customers;
```

```
INSERT INTO customers(name,isbn) VALUES
    ('Joe Bloggs','9780099533474'),
    ('Mary Smith','9780582506206'),
    ('Jack Wilson','9780517123201');
```

```
SELECT name,author,title FROM customers
JOIN classics ON customers.isbn=classics.isbn;
```

3.2.1.11 Suppression dans une table

Exemple :

```
DELETE FROM classics WHERE title='Little Dorrit';
```

3.2.1.12 Mise-à-jour d'une table

Exemple :

```
UPDATE classics SET author='Mark Twain (Samuel Langhorne Clemens)'
WHERE author='Mark Twain';
```

3.2.2 Postgres

3.2.3 Accès BDD via PHP

Références [

<https://www.numelion.com/se-connecter-a-une-base-de-donnees-en-php.html> ;

<https://www.php.net/manual/fr/>;

[Learning PHP, MySQL, & JavaScript, WITH JQUERY, CSS & HTML5, Robin Nixon, 2018, 5ème Edition](#)]

La connexion à sa base de données est une étape courante qui permet de récupérer des informations dans cette dernière par le biais d'une communication entre le langage PHP et MySQL. Le langage PHP a évolué dans le temps, et la méthode pour se connecter (le code) à également évolué.

Désormais, pour se connecter à une base de données MySQL en PHP on utilise l'extension PDO. Celle-ci permet une connexion de manière simple et sécurisée au serveur. Avec cette extension, il est possible de se connecter sur différents types de serveur SGBDR (Système de gestion de bases de données relationnelles).

Vous pourrez par exemple vous connecter sur un serveur MySQL, SQLite, Microsoft SQL, ODBC, Firebird, PostgreSQL...

Dans un premier temps, il faut tester l'extension PDO pour s'assurer que celle-ci est bien fonctionnelle sur le serveur. La plupart des serveurs web l'ont installés par défaut, mais il vaut mieux s'en assurer afin d'éviter une erreur inutile. Pour cela, vous devez utiliser le code ci-dessous :

```
<?php  
if (extension_loaded ('PDO')) {  
    echo 'PDO OK';  
} else {  
    echo 'PDO KO';  
}  
?>
```

Pour établir la connexion, vous allez devoir créer un DSN. Il s'agit en anglais du « Data Source Name ». Ce sont des paramètres permettant d'établir la connexion. Vous allez devoir renseigner plusieurs informations.

```
$dsn = 'mysql:host=ADRESSE_DU_SERVEUR;dbname=VOTRE_BASE_DE_DONNEES;  
port=VOTRE_PORT;charset=VOTRE_ENCODAGE';
```

Maintenant, nous allons pouvoir créer la connexion au serveur MySQL. A la suite de cette dernière, nous allons effectuer une requête et afficher le résultat. Dans un premier temps, il faut renseigner les paramètres permettant la connexion. Ceux-ci seront stockés dans une variable. Pour faire simple, on la nommera « pdo ». Voici un exemple :

```
$pdo = new PDO($dsn, 'VOTRE_NOM_UTILISATEUR_SERVEUR' ,  
'VOTRE_MOT_DE_PASSE');
```

Vous pouvez vérifier si des erreurs de relation entre votre script PHP et la base de données existent. Vous pouvez même en être informé par e-mail. Idéal si votre serveur de base de données ne fonctionne plus. Vous serez rapidement informé. Si vous utilisez le script sur un serveur en ligne, il faut utiliser les blocs « try / catch » comme dans l'exemple ci-dessous.

```
try {  
    $pdo = new PDO($dsn, 'VOTRE_NOM_UTILISATEUR_SERVEUR' ,  
'VOTRE_MOT_DE_PASSE');  
}  
catch (PDOException $exception) {  
    mail('VOTRE_EMAIL', 'PDOException', $exception->getMessage());  
    exit('Erreur de connexion à la base de données');  
}
```

Si aucune erreur n'apparaît, nous sommes normalement connectés. Le meilleur moyen de vérifier est d'en avoir le cœur net en utilisant une requête et en affichant le résultat. Nous allons utiliser une requête pour sélectionner des clients dans notre table. Pour cela, il faut utiliser une fonction qui se nomme « query » avec PDO. Le résultat sera stocké dans une variable. Nous verrons comment la réutiliser pour afficher ce dernier dans notre page. Voici le code à utiliser.

```
$query = $pdo->query("VOTRE_REQUETE_MYSQL");  
$resultat = $query->fetchAll();
```

Avec l'ensemble des éléments que nous avons vus précédemment, nous pouvons créer un script complet. Celui-ci va donc se connecter à une base de données MySQL en PHP. Il s'assurera qu'il n'y a pas d'erreur, puis exécuter une requête pour extraire les clients masculins. On affichera plusieurs informations comme le nom et prénom dans un tableau. Voici le code complet que vous pouvez utiliser. Il y a des commentaires pour permettre une meilleure compréhension.

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

<?php

// Création du DSN
$dsn = 'mysql:host=localhost;dbname=commerce;port=3306;charset=utf8';
// Création et test de la connexion
try {
    $pdo = new PDO($dsn, 'root' , '');
}
catch (PDOException $exception) {
    mail('fauxmail@votreemail.com', 'PDOException', $exception->getMessage());
    exit('Erreur de connexion à la base de données');
}
// Requête pour tester la connexion
$query = $pdo->query("SELECT * FROM `clients` WHERE `gender` LIKE 'male'");
$resultat = $query->fetchAll();

//Afficher le résultat dans un tableau
print("<table border=\"1\">");
foreach ($resultat as $key => $variable)
{
    print("<tr>");
    print("<td>".$resultat[$key]['gender']."</td>");
    print("<td>".$resultat[$key]['title']."</td>");
    print("<td>".$resultat[$key]['givenname']."</td>");
    print("<td>".$resultat[$key]['surname']."</td>");
    print("</tr>");
}
print("</table>");
```

```
?>
</body>
</html>
```

3.3 Interaction clients/serveur et communication asynchrone

Références :

[<http://sdz.tdct.org/sdz/ajax-et-l-echange-de-donnees-en-javascript.html>]

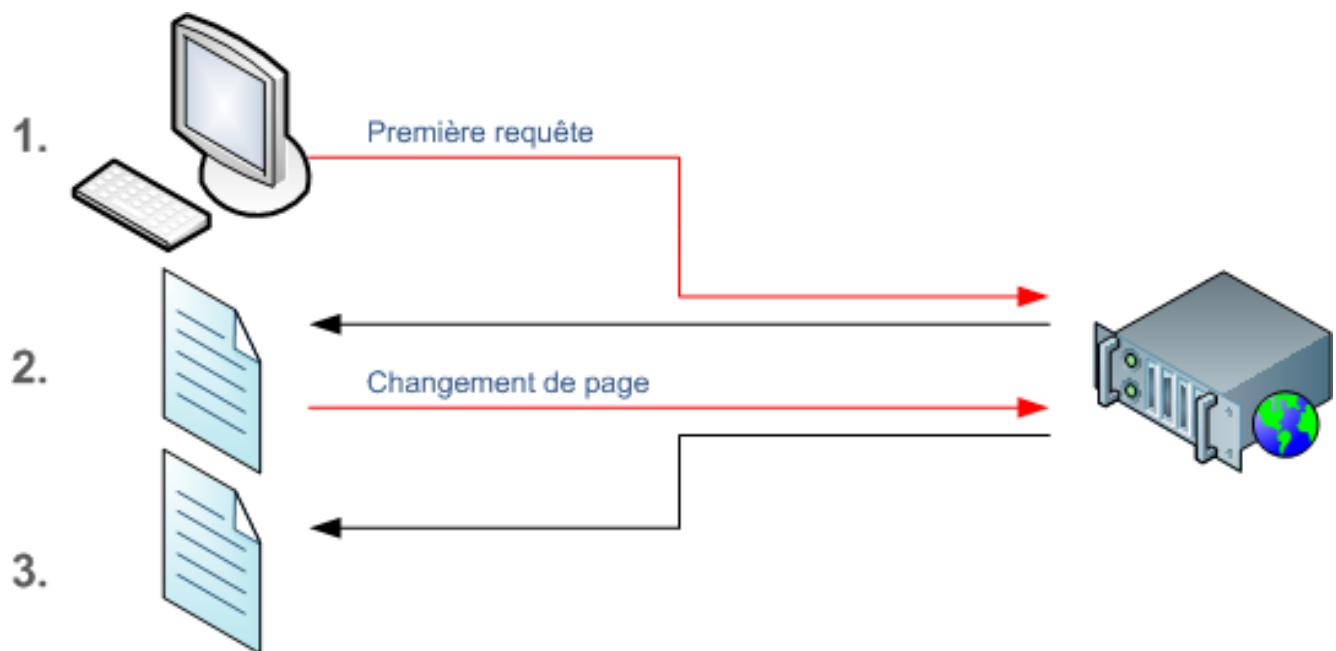
[https://www.tutorialspoint.com/php/php_and_ajax.htm]

[https://www.w3schools.com/php/php_ajax_php.asp]

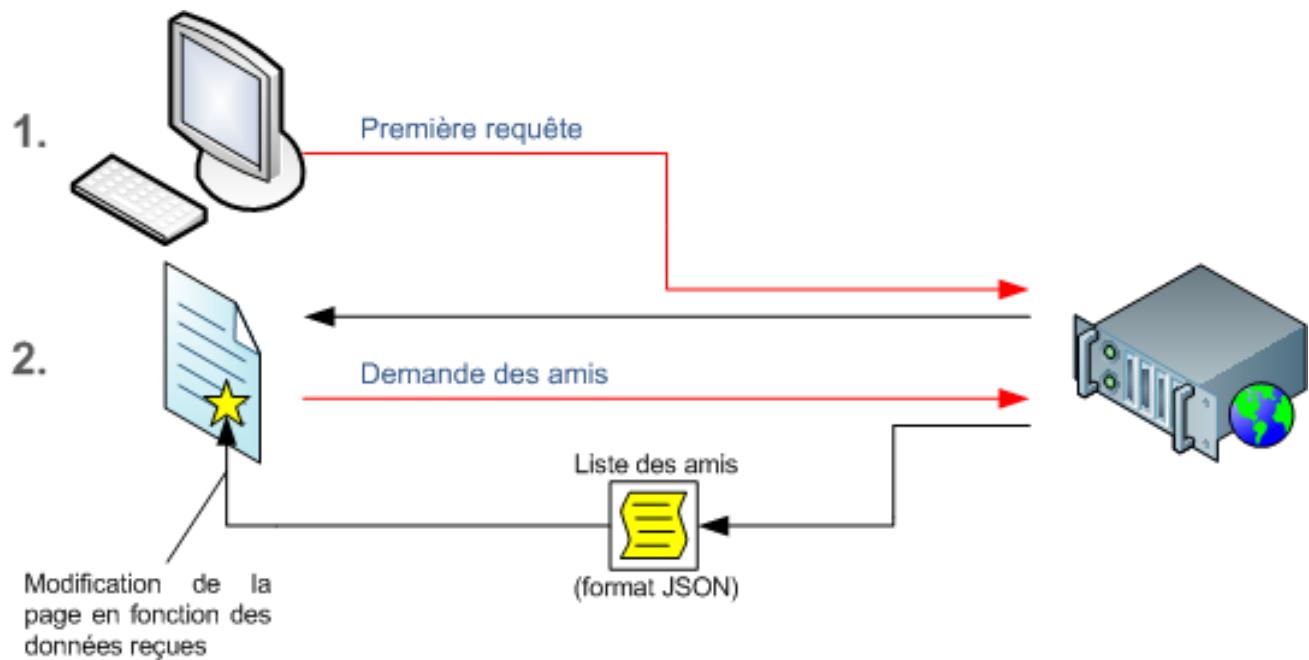
- [<https://code.tutsplus.com/tutorials/how-to-use-ajax-in-php-and-jquery--cms-32494>]

- [<http://thisinterestsme.com/ajax-search-mysql-php-jquery/>]

Site Web traditionnel : Tout d'abord le navigateur envoie une requête – via une URL – au serveur. Le serveur répond en renvoyant au navigateur le code HTML de la page ainsi que tout ce qui lui est associé comme les scripts JavaScript, les images ou les éventuels médias et autres objets embarqués – donc la réponse du serveur est beaucoup plus volumineuse que la requête. Le navigateur affiche la page et l'utilisateur peut la parcourir quelques instants avant de cliquer sur un lien hypertexte qui enverra une nouvelle requête au serveur qui lui-même renverra le HTML correspondant... et ainsi de suite.



Application Ajax : La première requête est la même – ça on ne sait rien y faire. La différence va résider dans le fait que quand l'utilisateur cliquera sur un lien – ou un autre élément cliquable – la page ne se rechargera pas et le navigateur enverra une requête au serveur, lequel renverra les données demandées dans un format léger – comme le format JSON. Dans ce cas, le serveur n'aura renvoyé qu'un minimum de données ce qui est beaucoup plus léger et donc plus rapide. Le navigateur, par le biais de JavaScript, peut alors mettre à jour une petite partie de la page avec les données reçues du serveur.



3.3.1 Dialoguer avec le serveur

Le navigateur et le serveur ne peuvent se parler que via un format de type texte brut. Le navigateur peut donc demander au serveur "Liste amis Alyx" et le serveur renverra la liste des amis d'Alyx. C'est assez nul comme format : si la question est claire, la liste des amis que renverra le serveur le sera nettement moins et c'est ici qu'il va falloir opter pour le format adéquat. Plusieurs formats sont possibles :

- Texte simple ;
- HTML ;
- XML ;
- JSON.

3.3.1.1 Le HTML

Le HTML est intéressant car il suffira de l'insérer directement dans la page avec la propriété innerHTML. C'est rapide. Cela dit le HTML est verbeux et peut se révéler assez lourd quand il s'agit d'un grand volume de données.

```
<ul>
  <li><span title="a4242">Gordon</span></li>
  <li><span title="j3781">Barney</span></li>
  <li><span title="j7638">Eli</span></li>
  <li><span title="o7836">Chell</span></li>
  <li><span title="e5831">Odessa</span></li>
</ul>
```

3.3.1.2 Le XML

Avec certains objets AJAX, comme XMLHttpRequest il est possible de récupérer du texte sous forme de XML et de l'interpréter comme tel, ce qui permet de manipuler les données avec les fonctions DOM. Ca peut être intéressant mais XML souffre du même problème que le HTML : il est verbeux. De plus le traitement via DOM peut se révéler assez lent s'il s'agit d'un grand volume de données et suivant le navigateur utilisé par le client.

```
<friends>
  <f name="Gordon" id="a4242" />
  <f name="Barney" id="j3781" />
  <f name="Eli" id="j7638" />
  <f name="Chell" id="o7836" />
  <f name="Odessa" id="e5831" />
</friends>
```

3.3.1.3 Le JSON

Reste le JSON. Le JSON est une manière de structurer l'information en utilisant la syntaxe objet de JavaScript – des objets et des *tableaux associatifs*. JSON est très léger, car non-verbeux mais nécessite d'être évalué par le compilateur JavaScript pour pouvoir être utilisé comme un objet. L'évaluation se fait via eval pour les navigateurs obsolètes ou via la méthode parse de l'objet natif JSON. L'évaluation est souvent décriée car peut se révéler dangereuse, mais dans la mesure où vous connaissez la source des données à évaluer il n'y a pas de danger.

```
[  
  { "name": "Gordon", "id": "a4242" },  
  { "name": "Barney", "id": "j3781" },  
  { "name": "Eli", "id": "j7638" },  
  { "name": "Chell", "id": "o7836" },  
  { "name": "Odessa", "id": "e5831" }  
]
```

Le JSON est donc le format travaillant de paire avec AJAX quand il s'agit de recevoir des données classées et structurées. Les autres formats peuvent bien évidemment servir et se révéler intéressants dans certains cas, mais d'une façon générale les grandes pointures du JavaScript, comme Douglas Crockford incitent à utiliser JSON.

3.3.2 Principes synchrones et asynchrones

La principale particularité d'AJAX est l'asynchronisme : la fonction qui envoie une requête au serveur n'est pas la même que celle qui en recevra la réponse. Avant d'aborder la pratique d'AJAX, il est bon de bien cerner cette notion d'asynchronisme qui est très importante.

Quand un programme ou un script s'exécute, il appelle les différentes instructions dans l'ordre dans lequel elles sont placées :

```
var plop = 0; // première instruction  
plop += 2; // deuxième  
alert(plop); // et troisième
```

Maintenant, imaginons qu'il y ait un appel de fonction :

```
var plop = 0; // première instruction  
plop = additionner(plop, 2); // deuxième  
alert(plop); // et troisième
```

Quand la fonction additionner est appelée, le script principal se met en pause, et attend que la fonction soit exécutée, et qu'elle ait renvoyé une valeur (si elle ne renvoie rien, c'est pareil).

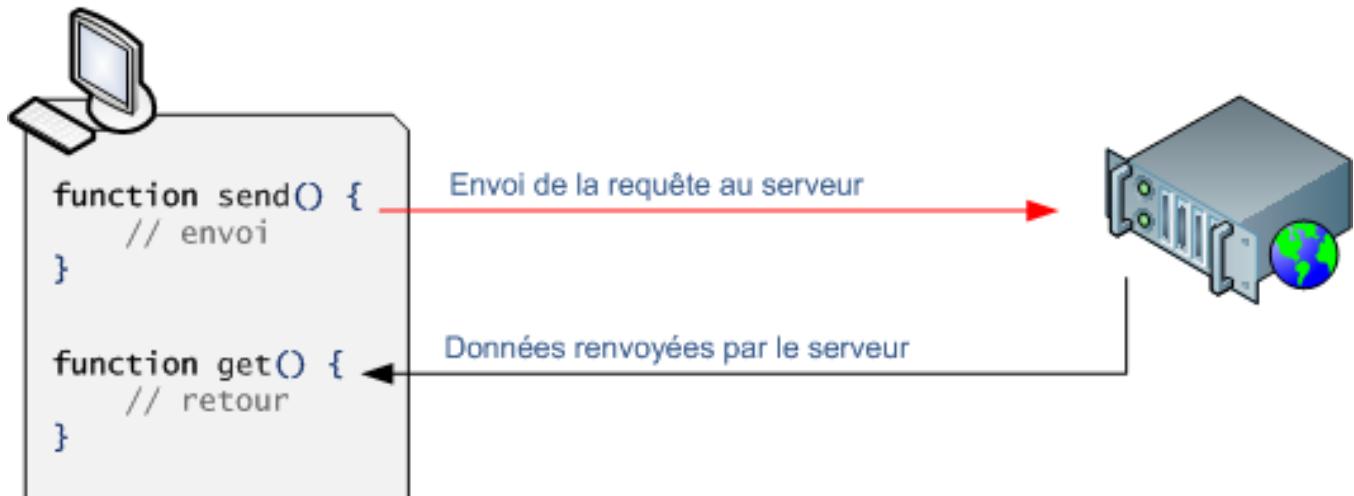
On dit que le script est exécuté de façon synchrone : quand un appel externe au script principal est réalisé, le script en attend la réponse ou la fin de l'exécution.

Le contraire de synchrone est asynchrone. **Quand un appel est asynchrone, le script principal n'attend pas d'avoir reçu les données pour continuer.** Evidemment, si mon exemple synchrone marche bien avec des fonctions, il ne marche pas si le script est asynchrone ; imaginons donc une requête de type AJAX !

Le script s'exécute et rencontre une requête AJAX, laquelle est envoyée en mode asynchrone. Dans ce cas, la requête est envoyée, mais le script n'attend pas que la requête ait abouti, il continue quoi qu'il arrive. L'intérêt est que si la requête met quelques secondes à être traitée par le serveur, le script n'est pas ralenti.

Ainsi, quand la requête est envoyée, le script continue son exécution, et quand la requête renvoie quelque chose, c'est la fonction de callback qui est appelée, et c'est elle qui va faire "suite" au script principal, en traitant les informations renvoyées.

On peut résumer l'asynchronisme en AJAX par ce schéma :



3.3.3 L'objet XMLHttpRequest

Le principe de fonctionnement d'XMLHttpRequest est d'envoyer une requête HTTP vers le serveur, et une fois la requête envoyée, les données renvoyées par le serveur peuvent être récupérées. Pour ce faire, il faut disposer d'un objet disposant de cette fonctionnalité.

Pour la suite du tutoriel, j'abrègerai XMLHttpRequest par XHR, c'est plus simple.

Pour instancier (créer, déclarer) un objet XHR, on procède de la même façon que pour n'importe quel objet JavaScript à savoir avec le mot-clé `new` :

```
var xhr = new XMLHttpRequest();
```

Il y a deux façons d'instancier un objet XHR avec un contrôle ActiveX et elles dépendent de la version d'XMLHTTP utilisée. Pour faire simple, on va utiliser un `try...catch`, l'instanciation indiquée dans le `try` étant la plus récente :

```
try {
    var xhr = new ActiveXObject("Msxml2.XMLHTTP");
} catch(e) {
    var xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Pour faire un script homogène, rassemblons ce code en un seul, en prenant soin de tester la prise en charge des différentes méthodes d'instanciation :

```
var xhr = null;

if (window.XMLHttpRequest || window.ActiveXObject) {
    if (window.ActiveXObject) {
        try {
            xhr = new ActiveXObject("Msxml2.XMLHTTP");
        } catch(e) {
            xhr = new ActiveXObject("Microsoft.XMLHTTP");
        }
    } else {
        xhr = new XMLHttpRequest();
    }
} else {
    alert("Votre navigateur ne supporte pas l'objet
XMLHttpRequest...");
    return;
}
```

Par la suite, j'utiliserai une fonction qui retournera l'objet XMLHttpRequest instancié, ce sera plus simple :

```
function getXMLHttpRequest() {  
    var xhr = null;  
  
    if (window.XMLHttpRequest || window.ActiveXObject) {  
        if (window.ActiveXObject) {  
            try {  
                xhr = new ActiveXObject("Msxml2.XMLHTTP");  
            } catch(e) {  
                xhr = new ActiveXObject("Microsoft.XMLHTTP");  
            }  
        } else {  
            xhr = new XMLHttpRequest();  
        }  
    } else {  
        alert("Votre navigateur ne supporte pas l'objet  
XMLHttpRequest...");  
        return null;  
    }  
  
    return xhr;  
}
```

Par la suite, pour instancier un objet XHR, il suffira de faire :

```
var xhr = getXMLHttpRequest();
```

3.3.4 Envoi d'une requête http

Dans un premier temps, il faut définir les modalités d'envoi avec la méthode `open`, et on l'enverra ensuite avec la méthode `send` :

```
var xhr = getXMLHttpRequest(); // Voyez la fonction
getXMLHttpRequest() définie dans la partie précédente

xhr.open("GET", "handlingData.php", true);
xhr.send(null);
```

`open` s'utilise de cette façon : `open(sMethod, sUrl, bAsync)`

- `sMethod` : la méthode de transfert : GET ou POST;
- `sUrl` : la page qui donnera suite à la requête. Ça peut être une page dynamique (PHP, CFM, ASP) ou une page statique (TXT, XML...);
- `bAsync` : définit si le mode de transfert est asynchrone ou non (synchrone). Dans ce cas, mettez `true`. Ce paramètre est optionnel et vaut `true` par défaut, mais il est courant de le définir quand même (je le fais par habitude).

Si vous utilisez la méthode POST, vous devez absolument changer le type MIME de la requête avec la méthode `setRequestHeader`, sinon le serveur ignorera la requête :

```
xhr.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
```

3.3.4.1 Passer des variables

Vous avez la possibilité de passer des variables au serveur. Suivant la méthode d'envoi utilisée une petite différence fait son apparition. Dans le cas de GET les variables sont transmises directement dans l'URL :

```
xhr.open("GET", "handlingData.php?variable1=truc&variable2=bidule",
true);
xhr.send(null);
```

Pour POST, il faut spécifier les variables dans l'argument de `send` :

```
xhr.open("POST", "handlingData.php", true);
xhr.setRequestHeader("Content-Type", "application/x-www-form-
urlencoded");
xhr.send("variable1=truc&variable2=bidule");
```

3.3.4.2 Protéger les caractères

Avant de passer des variables, il est important de les protéger pour conserver les caractères spéciaux et les espaces. Pour cela, utilisez la fonction globale `encodeURIComponent`, comme ceci :

```
var sVar1 = encodeURIComponent("contenu avec des espaces");
var sVar2 = encodeURIComponent("je vois que vous êtes un bon élève...
oopa !");

xhr.open("GET", "handlingData.php?variable1=" + sVar1 + "&variable2=
" + sVar2, true);
xhr.send(null);
```

3.3.5 Traitement côté serveur (PHP)

Une page PHP se comporte comme une page PHP normale, il n'y a rien de spécifique. La récupération des variables se fait d'une façon classique (via `$_GET` ou `$_POST`, suivant la méthode d'envoi) et il ne faut pas perdre de vue de sécuriser la page, car un petit malin pourrait l'appeler directement sans passer par l'appel via XMLHttpRequest (il peut donc mettre directement l'URL de la page) :

```
<?php
header("Content-Type: text/plain"); // Utilisation d'un header pour
spécifier le type de contenu de la page. Ici, il s'agit juste de
texte brut (text/plain).
$variable1 = (isset($_GET["variable1"])) ? $_GET["variable1"] : NULL;
$variable2 = (isset($_GET["variable2"])) ? $_GET["variable2"] : NULL;
if ($variable1 && $variable2) {
    // Faire quelque chose...
    echo "OK";
} else {
    echo "FAIL";
}
?>
```

Récapitulons : en 1, on a envoyé la requête, et en 2 la requête a trouvé des données à récupérer (les données fournies par le fichier PHP, TXT ou XML).

Il faut savoir que quand on envoie une requête HTTP via XMLHttpRequest, celle-ci passe par plusieurs états différents :

- 0 : L'objet XHR a été créé, mais pas encore initialisé (la méthode open n'a pas encore été appelée)
- 1 : L'objet XHR a été créé, mais pas encore envoyé (avec la méthode send)
- 2 : La méthode send vient d'être appelée
- 3 : Le serveur traite les informations et a commencé à renvoyer des données
- 4 : Le serveur a fini son travail, et toutes les données sont réceptionnées

En vue de cela, il va donc nous falloir détecter les changements d'état pour savoir où en est la requête. Pour cela, on va utiliser la propriété `onreadystatechange`, et à chaque changement d'état (*state* en anglais), on regardera lequel il s'agit :

```
var xhr = getXMLHttpRequest();

xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {
        alert("OK"); // C'est bon \o/
    }
};

xhr.open("GET", "handlingData.php", true);
xhr.send(null);
```

On utilise readyState pour connaître l'état de la requête. En addition, nous devons aussi vérifier le code d'état (comme le fameux code 404 pour les pages non trouvées ou le code 500 pour l'erreur de serveur) de la requête, pour vérifier si tout s'est bien passé. Pour cela, on utilise la propriété status. Si elle vaut 200 ou 0 (aucune réponse, pratique pour les tests en local, sans serveur d'évaluation), tout est OK.

3.3.5.1 Récupérer les données

Rien de plus simple, il suffit d'utiliser une des deux propriétés disponibles :

- `responseText` : pour récupérer les données sous forme de texte brut
- `responseXML` : pour récupérer les données sous forme d'arbre XML

L'utilisation est assez simple, il suffit de procéder comme ceci :

```
xhr.onreadystatechange = function() {
    if (xhr.readyState == 4 && (xhr.status == 200 || xhr.status == 0)) {
        alert(xhr.responseText); // Données textuelles récupérées
    }
};
```

3.3.6 Une fonction de callback

Le alert c'est bien beau, mais à part vérifier si la réception est bonne, ça ne sert à rien. Si on s'amuse à récupérer des données, c'est pour les traiter ! Le traitement peut se faire directement dans la fonction anonyme définie dans **onreadystatechange**. Mais c'est très laid et ça n'a rien à faire là, il vaut mieux séparer les codes pour avoir deux fonctions concises plutôt qu'une grosse. On va donc définir ce qu'on appelle une fonction dite "de callback". Une fonction de callback est une fonction de rappel, exécutée généralement après qu'un script a été exécuté. Ce n'est pas très clair, mais c'est tout simple en fait : il s'agit de passer à une fonction A le nom d'une fonction B qui sera lancée une fois que la fonction A aura été exécutée. Comme ceci en gros :

```
function funcA(callback) {
    // instruction...
    // instruction...
    // instruction...
    callback();
}

function funcB() {
    alert("Plop");
}

funcA(funcB);
```

En me basant sur cet exemple de callback, il est donc facile de le transposer pour utiliser XMLHttpRequest :

```
function request(callback) {
    var xhr = getXMLHttpRequest();
    xhr.onreadystatechange = function() {
        if (xhr.readyState == 4 && (xhr.status == 200 || 
            xhr.status == 0)) {
            callback(xhr.responseText);
        }
    };
    xhr.open("GET", "handlingData.php", true);
    xhr.send(null);
}

function readData(sData) {
    // On peut maintenant traiter les données sans encombrer l'objet
    XHR.
    if (sData == "OK") {
        alert("C'est bon");
    } else {
        alert("Y'a eu un problème");
    }
}

request(readData);
```

3.3.7 Ajax avec jQuery

Références :

- Un site web dynamique avec jQuery !

<https://openclassrooms.com/fr/courses/1567926-un-site-web-dynamique-avec-jquery>

La fonction `$.ajax()` va être incontournable pour nos appels AJAX en jQuery, nous l'employons pour envoyer une requête HTTP. Nous allons désormais avoir très souvent recours à cette fonction. Le code ci-dessous comporte un squelette pour effectuer les requêtes Ajax.

```
$("#more_com").click(function () {
    $.ajax({
        url: 'more_com.php',
        type: 'GET',
        dataType: 'html',

        success: function (code_html, statut) {

        },
        error: function (resultat, statut, erreur) {

        },
        complete: function (resultat, statut) {

        }
    });
});
```

jQuery nous propose un paramètre pour gérer le retour de notre fonction `$.ajax()` en cas de réussite. Le paramètre s'appelle `success`, et il ne sera exécuté QUE si l'appel AJAX a abouti. Ce paramètre prend en charge une fonction qui sera exécutée. C'est logique, lorsque l'appel AJAX aura réussi, cette fonction est automatiquement appelée ; c'est elle que l'on utilise pour mettre à jour notre page !

Si l'appel AJAX a rencontré une erreur, on va avoir de quoi gérer l'erreur de la même manière qu'avec `success`. Le paramètre qui va être employé ici sera `error`. Le paramètre exécute une fonction si l'appel AJAX a échoué.

Vous le devinez, `complete` va s'exécuter une fois l'appel AJAX effectué.

Le paramètre va prendre deux arguments, l'objet `resultat` ainsi qu'un `statut`.

3.3.8 Exemple d'une API Serveur/Client

Soit une application manipulant une BD utilisateur à l'université. Ci-dessous le processus de création de la BD.

La commande pour lancer le client est tout simplement son nom :

```
mysql -h localhost -u root -p
```

La commande SQL pour créer une base de données est la suivante (définir l'encodage utilisé (l'UTF-8 dans notre cas)) :

```
CREATE DATABASE bduniv CHARACTER SET 'utf8';
```

Pour utiliser la base de données, il suffit d'invoquer :

```
USE bduniv
```

Puis la création du schéma :

```
CREATE TABLE ajax_example (
    nom varchar(50) NOT NULL,
    age int(11) NOT NULL,
    email varchar(50) NOT NULL,
    enseignantEtudiant varchar(1) NOT NULL,
    pass varchar(50) NOT NULL,
    PRIMARY KEY (email)
)
```

```
INSERT INTO ajax_example VALUES ('Omar Gali', 50, 'Omar.Gali@univ-oran.com', 'E', '123');
INSERT INTO ajax_example VALUES ('Mourad Adda', 45, 'Mourad.Adda@univ-oran.com', 'E', '456');
INSERT INTO ajax_example VALUES ('Mohammed Dhali', 30, 'Mohammed.Dhali@univ-oran.com', 'E', '789');
INSERT INTO ajax_example VALUES ('Reda Allab', 20, 'Reda.Allab@univ-oran.com', 'T', '159');
INSERT INTO ajax_example VALUES ('Samir Alioua', 22, 'Samir.Alioua@univ-oran.com', 'T', '357');
INSERT INTO ajax_example VALUES ('Ali Djar', 19, 'Ali.Djar@univ-oran.com', 'T', '793');
```

Soit le programme “ajax.html“ qui interagit avec le serveur. Analysons le :

```
<html>
<body>
    <script language="javascript" type="text/javascript">
        //Browser Support Code
        function ajaxFunction() {
            var ajaxRequest; // The variable that makes Ajax
possible!
            try {
                // Opera 8.0+, Firefox, Safari
                ajaxRequest = new XMLHttpRequest();
            } catch (e) {
                // Internet Explorer Browsers
                try {
                    ajaxRequest = new
ActiveXObject("Msxml2.XMLHTTP");
                } catch (e) {
                    try {
                        ajaxRequest = new
ActiveXObject("Microsoft.XMLHTTP");
                    } catch (e) {
                        // Something went wrong
                        alert("Your browser broke!");
                        return false;
                    }
                }
            }
            // Create a function that will receive data
            // sent from the server and will update
            // div section in the same page.
            ajaxRequest.onreadystatechange = function () {
                if (ajaxRequest.readyState == 4) {
                    var ajaxDisplay =
document.getElementById('ajaxDiv');
                    ajaxDisplay.innerHTML =
ajaxRequest.responseText;
                    console.log(ajaxRequest.responseText);
                }
            }
            // Now get the value from user and pass it to
            // server script.
            var age = document.getElementById('age').value;
```

```

var ensEtud =
document.getElementById('ensEtud').value;
var queryString = "?age=" + age;

queryString += "&ensEtud=" + ensEtud;
console.log(queryString);
ajaxRequest.open("GET", "ajax-example.php" +
queryString, true);
ajaxRequest.send(null);
}
//-->
</script>
```

```

<!-- Chrome, however, blocks this by default. To enable it,
you need to launch Chrome from a command prompt,
specifying the --allow-file-access-from-files flag. -->
<form name='myForm'>
  Max Age: <input type='text' id='age' /> <br />
  <br />

  Enseignant(E) ou Etudiant(T) : <select id='ensEtud'>
    <option value="E">E</option>
    <option value="T">T</option>
  </select>

  <input type='button' onclick='ajaxFunction()' value='Query
MySQL' />

</form>

<div id='ajaxDiv'>Your result will display here</div>
</body>

</html>
```

Soit le programme "ajax-example.php" de la partie Serveur. Analysons le :

```
<?php
```

```
$dbhost = "localhost";
$dbuser = "root";
$dbpass = "";
$dbname = "bduniv";

//Connect to MySQL Server
mysql_connect($dbhost, $dbuser, $dbpass);

//Select Database
mysql_select_db($dbname) or die(mysql_error());

// Retrieve data from Query String
$age = $_GET['age'];
$ensEtud = $_GET['ensEtud'];

// Escape User Input to help prevent SQL Injection
$age = mysql_real_escape_string($age);
$ensEtud = mysql_real_escape_string($ensEtud);

//build query
$query = "SELECT * FROM ajax_example WHERE (enseignantEtudiant =
'$ensEtud')";

if(is_numeric($age))
    $query .= " AND (age >= $age)";

//Execute query
$qry_result = mysql_query($query) or die(mysql_error());

//Build Result String
$display_string = "<table>";
$display_string .= "<tr>";
$display_string .= "<th>Nom</th>";
$display_string .= "<th>Age</th>";
$display_string .= "<th>Enseignant(E)-Etudiant(T)</th>";
$display_string .= "<th>Email</th>";
$display_string .= "</tr>";

// Insert a new row in the table for each person returned
while($row = mysql_fetch_array($qry_result)) {
    $display_string .= "<tr>";
    $display_string .= "<td>$row[nom]</td>";
    $display_string .= "<td>$row[age]</td>";
```

```
$display_string .= "<td>$row[email]</td>";
$display_string .= "<td>$row[enseignantEtudiant]</td>";
$display_string .= "</tr>";
}
echo "Query: " . $query . "<br />";

$display_string .= "</table>";
error_log($display_string, 0);
echo $display_string;
?>
```

3.4 Framework Laravel

3.5 Framework Django

3.6 Exemples d'application

3.6.1 Application 1

3.6.2 Application 2

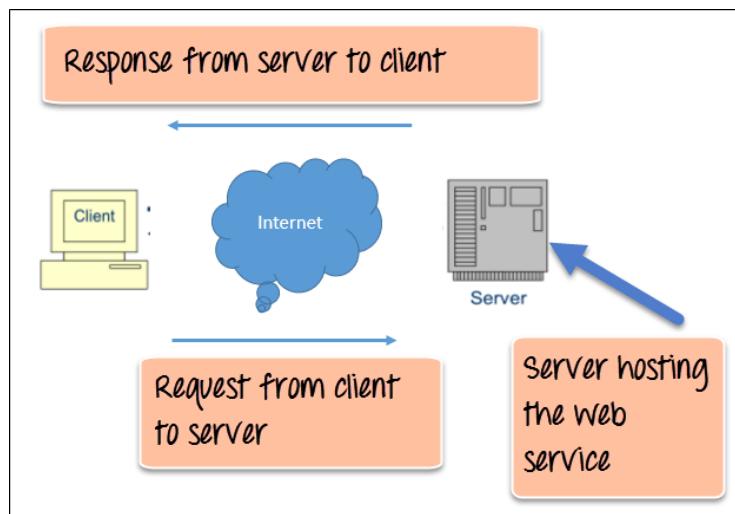
3.6.3 ...

4 Services Web

Références :

- Tutoriel : Les services Web, <http://sdz.tdct.org/sdz/les-services-web.html>
- What are Web Services? Architecture, Types, Example, <https://www.guru99.com/web-service-architecture.html>

Un service web est une application/serveur qui va répondre de manière automatisée à des applications/clients.



La technologie des services Web est un **moyen rapide de distribution de l'information** entre clients, fournisseurs, partenaires commerciaux et leurs différentes plates-formes. Les services Web sont basés sur le modèle **SOA (Service Oriented Application)**.

D'autres technologies telles que **RMI**, **DCOM** et **CORBA** ont précédemment adopté ce style architectural mais ont généralement échoué en raison de la **diversité des plates-formes** utilisées dans les organisations et aussi parce que leur usage n'était **pas adapté à Internet** (problème de passage à travers des **FireWalls**, etc.) d'où la lenteur, voire l'absence de réponses sur ce réseau.

Plusieurs définitions des services Web :

- **W3C** : Un service Web est un composant logiciel identifié par une **URI**, dont les **interfaces publiques sont définies et appelées en XML**. Sa définition peut être découverte par d'autres systèmes logiciels. Les services Web peuvent interagir entre eux d'une manière prescrite par leurs définitions, en utilisant des messages XML portés par les protocoles Internet.

- **Wikipédia :** Un service Web est un programme informatique permettant la communication et l'échange de données entre applications et systèmes hétérogènes dans des environnements distribués. Il s'agit donc d'un ensemble de fonctionnalités exposées sur internet ou sur un intranet, par et pour des applications ou machines, sans intervention humaine, et en temps réel.

En d'autres termes, un service Web est un programme accessible au moyen d'Internet, qui utilise un système de messagerie standard XML, et n'est lié à aucun système d'exploitation ou langage de programmation !

En reprenant la définition du consortium W3C, voici les principaux avantages d'un service Web, à savoir :

- **son interface** décrite d'une manière **interprétable par les machines**, qui permet aux applications clientes d'accéder aux services de manière **automatique** ;
- **son utilisation de langages** et protocoles **indépendants** des plates-formes d'implantation, qui renforcent l'**interopérabilité** entre services ;
- **son utilisation des normes actuelles du Web**, qui permettent la réalisation des interactions faiblement couplées et favorisent aussi l'interopérabilité.

Les services Web fournissent un lien entre applications. Ainsi, des applications **utilisant des technologies différentes peuvent envoyer et recevoir des données au travers de protocoles compréhensibles par tout le monde.**

Un service Web possède les caractéristiques suivantes :

- il est accessible via le réseau ;
- il dispose d'une **interface publique (ensemble d'opérations)** décrite en **XML** ;
- il communique en **utilisant des messages XML**, ces messages sont transportés par des **protocoles Internet** (généralement HTTP, mais rien n'empêche d'utiliser d'autres protocoles de transfert tels : SMTP, FTP, BEEP...) ;
- l'intégration d'application en implémentant des services Web produit des **systèmes faiblement couplés**, le demandeur du service **ne connaît pas forcément le fournisseur**.

Ce dernier peut disparaître sans perturber l'application cliente **qui trouvera un autre fournisseur** en cherchant dans l'annuaire.

Les technologies utilisées par les services Web sont **HTTP, WSDL, REST, XML-RPC, SOAP et UDDI** :

- **REST** : REST (*Representational State Transfer*) est une architecture de services Web. Élaborée en l'an 2000 par *Roy Fielding*, l'un des créateurs du protocole HTTP, du serveur Apache HTTPd et d'autres travaux fondamentaux, REST est une manière de construire une application pour les **systèmes distribués** comme le World Wide Web.

- **XML-RPC** : XML-RPC (remote procedure call) est un protocole simple utilisant XML pour effectuer des **messages RPC**. **Les requêtes sont écrites en XML et envoyées via HTTP POST**. Les requêtes sont intégrées dans le corps de la réponse HTTP. XML-RPC est indépendant de la plate-forme, ce qui lui permet de communiquer avec diverses applications. Par exemple, un client Java peut parler de XML-RPC à un PerlServer !
- **SOAP** : SOAP (*Simple object Access Protocol*) est un **protocole standard de communication**. C'est l'épine dorsale du système d'interopérabilité. SOAP est un protocole décrit en XML et standardisé par le W3C. Il se présente comme une enveloppe pouvant être signée et pouvant contenir des données ou des pièces jointes. **Il circule sur le protocole HTTP et permet d'effectuer des appels de méthodes à distance.**
- **WSDL** : WSDL (*Web Services Description Language*) est un langage de description standard. C'est l'interface présentée aux utilisateurs. **Il indique comment utiliser le service Web et comment interagir avec lui**. WSDL est basé sur XML et permet de décrire de façon précise les détails concernant le service Web tels que les protocoles, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie et les exceptions pouvant être envoyées.
- **UDDI** : UDDI (*Universal Description, Discovery and Integration*) est un **annuaire de services**. Il fournit l'**infrastructure de base pour la publication et la découverte des services Web**. UDDI permet aux fournisseurs de présenter leurs services Web aux clients.

4.1 XML

Références :

- [Tutoriel : Le point sur XML, <http://sdz.tdct.org/sdz/le-point-sur-xml.html>]

XML est un langage de balisage, standardisé par le [World Wide Web Consortium](#) (à l'origine du HTML), qui **définit un ensemble de règles syntaxiques pour la présentation structurée de l'information**.

Un des premiers défis de l'informatique est de faire **communiquer des programmes, de partager des informations**, etc. Puis, il y a le défi d'essayer de faire parler les ordinateurs entre eux ! Internet était né, avec **http** comme protocole pour la distribution des données. Mais avec la naissance d'Internet et des échanges de données à travers la planète, il a fallu inventer **un langage qui serait compris de tous**. On l'appela **SGML**, c'est lui qui donna naissance au **HTML** et **XML**. Un vrai succès !

Malheureusement, la norme **SGML** était trop compliquée pour beaucoup, et c'est alors que fut inventé par le W3C le **XML**. Ce fut un véritable succès. L'industrie s'en empara et inventa un tas de dialectes (ou langages) à partir de XML.

Car c'est bien là l'essence de **XML** : tout le monde peut créer son langage à partir des règles de syntaxes dictées par cette norme. C'est pour ça qu'on dit que

XML est un méta-langage : il permet d'en créer plein d'autres.

Bien sûr, ces langages créés ont tous la même fonction : présenter de l'information structurée. Par exemple, les entreprises aéronautiques ont créé un langage basé sur XML pour l'échange d'informations relatives aux données techniques des avions. De même, il existe plusieurs langages utilisés dans les milieux financiers pour l'échange d'informations relatives aux transactions.

Xml est reconnu pour ce qu'il est : une norme de structuration de données.

XML n'est pas :

- XML n'est pas le remplaçant de HTML, tout simplement parce qu'ils n'ont pas le même but.
- XML ne doit pas être utilisé pour faire du document web.
- XML n'est pas fait pour le stockage.

4.1.1 Les règles de base

XML est un langage de balisage : une balise commence par le signe < et se termine par >.

<balise></balise>

Une balise peut contenir du texte, d'autres balises, les deux ou rien. Par exemple :

<balise><superbalise>Du texte</superbalise> et encore du texte</balise>

Les éléments peuvent porter des attributs, délimités par des " ou des ' :

<mabalise attribut1="Salut" attribut2="tout" attribut3="le" attribut4="monde"/>

Si j'ai un attribut nom dedans, je fais comment ? L'apostrophe va tout faire bugger, non ?

Et bien si ! C'est pour ça qu'il existe avec XML cinq entités prédéfinies :

- < remplace <
- > remplace >
- ' remplace '
- " remplace "
- et & remplace &

Plus de problème :

**<personne nom='O''Neill' métier="secret défense"/>
<personne nom="O'Neill" métier="secret défense"/>
<personne nom="O 'Neill" métier="secret défense"/>**

4.1.2 Le prologue XML

Un prologue peut être placé au tout début du fichier pour indiquer différentes informations (il est optionnel). Ça ressemble à ça :

```
<?xml version="1.0" encoding="iso-8859-1" standalone="yes"?>
```

- On indique d'abord la **version de XML** qu'on utilise. Vous n'utiliserez que la version 1.0 (sauf si vous avez l'envie subite d'écrire vos balises en chinois traditionnel) ;
- la seconde information est **l'encodage du document**. L'encodage par défaut de XML est l'**UTF-8**, mais votre éditeur de texte enregistre probablement vos fichiers en iso-8859-1. Regardez dans ses options pour en être sûrs ;
- la dernière information sert à indiquer si le fichier XML est **susceptible de recevoir une DTD externe** (no) ou non (yes).

4.1.3 Les instructions de traitement

Les instructions de traitement, ou *processing-instruction* en anglais (PI), s'adressent à **l'utilisation d'un langage-cible** qui se sert des informations fournies pour faire une action... quelconque !

La syntaxe est la suivante :

```
<?langage-cible instructions?>
```

Prenons pour exemple la PI *xmlstylesheet* qui permet de lier une feuille de style à un document XML :

```
<?xml-stylesheet type="text/css" href="print.css" title="Imprimer un couleur" media="print" charset="iso-8859-1" alternate="yes"
```

4.1.4 Les sections CDATA

Une section CDATA sert à s'affranchir de l'échappement des caractères spéciaux XML.

Au lieu de :

```
<p>Hier j'ai écrit un fichier XML de ouf !</p>
<p>&lt;monfichierXML&gt;
    &lt;balise1&gt;
        &lt;balise2&gt;Du texte&lt;/balise2&gt;
    &lt;/balise1&gt;
    &lt;balise1 truc="pouf" paf="pif"/&gt;
<&lt;/monfichierXML&gt;</p>
```

On peut écrire :

```
<p>Hier j'ai écrit un fichier XML de ouf !</p>
<p><![CDATA[<monfichierXML>
    <balise1>
        <balise2>Du texte</balise2>
```

```
</balise1>
<balise1 truc="pouf" paf="pif"/>
</monfichierXML>]]></p>
```

On écrit donc du xml sans que le langage cible ne se rende compte que c'est du XML !

4.1.5 Les commentaires

```
<!--le commentaire se place ici-->
```

4.1.6 La galaxie XML

L'intérêt de XML réside dans tous les langages qui gravitent autour, et qui sont intéressants. Ils font partie de la galaxie XML.

Il y a des **langages trop spécialisés**, comme le **CML** par exemple qui est utilisé pour décrire des composés chimiques, ou **SVG** pour le dessin vectoriel en 2D.

Il y a des **langages peu spécialisés** (utiles un peu partout), comme

- (DOM, SAX) pour manipuler XML,
- (Xslt) pour transformer des doc XML,
- (DTD, Schémas XML) pour décrire le contenu d'un doc XML,
- (CSS) pour les mettre en forme,
- (XLink) pour établir des liens entre eux.

4.1.7 La notion de *namespace*

Imaginons que nous voulons générer du SVG (graphisme vectoriel) avec Xslt. Les deux langages ont une balise <text/>. Or, comment le programme qui traitera notre document pourra **savoir que ce <text/> ci est un élément SVG et que celui-là appartient au langage Xslt ?** C'est pour éviter ce genre de problème qu'on appelle collision de nom

Un **namespace** est une chaîne de caractères (une adresse internet, mais elle n'est pas lue) qui sert à **identifier le langage auquel appartient la balise**, ou l'attribut qui fait partie de ce *namespace*.

Tout d'abord, il faut déclarer le **namespace** grâce à l'attribut réservé **xmlns**. Par exemple, pour le XHtml, on fait :

```
<html xmlns="http://www.w3.org/1999/xhtml">
```

Problème : Ajouter un second attribut xmlns est interdit s'il y a deux attributs ayant le même nom !

Solution : **déclarer un préfixe pour un namespace**, qui comme son nom l'indique se placera avant un nom de balise ou d'attribut, et dire que toutes les balises et tous les attributs portant ce préfixe appartiennent au *namespace* précisé, et donc à un langage spécifique.

Pour déclarer un **namespace et un préfixe**, on utilise la syntaxe suivante, toujours avec l'attribut xmlns :

```
xmlns:prefix="adresse-du-namespace"
```

On a une totale liberté sur le choix du préfixe. Ensuite, on l'utilise comme ceci sur les balises et les attributs :

```
<prefix:balise prefix:attribut="truc pif paf pouf"/>
```

Au lieu de <html xmlns="http://www.w3.org/1999/xhtml">, avec du XHtml, cela nous donnera donc par exemple :

```
<mesbalises:html xmlns:mesbalises="http://www.w3.org/1999/xhtml"
mesbalises:lang="en">
<mesbalises:head>
<!-- etc etc -->
</mesbalises:head>
<mesbalises:body>
<mesbalises:p>
<mesbalises:a mesbalises:href="http://www.siteduzero.com">Cours
XHtml/css, php et C++</mesbalises:a>
</mesbalises:p>
</mesbalises:body>
</mesbalises:html>
```

Pour simplifier la chose, on peut enlever les préfixes des attributs : si aucun préfixe n'est associé à un attribut, il appartient au même *namespace* que l'élément auquel il appartient. Ce qui nous donne :

```
<mesbalises:html xmlns:mesbalises="http://www.w3.org/1999/xhtml"
lang="en">
<mesbalises:head>
<!-- etc etc -->
</mesbalises:head>
<mesbalises:body>
<mesbalises:p>
<mesbalises:a href="http://www.siteduzero.com">Cours XHtml/css, php
et C++</mesbalises:a>
</mesbalises:p>
</mesbalises:body>
```

</mesbalises:html>

Maintenant, pour inclure un autre langage, du MathML par exemple, il suffit de déclarer son *namespace* avec un autre préfixe, comme ceci :

```
<xhtml:html xmlns:xhtml="http://www.w3.org/1999/xhtml"
  xmlns:mathml="http://www.w3.org/1998/Math/MathML" lang="en">
  <xhtml:head>
    <!-- etc etc -->
  </xhtml:head>
  <xhtml:body>
    <xhtml:p>
      <mathml:math>
        <mathml:msup>
          <mathml:msqrt>
            <mathml:mrow>
              <mathml:mi>a</mathml:mi>
              <mathml:mo>+</mathml:mo>
              <mathml:mi>b</mathml:mi>
            </mathml:mrow>
          </mathml:msqrt>
          <mathml:mn>27</mathml:mn>
        </mathml:msup>
      </mathml:math>
    </xhtml:p>
  </xhtml:body>
</xhtml:html>
```

4.1.8 Le style avec CSS

CSS (*Cascading Style Sheets*, Feuille de Style en Cascade en français) est un langage non XML (mais très simple), dédié à la présentation de documents XML. CSS couvre un large choix de propriétés, allant de la couleur du texte à la taille d'un bloc, en passant par la position d'une image d'arrière-plan ou des compteurs...

4.1.9 Le style avec Xslt/XPath

La notion de style avec Xslt est très différente de celle avec CSS. Ici, pas question de colorer un bloc en rouge ou d agrandir la taille du texte, même si on parle toujours de feuille de style.

Xslt est un dialecte XML servant à *transformer* un document XML source en un autre document XML ou en texte.

Concrètement, ça marche comme ceci : on a un premier fichier XML source sur lequel on applique une feuille de style Xslt (qui est le second fichier XML), et cela produit un troisième document XML ou texte.

L'intérêt ? Imaginons un document XML qui décrit des statistiques : **on peut grâce à des feuilles de style Xslt le transformer très facilement en un beau graphique 2D SVG, en document XHTML pour la consultation sur le web, ou en pdf en incluant le beau graphique 2D (grâce à FO).**

Pour se situer dans un document XML, **Xslt utilise XPath qui est un langage d'adressage non XML**, lui aussi très simple et très intuitif. Il est indispensable à Xslt, c'est pourquoi on parle généralement du couple Xslt / XPath.

Ces deux langages font partie de la base de XML.

4.1.10XQuery : XML comme une base de données

Il existe un autre langage pour extraire des informations d'un document XML : XQuery. Il adopte une syntaxe très proche de SQL, et utilise XPath pour retirer n'importe quelle donnée d'un fichier XML.

Il utilise aussi Xpath pour se repérer dans le document.

Ce langage permet en plus toutes les opérations que l'on peut vouloir faire avec SQL : insertion de données, mise à jour, suppression, etc.

4.1.11DTDs et schémas XML

Les DTD et les schémas ont les mêmes buts : donner des règles d'écriture d'un document XML spécifique (super important dans l'industrie par exemple). C'est-à-dire qu'en plus de règles de syntaxe de base de XML, **les DTD et les schémas rajoutent des contraintes sur les éléments autorisés, les valeurs possibles d'un attribut, leur ordre d'apparition, etc.**

Par exemple, il est écrit dans la DTD de XHTML qu'un élément `` ne peut contenir que des ``, ou que l'élément `<title>` ne peut se trouver qu'une fois dans `<body>`

Un document qui est correct grâce à ces contraintes est dit valide, par rapport à une DTD ou à un schéma. Un document valide est forcément bien formé, mais un document bien formé n'est pas forcément valide par rapport à une DTD ou à un schéma.

Il est vite apparu que les DTD étaient insuffisantes (ne gèrent pas les *namespace*). C'est pourquoi on a inventé les schémas XML.

4.1.12DOM

DOM (*Document Object Model*) est une API (comprenez : un moyen de programmer) normalisée par le W3C qui construit en mémoire le document XML sous forme arborescente (sous forme d'arbre).

À partir du moment où cet arbre est construit en mémoire, on peut accéder à n'importe quel élément, attribut, et on peut aussi changer leurs valeurs, créer des éléments, des attributs, en cloner d'autres, en supprimer certains, etc. Bref avec DOM, on peut facilement manipuler un document XML, et ce avec différents langages, autant côté client que côté serveur.

4.1.13XML sur le web

4.1.13.1 XHtml

Remplaçant de Html (qui n'est pas un dialecte XML), le rôle de Xhtml est de présenter du document web. Chaque balise a un sens, et la présentation est gérée par CSS.

4.1.13.2 SVG (pour Scalable Vector Graphics)

Un langage puissant : il permet de faire du **dessin vectoriel en 2D** et de faire de l'animation ! du même niveau que Flash, d'autant plus que son implémentation dans les navigateurs s'accélère, même s'il faudra encore un peu de temps avant d'avoir quelque chose de convenable. CSS s'occupe encore et toujours de la présentation.

4.1.13.3 SMIL (pour Synchronized Multimedia Integration Language)

SMIL permet simplement de faire de l'**animation**. SVG utilise SMIL pour le faire.

4.1.13.4 XForms

XForms est la prochaine génération de formulaires web. C'est un langage vraiment très puissant. Il permet entre autres la soumission des données en XML (on peut donc transformer les données reçues avec Xslt, la vérification des données en direct, grâce à CSS et XML Schéma, tout ce que permet XPath pour le calcul des données soumises, de s'affranchir de scripts pour la répétition d'éléments, etc.

XForms est tributaire d'un langage hôte, c'est là que la technologie des namespace prend tout son sens.

4.1.13.5 XLink

XLink met à disposition les six attributs qui permettent de **décrire des liens entre divers documents**, mais en allant beaucoup plus loin que les traditionnels liens HTML tel qu'on les connaît. Les liens sont bidirectionnels, ils peuvent pointer vers plusieurs documents, etc. SVG et Xhtml2, entre autres, utilisent XLink.

4.1.13.6 XPointer

XPointer est en quelque sorte le prolongement de XLink. Il s'appuie sur XPath pour **pointer sur une partie précise d'un document XML**, un peu comme lorsque l'on pointe vers un id avec le # (par exemple mondoc.html#section1), mais en beaucoup plus puissant.

4.1.13.7 XMPP

XMPP signifie *eXtensible Messaging and Presence Protocol*. C'est un protocole de **messagerie instantanée**, normalisé par l'IETF (Internet Engineering Task Force, un organisme de normalisation comme le W3C), et son réseau est appelé Jabber.

Il adopte évidemment une syntaxe XML, ce qui rend le développement d'applications très aisés. Par exemple, ceci m'envoie un message :

```
<message to="Tangui@im.apinc.org" type="chat" id="42"
from="vous@serveur.jabber.org">
<body>Slt, ton tuto sur XML est vraiment génial !</body>
</message>
```

Il est de plus en plus utilisé (même s'il est peu utilisé par rapport aux autres systèmes de messageries instantanées), notamment par les gens qui refusent de voir leur liberté appartenir à des firmes privées, ou par des applications tierces. On peut très bien imaginer mettre à jour un panneau d'affichage numérique en pleine ville via XMPP.

Il utilise à fond le mécanisme des namespace, notamment pour incorporer du XHTML à des messages.

4.1.13.8 SOAP et WSDL

SOAP (Simple Object Access Protocol) et **WSDL (Web Service Description Language)** sont utilisés en commun pour fournir un service web.

Mais qu'est-ce qu'un service web ?

Un service web est une application/serveur qui va répondre de manière automatisée à des applications/clients.

Par exemple, un service web peut fournir la météo à votre barre de tâche sous Ubuntu.

WSDL s'occupe de décrire les informations qu'on peut trouver, et comment on peut les obtenir ("tu peux obtenir l'auteur d'un roman dont tu me fourniras le titre ici en passant par là").

SOAP s'occupe des données transmises : on envoie une enveloppe avec un message décrivant les données que l'on veut, et le service web nous en renvoie une avec les données !

4.2 SOAP

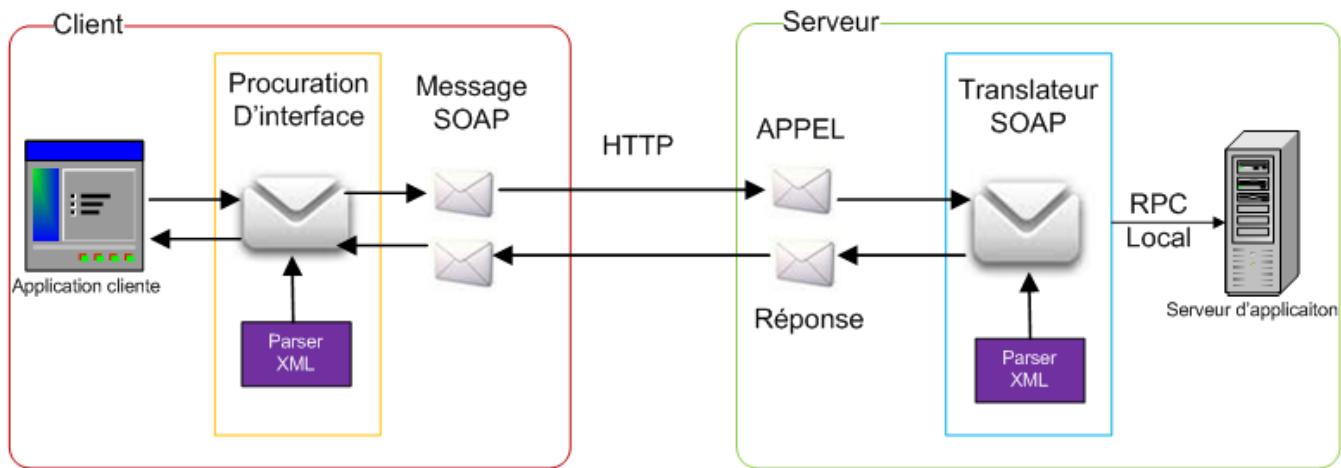
Références :

- <https://www.sitepoint.com/web-services-with-php-and-soap-1/>

En tant que développeurs d'applications, il est nécessaire de pouvoir développer des logiciels et des services pour une large gamme de plates-formes. Toutefois, tout le monde n'utilise pas le même langage ou la même plate-forme, et écrire du code pour les prendre en charge n'est pas réalisable. Si seulement il y avait une norme qui nous permettrait d'écrire du code une fois et de permettre aux autres d'interagir facilement avec à partir de leur propre logiciel ! Heureusement,

il y en a... et son nom est SOAP. (SOAP était un acronyme signifiant « **Simple Object Access Protocol** », mais à partir de la version 1.2, le protocole s'appelle simplement SOAP.)

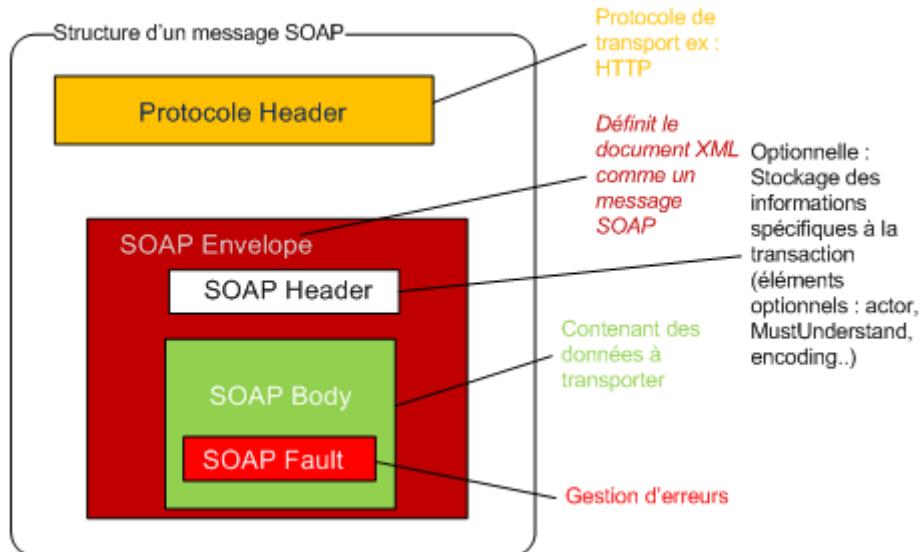
SOAP vous permet de créer **un logiciel interopérable** et permet à d'autres d'exploiter votre logiciel sur un réseau. Il définit les **règles d'envoi et de réception d'appels de procédure distante (RPC)** telles que la structure de la demande et les réponses. Par conséquent, SOAP n'est lié à **aucun système d'exploitation ou langage de programmation** spécifique. Dans la mesure du possible, une personne peut formuler et analyser un message SOAP dans le langage de son choix.



Nous verrons comment créer un serveur et un client SOAP en utilisant la **bibliothèque NuSOAP** pour illustrer le flux de SOAP. Par la suite, nous parlerons de l'importance des fichiers **WSDL**, de la manière de les générer facilement avec NuSOAP et de la manière dont un client peut utiliser un fichier WSDL pour mieux comprendre votre service Web.

4.2.1 La structure d'un message SOAP

SOAP étant basé sur XML, il est censé être compréhensible par un humain, mais il existe un **schéma spécifique qu'il faut respecter**. Commençons par décomposer un message SOAP, en supprimant toutes ses données, et examinons simplement les éléments spécifiques qui constituent un message SOAP.



```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Header>
    ...
    </soap:Header>
    <soap:Body>
    ...
        <soap:Fault>
        ...
            </soap:Fault>
    </soap:Body>
</soap:Envelope>
```

Il ressemble à un fichier XML ordinaire, mais ce qui en fait un message SOAP est l'élément racine **Envelope** avec le nom d'espace de nommage tel que `http://www.w3.org/2001/12/soap-envelope`. L'attribut `soap: encodingStyle` détermine les types de données utilisés dans le fichier, mais SOAP lui-même n'a pas d'encodage par défaut.

`soap:Envelope` est obligatoire, mais l'élément suivant, `soap:Header`, est facultatif et contient généralement des informations relatives à l'authentification et au traitement de la session. Le protocole SOAP n'offre aucune authentification intégrée, mais permet aux développeurs de l'inclure dans cette balise d'en-tête.

Ensuite, il y a l'élément **soap:Body** qui contient le message **RPC** réel, y compris les **noms de méthodes** et, dans le cas d'une réponse, les **valeurs renvoyées par la méthode**. L'élément **soap:Fault** est facultatif; **s'il est présent, il contient tous les messages d'erreur** ou informations d'état pour le message **SOAP** et doit être un élément enfant de **soap:Body**.

Afin de récupérer le plus grand nombre d'erreurs, l'approche SOAP se base essentiellement sur le bon usage de la **balise <soap:fault>** qui est contenue dans le corps SOAP. La balise **<soap:fault>** peut contenir **quatre autres balises facultatives** :

- **faultcode** : cet élément est requis par le cahier des charges. Il contient un **code indiquant la nature du problème**.
- **faultstring** : est la version lisible par l'homme de la balise *faultcode*. C'est la traduction en **langage naturel du code d'erreur**.
- **faultactor** : indique le **service qui a généré l'erreur**. Cela est important lorsqu'une chaîne de services a été utilisée pour traiter la demande.
- **detail** : cet élément doit contenir autant d'informations que possible sur l'état du serveur à l'instant de l'apparition de l'erreur. Il contient souvent des **valeurs de variables au moment de l'échec**.

Quatre types de codes d'erreur sont définis par la spécification :

- **soap:Server** : indique qu'une erreur s'est produite sur le serveur, mais pas avec le message lui-même.
- **soap:Client** : signifie que le message reçu contient une erreur.
- **soap:VersionMismatch** : cette erreur se produit lorsque les **versions** des protocoles SOAP utilisés par le client et le serveur sont **différentes**.
- **soap:MustUnderstand** : cette erreur est générée lorsqu'un élément dans l'en-tête ne peut pas être traité alors qu'il est marqué comme obligatoire.

4.2.1.1 Exemple 1 message SOAP

Maintenant que nous comprenons les bases d'un message SOAP, voyons à quoi ressemblent les messages de requête et de réponse SOAP. Commençons par une demande.

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body xmlns:m="http://www.yourwebroot.com/stock">
        <m:GetStockPrice>
            <m:StockName>IBM</m:StockName>
        </m:GetStockPrice>
    </soap:Body>
</soap:Envelope>
```

Ci-dessus, un exemple de **message de demande SOAP** permettant d'obtenir le cours de l'action d'une société donnée. Dans **soap:Body**, vous remarquerez l'élément **GetStockPrice** spécifique à l'application. **Ce n'est pas un élément SOAP** et prend son nom de la fonction sur le serveur qui sera appelée pour cette requête. **StockName** est également spécifique à l'application et constitue un argument pour la fonction.

Le message de réponse est similaire à la demande :

```
<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
    <soap:Body xmlns:m="http://www.yourwebroot.com/stock">
        <m:GetStockPriceResponse>
            <m:Price>183.08</m:Price>
        </m:GetStockPriceResponse>
    </soap:Body>
</soap:Envelope>
```

L'élément **soap:Body** contient un élément **GetStockPriceResponse** avec un fils **Price** qui contient les données de retour. Comme on peut le voir, **GetStockPriceResponse** et **Price** sont spécifiques à cette application.

Maintenant que vous avez vu un exemple de demande et de réponse et que vous comprenez la structure d'un message SOAP, laissez-nous installer NuSOAP et créez un client et un serveur SOAP pour démontrer la génération de tels messages.

4.2.1.2 Exemple 2 message SOAP

Requête :

```
<!-- Protocole de transport ex. HTTP -->
<?xml version="1.0" encoding="utf-8"?>

<!-- Définit le document XML comme un message SOAP. -->
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
    <!-- Contenant des données à transporter. -->
    <soap:Body>
        <GetQuote xmlns="http://www.webserviceX.NET/">
            <symbol>string</symbol>
        </GetQuote>
    </soap:Body>
</soap:Envelope>
```

Réponse :

```
.....
<StockQuotes>
    <Stock>
        <Symbol>Pegeot</Symbol>
        <Last>2.28</Last>
        <Date>20/11/2009</Date>
        <Time>4:00pm</Time>
        <Change>+0.20</Change>
        <Open>2.20</Open>
        <High>2.30</High>
        <Low>2.07</Low>
        <Volume>124718</Volume>
        <MktCap>18.0M</MktCap>
        <PreviousClose>2.08</PreviousClose>
        <PercentageChange>+9.62%</PercentageChange>
        <AnnRange>1.51 - 3.27</AnnRange>
        <Earns>-0.174</Earns>
        <P-E>N/A</P-E>
        <Name>Forward Industrie</Name>
    </Stock>
</StockQuotes>
.....
```

4.2.2 Construire un serveur SOAP

Il est facile d'installer **NuSOAP** sur votre serveur ; Il suffit de visiter le site sourceforge.net/projects/nusoap, de télécharger et de décompresser le package dans le répertoire racine de votre site Web. Pour utiliser la bibliothèque, inclure simplement le fichier nusoap.php dans votre code.

Pour le serveur, il est question de créer **un service fournissant une liste de produits pour une catégorie de produits**. Le serveur doit lire la catégorie à partir d'une demande, rechercher tous les produits correspondant à la catégorie et renvoyer la liste à l'utilisateur au format CSV.

Créez un fichier dans votre racine Web nommé productlist.php avec le code suivant :

```
<?php
require_once "nusoap.php";

function getProd($category) {
    if ($category == "books") {
        return join(", ", array(
            "The WordPress Anthology",
            "PHP Master: Write Cutting Edge Code",
            "Build Your Own Website the Right Way"));
    }
    else {
        return "No products listed under that category";
    }
}

$server = new soap_server();
$server->register("getProd");
$server->service($HTTP_RAW_POST_DATA);
```

Premièrement, le fichier nusoap.php est inclus pour exploiter la bibliothèque NuSOAP. Ensuite, la fonction getProd() est définie. Ensuite, une nouvelle instance de la classe soap_server est instanciée, la fonction getProd() est enregistrée avec sa méthode register().

C'est vraiment tout ce dont nous avons besoin pour créer notre propre serveur SOAP. Dans un scénario réel, vous recherchez probablement la liste des livres dans une base de données, mais comme nous voulons nous concentrer sur **SOAP**, nous nous sommes simplifiés la tâche de getProd() pour renvoyer une liste de titres codée en dur.

Si nous souhaitons inclure davantage de fonctionnalités dans le serveur, il nous suffit de définir les fonctions supplémentaires (ou même les méthodes dans les classes) et de les enregistrer comme précédemment.

Maintenant que nous avons un serveur opérationnel, construisons un client pour en tirer parti.

4.2.3 Construire un client SOAP

Créez un fichier nommé productlistclient.php et utilisez le code ci-dessous :

```
<?php
require_once "nusoap.php";
$client = new
nusoap_client("http://localhost/nusoap/productlist.php");

$error = $client->getError();
if ($error) {
    echo "<h2>Constructor error</h2><pre>" . $error . "</pre>";
}

$result = $client->call("getProd", array("category" => "books"));

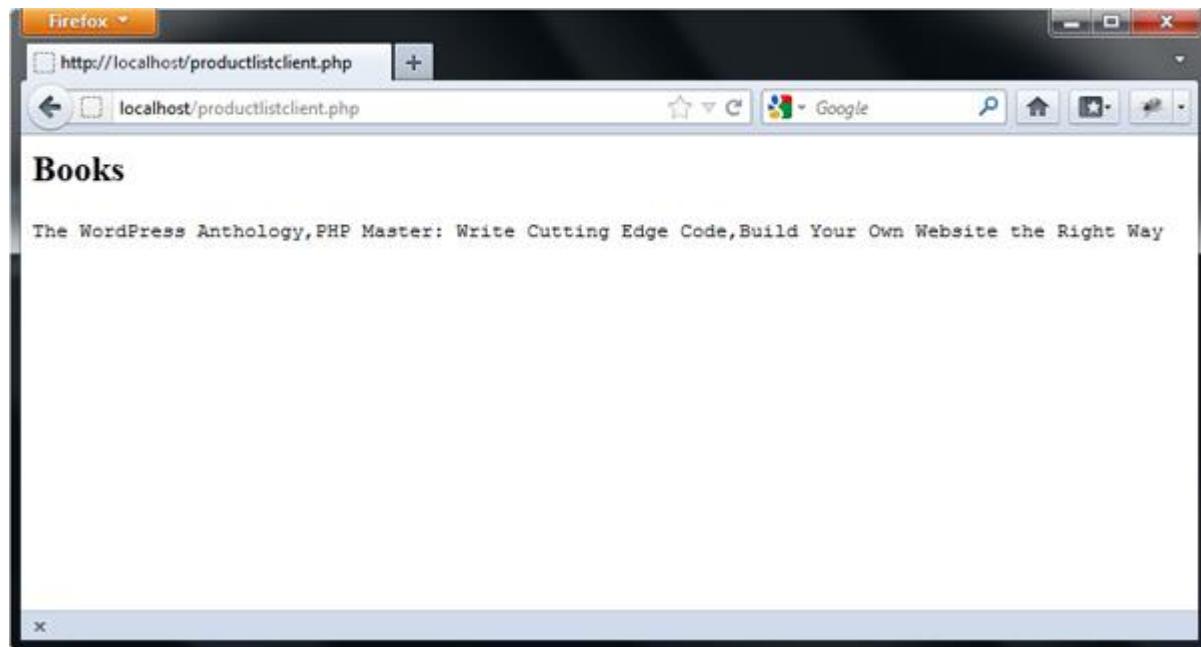
if ($client->fault) {
    echo "<h2>Fault</h2><pre>";
    print_r($result);
    echo "</pre>";
}
else {
    $error = $client->getError();
    if ($error) {
        echo "<h2>Error</h2><pre>" . $error . "</pre>";
    }
    else {
        echo "<h2>Books</h2><pre>";
        echo $result;
        echo "</pre>";
    }
}
```

Nous incluons `nusoap.php` avec `require_once`, puis nous créons une nouvelle instance de `nusoap_client`. Le constructeur prend l'emplacement du serveur SOAP nouvellement créé auquel on va se connecter. La méthode `getError()` vérifie si le client a été créé correctement et le code affiche un message d'erreur si ce n'était pas le cas.

La méthode `call()` génère et envoie la demande SOAP pour appeler la méthode ou la fonction définie par le premier argument. Le second argument de `call()` est un tableau associatif d'arguments pour le RPC. La propriété `fault` et la méthode `getError()` permettent de rechercher

et d'afficher les erreurs éventuelles. S'il n'y a pas d'erreur, le résultat de la fonction est alors généré.

Maintenant, avec les deux fichiers dans votre répertoire racine Web, lancez le script client (dans mon cas, <http://localhost/nusoap/productlistclient.php>) dans votre navigateur. Vous devriez voir :



Si vous souhaitez inspecter les messages de requête et de réponse **SOAP** à des fins de débogage, ou si vous souhaitez les sélectionner séparément, ajoutez ces lignes au bas de `productlistclient.php` :

```
echo "<h2>Request</h2>";
echo "<pre>" . htmlspecialchars($client->request, ENT_QUOTES) .
"</pre>";
echo "<h2>Response</h2>";
echo "<pre>" . htmlspecialchars($client->response, ENT_QUOTES) .
"</pre>";
```

Dans cette première partie, nous avons vu à quel point développer des applications avec le protocole **SOAP** est un moyen efficace pour créer un logiciel interopérable. Nous avons également montré à quel point il est facile de créer notre propre serveur et client **SOAP** à l'aide de la bibliothèque **NuSOAP**. Maintenant, nous voudrions faire quelque chose que nous rencontrerions certainement lorsque nous travaillerons avec des fichiers **SOAP-WSDL**.

Nous décrirons les fichiers WSDL et comment les utiliser. Nous allons voir comment construire rapidement des fichiers **WSDL** avec **NuSOAP** et inclure un fichier WSDL au serveur SOAP et aux exemples de clients de la première partie.

4.2.4 Implémentation de SOAP dans les autres langages

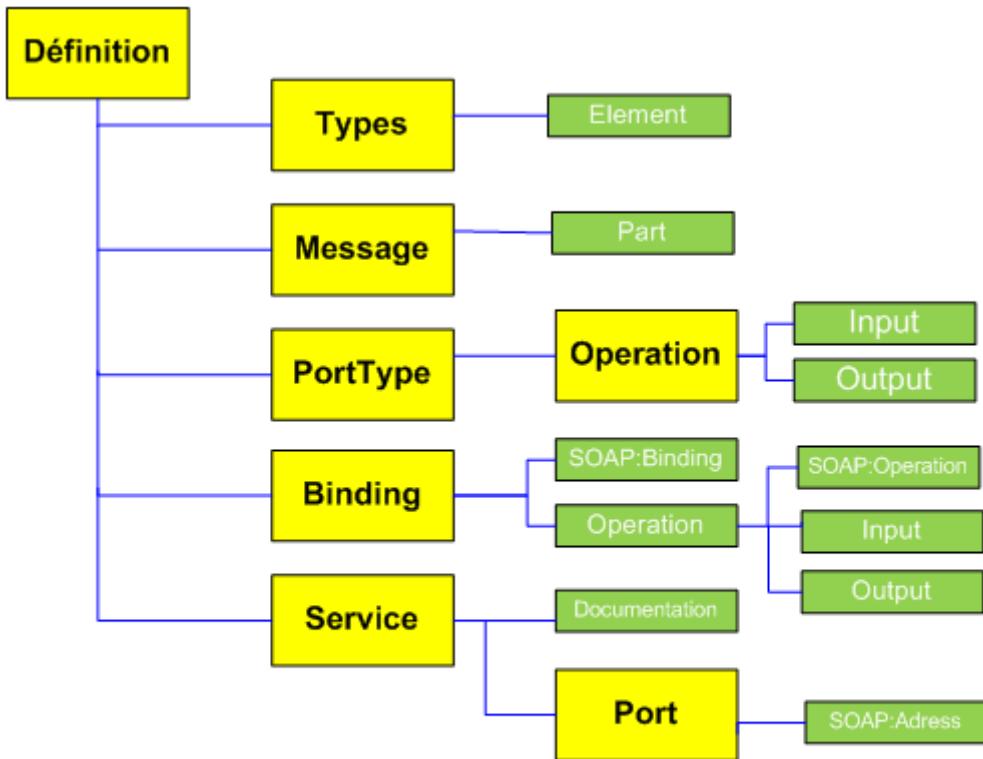
Comme nous l'avons évoqué, les services Web ne se limitent pas à un langage en particulier ou à un système d'exploitation précis, voici quelques langages avec l'implémentation de SOAP :

- JAVA (API et outils associés)
 - JAX-RPC (Java XML ? based RPC) : utilisation de SOAP en mode RPC.
 - JAXR (JA XML Registries) : utilisation de UDDI.
 - JAXM (JA XML Messaging) : utilisation de SOAP en mode message.
- Microsoft (technologie .NET)
 - API dans la bibliothèque de classes .NET.
- Classes PHP SOAP : divers projets *open source*.
- Perl : SOAP::Lite, UDDI::Lite, XMLRPC::Lite.
- etc.

4.2.5 Les fichiers WSDL

Les fichiers **WSDL (Web Services Description Language)** sont des documents XML fournissant des **métadonnées pour un service SOAP**. Ils contiennent des informations sur les **fonctions ou méthodes mises à disposition par l'application et sur les arguments à utiliser**. En mettant les fichiers WSDL à la disposition des utilisateurs de votre service, cela leur donne **les définitions dont ils ont besoin pour envoyer des demandes valides** exactement comme vous le souhaitez. Nous pouvons considérer les **fichiers WSDL comme un contrat complet pour les communications de l'application**. Si nous voulons vraiment rendre facile l'utilisation de notre service par d'autres personnes, nous devons incorporer WSDL à notre programme SOAP.

Tout comme les messages SOAP, les fichiers WSDL ont un schéma spécifique à respecter et des éléments spécifiques doivent être en place pour être valides. Examinons les principaux éléments qui constituent un fichier WSDL valide et expliquons leur utilisation.



Structure d'un document WSDL

Un fichier WSDL contient donc sept éléments.

- **Types** : fournit la définition de types de données utilisés pour décrire les messages échangés.
- **Messages** : représente une définition abstraite (noms et types) des données en cours de transmission.
- **PortTypes** : décrit un ensemble d'opérations. Chaque opération a zéro ou un message en entrée, zéro ou plusieurs messages de sortie ou d'erreurs.
- **Binding** : spécifie une liaison entre un `<portType>` et un protocole concret (SOAP, HTTP...).
- **Service** : indique les adresses de port de chaque liaison.
- **Port** : représente un point d'accès de services défini par une adresse réseau et une liaison.
- **Opération** : c'est la description d'une action exposée dans le port.

Concrètement :

```
<definitions>
  <types>
  .....
  </types>
  <message>
    <part></part>
  </message>
  <portType>
  .....
  </portType>
  <binding>
  .....
  </binding>
  <service>
  .....
  </service>
</definitions>
```

L'élément racine du fichier WSDL est l'élément **definitions**. Un fichier WSDL est par définition une définition d'un service Web.

- L'élément **types** décrit le **type de données utilisé**, qui, dans le cas de WSDL, utilise un schéma XML.
- L'élément **messages** contient la définition des **éléments de données du service**. Chaque élément de message peut contenir un ou plusieurs.
- L'élément **portType** définit les **opérations pouvant être effectuées avec votre service Web** et les messages de réponse à la demande utilisés.
- L'élément **binding** contient la spécification du **protocole et de format de données pour un type de port** particulier.
- Enfin, nous avons l'élément **service** qui définit une **collection d'éléments de service contenant l'URI** (emplacement) du service.

La terminologie a légèrement changé en nommant certains des éléments de la spécification WSDL 2.0. **portType**, par exemple, a changé de nom et s'appelle désormais **Interface**. Le support de WSDL 2.0 étant actuellement faible, nous avons choisi de rester à la version 1.1, qui est plus largement utilisée.

4.2.6 Construire un fichier WSDL

Les fichiers WSDL peuvent être fastidieux à écrire à la main car ils doivent contenir des balises spécifiques et sont généralement assez longs. L'avantage d'utiliser NuSOAP est qu'il peut créer un fichier WSDL pour vous ! Nous allons modifier le serveur SOAP que nous avons créé dans le premier article en conséquence.

Ouvrons `productlist.php` et changeons-le pour adopter le code ci-dessous :

```
<?php
require_once "nusoap.php";

function getProd($category) {
    if ($category == "books") {
        return join(", ", array(
            "The WordPress Anthology",
            "PHP Master: Write Cutting Edge Code",
            "Build Your Own Website the Right Way"));
    }
    else {
        return "No products listed under that category";
    }
}

$server = new soap_server();
$server->configureWSDL("productlist", "urn:productlist");

$server->register("getProd",
    array("category" => "xsd:string"),
    array("return" => "xsd:string"),
    "urn:productlist",
    "urn:productlist#getProd",
    "rpc",
    "encoded",
    "Get a listing of products by category");

$server->service($HTTP_RAW_POST_DATA);
```

Fondamentalement, il s'agit du même code que précédemment, mais avec seulement quelques modifications. La première modification ajoute un appel à `configureWSDL()`; la méthode **indique au serveur de générer un fichier WSDL pour notre service**. Le premier argument est le nom du service et le second est l'espace de nom de notre service. Une discussion sur les espaces de noms sort vraiment du cadre de cet article, mais sachez que, même si nous n'en tirons pas parti ici, des plates-formes comme Apache Axis et .NET le font. Il est préférable de les inclure pour être totalement interopérables.

La deuxième modification ajoute des arguments supplémentaires à la méthode `register()`. Le décomposer :

- getProd est le nom de la fonction
- array ("category" => "xsd: string") définit l'argument d'entrée de getProd et son type de données
- array ("return" => "xsd: string") définit la valeur de retour de la fonction et son type de données.
- urn:productlist définit le namespace
- urn:productlist#getProd définit l'action SOAP
- rpc définit le type d'appel (rpc ou document).
- encoded définit la valeur de l'attribut use (encoded ou literal peut être utilisé)
- Le dernier paramètre est une chaîne de documentation décrivant le rôle de la fonction getProd.

Maintenant, pointez votre navigateur sur

`http://votrewebroot/productlist.php?wsdl`

et vous verrez le nouveau fichier WSDL créé. Continuez, **copiez cette source, enregistrez-la dans son propre fichier nommé products.wsdl et placez-la dans votre répertoire Web.**

4.2.7 Utilisation de fichiers WSDL avec le client

Nous avons modifié le serveur SOAP pour générer un fichier WSDL. Modifions maintenant le client SOAP pour le consommer. Ouvrons productlistclient.php créé dans l'article précédent et modifions simplement la ligne qui initialise le client :

```
$client = new  
nusoap_client("http://localhost/nusoap/productlist.php");
```

vers

```
$client = new nusoap_client("products.wsdl", true);
```

Le deuxième paramètre de l'appel du constructeur `nusoap_client()` indique à NuSOAP de créer un **client SOAP pour accepter le fichier WSDL**. Lançons maintenant productlistclient.php dans notre navigateur et nous devrions voir le même résultat qu'auparavant, mais nous utilisons maintenant WSDL !

4.3 L'annuaire des services UDDI

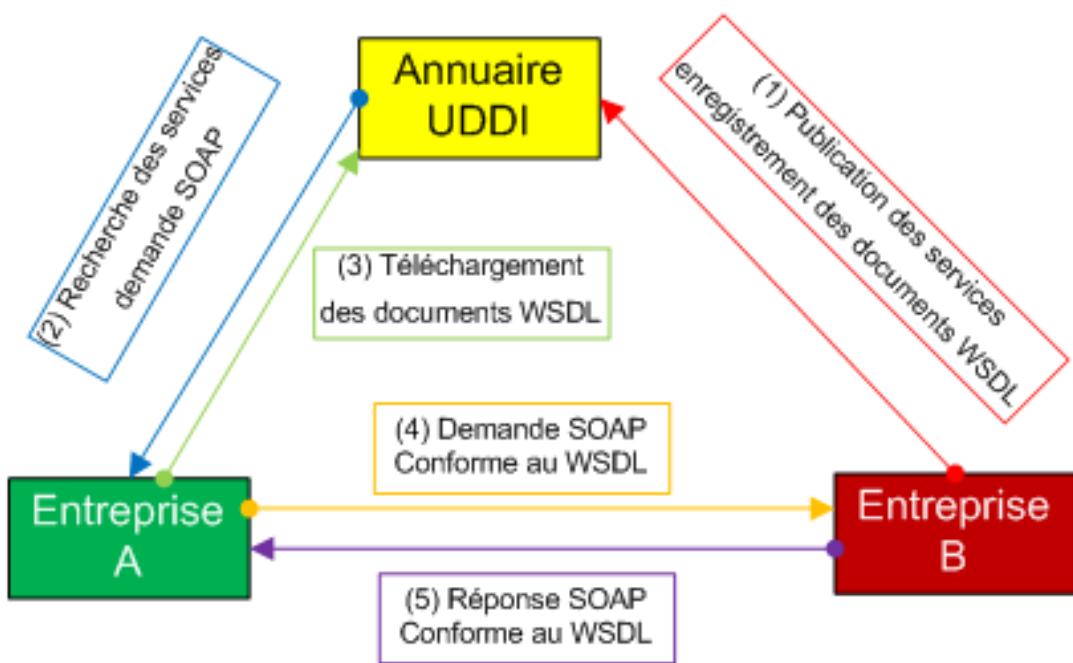


Schéma Général de l'annuaire UDDI

4.4 Application 1 : service web *restful*

Référence : <https://codinginfinite.com/restful-web-services-php-example-php-mysql-source-code/>

Nous utilisons l'organisation suivante du programme :

```
api
  config/
    database.php – file used for connecting to the database.
  objects/
    user.php – contains properties and methods for “user” database queries.
  User/
    signup.php – file that will accept user data to be saved to the DB.
    login.php – file that will accept username & password and validate
```

4.4.1 Création de la base de données et de la classe user

Création de la table :

```
CREATE TABLE `users` (
  `id` int(11) NOT NULL PRIMARY KEY AUTO_INCREMENT,
  `username` varchar(255) NOT NULL,
  `password` varchar(255) NOT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
```

Connexion à la base de données :

```
<?php
class Database{

    // specify your own database credentials
    private $host = "localhost";
    private $db_name = "PHPLearning";
    private $username = "root";
    private $password = "";
    public $conn;

    // get the database connection
    public function getConnection(){

        $this->conn = null;

        try{
            $this->conn = new PDO("mysql:host=" . $this->host .
"dbname=" . $this->db_name, $this->username, $this->password);
            $this->conn->exec("set names utf8");
        }catch(PDOException $exception){
            echo "Connection error: " . $exception->getMessage();
        }

        return $this->conn;
    }
}
?>
```

Class utilisateur et méthodes d'inscription et de login :

```
<?php
class User{

    // database connection and table name
    private $conn;
    private $table_name = "users";

    // object properties
    public $id;
    public $username;
    public $password;
    public $created;

    // constructor with $db as database connection
    public function __construct($db){
        $this->conn = $db;
    }

    // signup user
    function signup(){

    }

    // login user
    function login(){

    }

    // a function to check if username already exists
    function isAlreadyExist(){

    }
}
```

Pour l'inscription :

```
// signup user
function signup(){
    if($this->isAlreadyExist()){
        return false;
    }
    // query to insert record
$query = "INSERT INTO
            " . $this->table_name . "
            SET username=:username, password=:password,
created=:created";
    // prepare query
$stmt = $this->conn->prepare($query);
    // sanitize
$this->username=htmlspecialchars(strip_tags($this-
>username));
    $this->password=htmlspecialchars(strip_tags($this-
>password));
    $this->created=htmlspecialchars(strip_tags($this->created));
    // bind values
$stmt->bindParam(":username", $this->username);
$stmt->bindParam(":password", $this->password);
$stmt->bindParam(":created", $this->created);
    // execute query
if($stmt->execute()){
    $this->id = $this->conn->lastInsertId();
    return true;
}
return false;

}
```

Vérification que l'utilisateur existe :

```
// a function to check if username already exists
function isAlreadyExist(){

    $query = "SELECT *
              FROM
                  " . $this->table_name . "
              WHERE
                  username='". $this->username . "'";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // execute query
    $stmt->execute();

    if($stmt->rowCount() > 0){
        return true;
    }
    else{
        return false;
    }
}
```

La méthode login :

```
// login user
function login(){
    // select all query
    $query = "SELECT
                  `id`, `username`, `password`, `created`
              FROM
                  " . $this->table_name . "
              WHERE
                  username='". $this->username . "' AND
password='". $this->password . "'";

    // prepare query statement
    $stmt = $this->conn->prepare($query);

    // execute query
    $stmt->execute();
    return $stmt;
}
```

4.4.2 Les services web d'inscription et de login

Dans le répertoire api, on crée l'API service d'inscription ;

```
<?php

// get database connection
include_once '../config/database.php';

// instantiate user object
include_once '../objects/user.php';

$database = new Database();
$db = $database->getConnection();

$user = new User($db);

// set user property values
$user->username = $_POST['username'];
$user->password = $_POST['password'];
$user->created = date('Y-m-d H:i:s');

// create the user
if($user->signup()){
    $user_arr=array(
        "status" => true,
        "message" => "Successfully Signup!",
        "id" => $user->id,
        "username" => $user->username
    );
}
else{
    $user_arr=array(
        "status" => false,
        "message" => "Username already exists!"
    );
}
print_r(json_encode($user_arr));
?>
```

Puis l'API login :

```
<?php
// include database and object files
include_once '../config/database.php';
include_once '../objects/user.php';
// get database connection
$database = new Database();
$db = $database->getConnection();
// prepare user object
$user = new User($db);

// set ID property of user to be edited
$user->username = isset($_GET['username']) ? $_GET['username'] :
die();
$user->password = isset($_GET['password']) ? $_GET['password'] :
die();

// read the details of user to be edited
$stmt = $user->login();

if($stmt->rowCount() > 0){
    // get retrieved row
    $row = $stmt->fetch(PDO::FETCH_ASSOC);
    // create array
    $user_arr=array(
        "status" => true,
        "message" => "Successfully Login!",
        "id" => $row['id'],
        "username" => $row['username']
    );
}
else{
    $user_arr=array(
        "status" => false,
        "message" => "Invalid Username or Password!",
    );
}
// make it json format
print_r(json_encode($user_arr));
?>
```

Maintenant, vous pourrez tester l'API d'inscription via :

<http://localhost/api/users/signup.php>

avec les paramètres **Post** de username & password

et l'API login via :

<http://localhost/api/users/login.php?username=shehryar&password=12345>

4.5 Application 2 : service web SOAP (WSDL généré)

Réaliser les étapes de création d'un service web avec SOAP décrites dans la section 4.2.

4.6 Application 3 : service web SOAP (WSDL disponible)

Références :

- <https://gist.github.com/umidjons/f3de2533c51495a9c557>

4.6.1 Côté serveur

```
<?php
// turn off WSDL caching
ini_set("soap.wsdl_cache_enabled","0");

// model, which uses in web service functions as parameter
class Book
{
    public $name;
    public $year;
}

/**
 * Determines published year of the book by name.
 * @param Book $book book instance with name set.
 * @return int published year of the book or 0 if not found.
 */
function bookYear($book)
{
    // list of the books
    $_books=[
        ['name'=>'test 1','year'=>2011],
        ['name'=>'test 2','year'=>2012],
        ['name'=>'test 3','year'=>2013],
    ];
    // search book by name
    foreach($_books as $bk)
        if($bk['name']==$book->name)
            return $bk['year']; // book found
```

```

        return 0; // book not found
}

// initialize SOAP Server
$server=new SoapServer("test.wsdl",
    'classmap'=>[
        'book'=>'Book', // 'book' complex type in WSDL file mapped to
the Book PHP class
    ]
);

// register available functions
$server->addFunction('bookYear');

// start handling requests
$server->handle();

```

4.6.2 Côté client

```

<?php
// model
class Book
{
    public $name;
    public $year;
}

// create instance and set a book name
$book      =new Book();
$book->name='test 2';

// initialize SOAP client and call web service function
$client=new
SoapClient('http://a.uz/soap/server.php?wsdl',[ 'trace'=>1, 'cache_wsdl'
'=>WSDL_CACHE_NONE]);
$resp   =$client->bookYear($book);

// dump response
var_dump($resp);

```

4.6.3 Fichier WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<wsdl:definitions name="Library"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="Library"
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
    xmlns:tns="Library"
    xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

    <xsd:documentation></xsd:documentation>

    <wsdl:types>
        <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
            targetNamespace="Library">
            <xsd:complexType name="book">
                <xsd:sequence>
                    <xsd:element name="name"
                        type="xsd:string"></xsd:element>
                    <xsd:element name="year"
                        type="tns:integer"></xsd:element>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:schema>
    </wsdl:types>

    <wsdl:message name="bookYearRequest">
        <wsdl:part name="book" type="xsd:book"></wsdl:part>
    </wsdl:message>
    <wsdl:message name="bookYearResponse">
        <wsdl:part name="year" type="tns:integer"></wsdl:part>
    </wsdl:message>

    <wsdl:portType name="Library">
        <wsdl:operation name="bookYear">
            <wsdl:input message="tns:bookYearRequest"/>
            <wsdl:output message="tns:bookYearResponse"/>
        </wsdl:operation>
    </wsdl:portType>

    <wsdl:binding name="Library" type="tns:Library">
        <soap:binding style="rpc"
            transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="bookYear">
```

```
<soap:operation  
soapAction="http://a.uz/soap/server.php"/>  
    <wsdl:input>  
        <soap:body use="literal" namespace="Library"/>  
    </wsdl:input>  
    <wsdl:output>  
        <soap:body use="literal" namespace="Library"/>  
    </wsdl:output>  
  </wsdl:operation>  
</wsdl:binding>  
  
<wsdl:service name="Library">  
  <wsdl:port binding="tns:Library" name="BookLibrary">  
    <soap:address location="http://a.uz/soap/server.php"/>  
  </wsdl:port>  
</wsdl:service>  
  
</wsdl:definitions>
```

4.7 Platform .NET

4.8 Platform Java

5 Sécurité

Références :

- Tutoriel : Sécurité PHP, sécuriser les flux de données
<http://sdz.tdct.org/sdz/securite-php-securiser-les-flux-de-donnees.html>

Dans un site web, tout est axé autour du **traitement des données**. Que ce soit des données issues d'une base de données ou les données soumises par les internautes eux-même par l'intermédiaire de formulaires. **Bien protéger ses données c'est assurer une certaine sécurité globale.**

Une architecture *3-tier* et plus généralement *n-tier* est un système composé de n couches. Dans le cas du Web, on parle souvent d'architecture *3-tier* ou *3 étages*.



Les flèches bleues représentent les flux de données qui transitent depuis l'internaute jusqu'à la base de données et les flèches vertes les données transitant depuis la base de données jusqu'au navigateur de l'internaute. Ces transferts de données sont schématisés par les flèches rouges.

Sécurisons nos données :

- **Les données entrantes** : dans un premier temps, nous allons devoir nous occuper de sécuriser les données qui proviennent du membre et donc de formulaires avant de pouvoir les entrer dans la BDD.
- **Les données sortantes** : puis, nous verrons comment sécuriser l'affichage des données issues de la BDD.

5.1 Protéger les données entrant dans la BDD

L'attaque par **injection SQL** est très fréquente car elle est **rapide à mettre en place**, peut occasionner des **dégâts irréversibles** dans votre base de données ou, si elle est utilisée de manière plus subtile, elle **permet de récupérer en toute discréption les mots de passe** et identifiants. Le **pirate détourne votre requête en injectant du code** dans les champs du formulaire : d'où le terme d'injection SQL.

5.1.1 Détournement de clause WHERE

On a l'habitude de formater nos requêtes SQL de cette manière :

```
<?php
    $requete = "SELECT * FROM membre
    WHERE pseudo = '". $_GET['pseudo']."'"
    AND password = '". $_GET['password']."' ";
?
>
```

Nous avons utilisé `$_GET` mais que l'injection SQL fonctionne tout aussi bien avec `$_POST` : ne vous croyez donc pas à l'abri en mettant `method = "post"` en attribut de vos balises de formulaire. C'est souvent une idée préconçue mais qui s'avère totalement fausse car il est fort possible de modifier les en-têtes HTTP pour transmettre des données sous forme `$_POST`, ... !

Imaginons un instant que je transmette ceci :

```
<?php
    $_GET['password'] = " ' OR 1 = '1 ";
?
>
```

Notre requête devient :

```
<?php
    $requete = "SELECT * FROM membre
    WHERE pseudo = '". $_GET['pseudo']."'"
    AND password = '' OR 1 = '1' ";
?
>
```

Ainsi, vous pouvez entrer n'importe quel mot de passe, il sera toujours valide puisque vous admettrez que 1 est toujours égal à 1. Il suffit donc de fournir un pseudo valide et la requête est vraie !

De cette manière, il est possible de récupérer facilement les informations sur les membres du site.

5.1.2 Détournement de la clause DELETE

Vous commencez à voir le danger ? C'est normal : il est évident de comprendre tout à coup que tout mon site web était rempli de failles avant que j'applique les quelques règles de sécurité qu'on va voir. Mais attendez de voir ce que nous réserve le détournement d'une clause DELETE : c'est peut-être ce qu'il y a de pire !

Prenons un exemple de requête :

```
<?php
    $requete = "DELETE FROM membre
    WHERE id = '" . $_GET['id'] . "' ";
?>
```

Voilà ce qu'on peut transmettre en paramètre :

```
<?php
    $_GET['id'] = "1 ' OR id > '0   ";
?>
```

La requête donne donc :

```
<?php
    $requete = "DELETE FROM membre
    WHERE id = '1' OR id > '0' ";
?>
```

Cette requête détruira donc toutes les entrées contenues dans votre table membre ! Vous aurez perdu tous vos membres en 10 ms...

5.1.3 Les magic quotes

Il nous faut donc protéger les données issues du formulaire avant de les entrer dans notre requête sinon on s'expose à de gros risques...

Nous devons échapper nos valeurs pour les rendre inertes et sans danger. Les magic_quotes font partie d'une directive de PHP visant à assurer la sécurité des requêtes SQL à son insu en **échappant systématiquement les caractères suivants** :

- les guillemets simples ' ;
- les guillemets doubles " ;
- les slashes / ;
- les caractères ***NULL***.

Pour échapper un caractère, cette directive ajoute des antislashes dans les chaînes qui transitent vers le script PHP.

En fait, elle joue le même rôle que la fonction *addslashes()*.

Regardons ce que donne l'activation de cette directive ensemble :

```
<?php
    $requete = "SELECT pseudo FROM membre
    WHERE id = '". $_GET['id'] . "' ";
?>
```

L'ajout du caractère d'échappement permet d'utiliser certains caractères dans notre requête sans qu'elle ne se transforme pour autant en injection.

```
<?php
    $requete = "SELECT pseudo FROM membre
    WHERE id = '\\"1\"mettez l'injection que vous voulez ici\\' ";
?>
```

Cette directive a donc la capacité de bloquer certaines injections SQL.

5.1.4 Sécuriser

Nous allons maintenant parler d'une fonction bien pratique : *mysql_real_escape_string()*.

Prototype :

```
string mysql_real_escape_string (
    string $unescape_string [, resource $link_identifier ] )
```

Cette fonction échappe les caractères suivants :

- les guillemets simples ' ;
- les guillemets doubles " ;
- les slashes / ;
- les caractères *NULL* ;
- les caractères suivants : , , \x00 et \x1a.

En gros, elle neutralise tous les caractères susceptibles d'être à l'origine d'une injection SQL. À partir de maintenant, utilisez toujours cette fonction pour sécuriser les chaînes transmises à vos requêtes. Par ailleurs, il faut aussi que vos données soient placées dans des guillemets (simples ou doubles) sinon, il est possible de réinjecter du code (notamment des sous-requêtes grâce aux parenthèses).

Exemple :

```
<?php
$pseudo = mysql_real_escape_string($_POST['pseudo']);
$requete = "SELECT * FROM membre WHERE pseudo = '$pseudo' ";
?>
```

Contrairement à ce qu'on pourrait penser, il est toujours possible de réaliser des **injections SQL** particulières qui visent notamment à surcharger votre serveur en alourdisant votre requête. Ce type d'injection utilise les caractères % et _.

Le caractère % est souvent utilisé dans MySQL avec la clause **LIKE**. Ce caractère est un joker qui représente n'importe quelle autre chaîne.

```
SELECT id, pseudo, password FROM membre WHERE pseudo LIKE 'a%'
```

5.1.5 addcslashes()

Prototype :

```
string addcslashes ( string $str , string $charlist )
```

La chaîne à échapper est le premier paramètre à passer à la fonction. Puis, on entre les caractères qui doivent être échappés. Elle s'utilise comme cela :

```
<?php
$chaine = "%salut_";
// Échappement des caractères % et _
$chaine = addcslashes($chaine, '%_');
echo $chaine; // Affiche \%salut\_
?>
```

Voici une fonction qui sécurise vos données avant de les passer dans votre requête. Centraliser le traitement est une bonne habitude à acquérir dès le début.

```
<?php
    function securite_bdd($string)
    {
        // On regarde si le type de string est un nombre entier (int)
        if(ctype_digit($string))
        {
            $string = intval($string);
        }
        // Pour tous les autres types
        else
        {
            $string = mysql_real_escape_string($string);
            $string = addslashes($string, '%_');
        }

        return $string;
    }
?>
```

Maintenant, je veux et j'exige qu'à chaque fois que vous récupérez une donnée issue d'un formulaire ou d'une URL, vous lui appliquez cette fonction avant de faire quoi que ce soit. Exemple :

```
<?php
    // Sécurisation des variables
    $pseudo = securite_bdd($_GET['pseudo']);
    $password = securite_bdd($_GET['password']);

    // Formatage de la requête
    $requete = "SELECT * FROM membre
    WHERE pseudo = '$pseudo'
    AND password = '$password' ";

    // Envoi au serveur en toute sécurité ^^
    mysql_query($requete) or exit(mysql_error());
?>
```


6 Fiches TP

6.1 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.

6.1.1 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.

6.1.2 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 c'est moins joli.

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>.

6.1.3 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.
```

6.1.4 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>
<html>
  <body>
    </body>
</html>
</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 `>` ?

6.1.5 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 -).

6.1.6 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>

```

6.1.7 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.

6.1.8 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> 17<sup>ème</sup>
</font> <td> Rembrandt <td> Molière <td> Bach
  <tr valign=middle align=left> <th><font size=6> 19<sup>ème</sup>
</font> <td> Monet <td> Goethe <td> Brahms
  <tr valign=bottom align=right> <th><font size=6> 20<sup>ème</sup>
</font> <td> Modigliani <td> Jarry <td> Ellington
</table>
```

<hr>

On peut faire l'alignement, case par case,


```
<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> 17<sup>ème</sup> </font> <td valign=top
align=center> Rembrandt <td> Molière <td> Bach
  <tr > <th><font size=6> 19<sup>ème</sup> </font> <td> Monet <td
valign=middle align=right > Goethe <td> Brahms
  <tr > <th><font size=6> 20<sup>ème</sup> </font> <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>
```

6.1.9 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 :

```
<a href="nouveau.html">nouvelle page</a>.
```

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 <a href="https://www.univ-oran1.dz/index.php/en/nos-  
services/uci">lien  
déjà visité</a>.<br>  
Un <a href="http://www.cerist.dz/index.php/fr/portails/1111-  
pnci">lien</a>  
pas encore visité.<br>  
Pour <a href="mailto:ucioran@gmail.com">m'écrire</a>.  
Pour ouvrir <a href="" target="_blank">cette même page dans une  
autre fenêtre</a>.
```

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>`.

6.1.10 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.

6.1.11 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:15%; padding:1em;">  
    Cette balise div a du style !  
</div>
```

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

6.1.12 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.

6.1.13 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.

6.1.14 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>
```

6.2 Fiche TP n°2 : Prise en main JavaScript

6.2.1 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.

6.2.2 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().

6.2.3 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 »
```

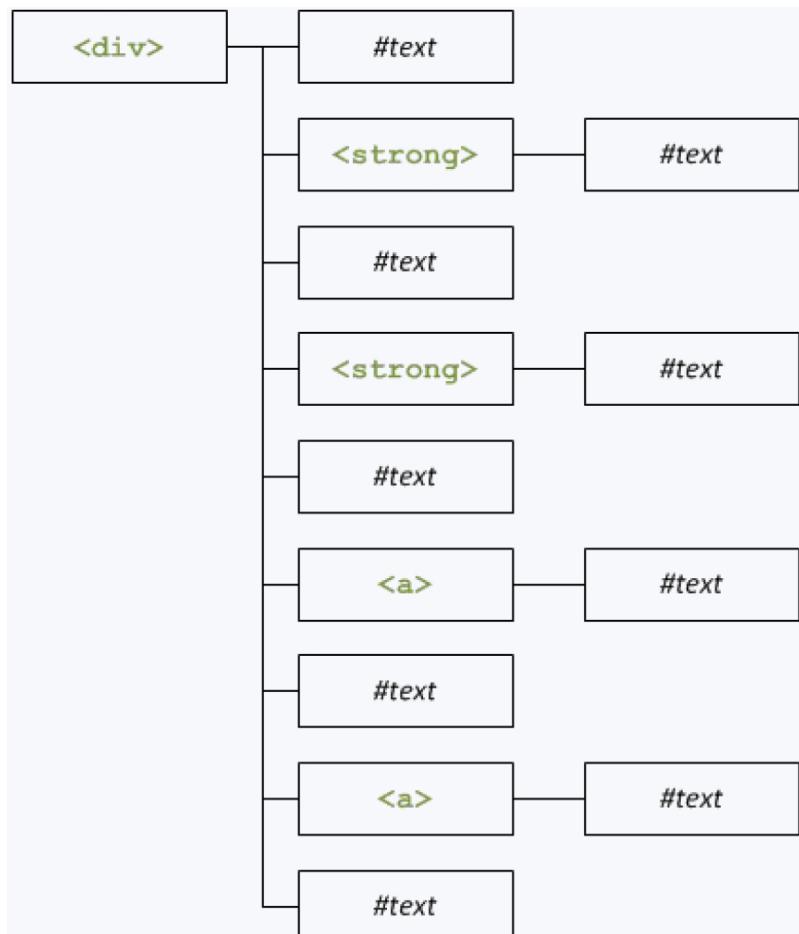
Vous voilà maintenant prêts à vous lancer dans l'écriture de votre code. Nous précisons de nouveau que les nombres à convertir vont de 0 à 999, mais rien ne vous empêche de faire plus si le cœur vous en dit. Évitez de faire moins, vous manqueriez une belle occasion de vous entraîner correctement.

6.2.4 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.

Code : HTML

```
<div id="divTP1">
    Le <strong>World Wide Web Consortium</strong>, abrégé par le sigle
    <strong>W3C</strong>, est un
        <a href="http://fr.wikipedia.org/wiki/Organisme_de_normalisation"
            title="Organisme de normalisation">organisme de standardisation</a>
            à but non-lucratif chargé de promouvoir la compatibilité des
            technologies du <a
                href="http://fr.wikipedia.org/wiki/World_Wide_Web" title="World Wide
                Web">World Wide Web</a>.
    </div>
```



Voici le code JavaScript qui le fait :

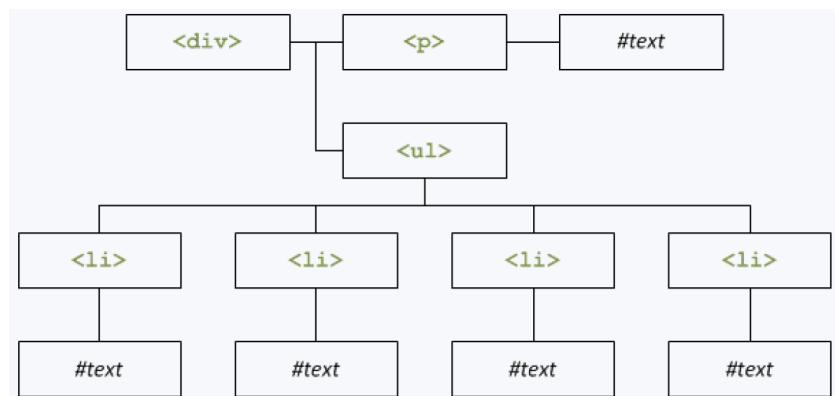
```
var mainDiv = document.createElement('div');
    mainDiv.id = 'divTP1';
// On crée tous les nœuds textuels, pour plus de facilité
var textNodes = [
    document.createTextNode('Le '),
    document.createTextNode('World Wide Web Consortium'),
    document.createTextNode(' ', abrégé par le sigle '),
    document.createTextNode('W3C'),
    document.createTextNode(' ', est un '),
    document.createTextNode('organisme de standardisation'),
    document.createTextNode(' à but non-lucratif chargé de
promouvoir la compatibilité des technologies du '),
    document.createTextNode('World Wide Web'),
    document.createTextNode('.')

];
// On crée les deux <strong> et les deux <a>
var w3cStrong1 = document.createElement('strong');
var w3cStrong2 = document.createElement('strong');
w3cStrong1.appendChild(textNodes[1]);
w3cStrong2.appendChild(textNodes[3]);
var orgLink = document.createElement('a');
var wwwLink = document.createElement('a');
orgLink.href =
'http://fr.wikipedia.org/wiki/Organisme_de_normalisation';
orgLink.title = 'Organisme de normalisation';
orgLink.appendChild(textNodes[5]);
wwwLink.href = 'http://fr.wikipedia.org/wiki/World_Wide_Web';
wwwLink.title = 'World Wide Web';
wwwLink.appendChild(textNodes[7]);
// On insère le tout dans mainDiv
mainDiv.appendChild(textNodes[0]);
mainDiv.appendChild(w3cStrong1);
mainDiv.appendChild(textNodes[2]);
mainDiv.appendChild(w3cStrong2);
mainDiv.appendChild(textNodes[4]);
mainDiv.appendChild(orgLink);
mainDiv.appendChild(textNodes[6]);
mainDiv.appendChild(wwwLink);
mainDiv.appendChild(textNodes[8]);
// On insère mainDiv dans le <body>
document.body.appendChild(mainDiv);
```

Commentez !

De même avec le code :

```
<div id="divTP2">
  <p>Langages basés sur ECMAScript :</p>
  <ul>
    <li>JavaScript</li>
    <li>JScript</li>
    <li>ActionScript</li>
    <li>EX4</li>
  </ul>
</div>
```



On y arrive avec :

```
// On crée l'élément conteneur
var mainDiv = document.createElement('div');
  mainDiv.id = 'divTP2';
// On crée tous les nœuds textuels, pour plus de facilité
var languages = [
  document.createTextNode('JavaScript'),
  document.createTextNode('JScript'),
  document.createTextNode('ActionScript'),
  document.createTextNode('EX4')
];
// On crée le paragraphe
var paragraph      = document.createElement('p');
var paragraphText = document.createTextNode('Langages basés sur
ECMAScript :');
paragraph.appendChild(paragraphText);
// On crée la liste, et on boucle pour ajouter chaque item
var uList = document.createElement('ul'),
  uItem;
for (var i = 0, c=languages.length; i < c; i++) {
  uItem = document.createElement('li');
```

```
uItem.appendChild(languages[i]);
uList.appendChild(uItem);
}
// On insère le tout dans mainDiv
mainDiv.appendChild(paragraph);
mainDiv.appendChild(uList);
// On insère mainDiv dans le <body>
document.body.appendChild(mainDiv);
```

Commentez !

6.2.5 TP : un formulaire interactif (*travail personnel*)

Faire un formulaire c'est bien, mais encore faut-il savoir quoi demander à l'utilisateur. Dans notre cas, nous allons faire simple et classique : un formulaire d'inscription. Notre formulaire d'inscription aura besoin de quelques informations concernant l'utilisateur, cela nous permettra d'utiliser un peu tous les éléments HTML spécifiques aux formulaires que nous avons vus jusqu'à présent. Voici les informations à récupérer ainsi que les types d'éléments HTML :

Information à relever	Type d'élément à utiliser
Sexe	<code><input type="radio" /></code>
Nom	<code><input type="text" /></code>
Prénom	<code><input type="text" /></code>
Âge	<code><input type="text" /></code>
Pseudo	<code><input type="text" /></code>
Mot de passe	<code><input type="password" /></code>
Mot de passe (confirmation)	<code><input type="password" /></code>
Pays	<code><select></code>
Si l'utilisateur souhaite recevoir des mails	<code><input type="checkbox" /></code>

Bien sûr, chacune de ces informations devra être traitée afin que l'on sache si le contenu est bon. Par exemple, si l'utilisateur a bien spécifié son sexe ou bien s'il n'a pas entré de chiffres dans son prénom, etc. Dans notre cas, nos vérifications de contenu ne seront pas très poussées pour la simple et bonne raison que nous n'avons pas encore étudié les « regex » à ce stade du cours, nous nous limiterons donc à la vérification de la longueur de la chaîne ou bien à la présence de certains caractères. Bref, rien d'incroyable, mais cela suffira amplement car le but de ce TP n'est pas vraiment de vous faire analyser le contenu mais plutôt de gérer les événements et le CSS de votre formulaire.

Voici donc les conditions à respecter pour chaque information :

Information à relever	Condition à respecter
Sexe	Un sexe doit être sélectionné
Nom	Pas moins de 2 caractères
Prénom	Pas moins de 2 caractères
Âge	Un nombre compris entre 5 et 140
Pseudo	Pas moins de 4 caractères
Mot de passe	Pas moins de 6 caractères
Mot de passe (confirmation)	Doit être identique au premier mot de passe
Pays	Un pays doit être sélectionné
Si l'utilisateur souhaite recevoir des mails	Pas de condition

Concrètement, l'utilisateur n'est pas censé connaître toutes ces conditions quand il arrive sur votre formulaire, il faudra donc les lui indiquer avant même qu'il ne commence à entrer ses informations, comme ça il ne perdra pas de temps à corriger ses fautes. Pour cela, il va vous falloir afficher chaque condition d'un champ de texte quand l'utilisateur fera une erreur.

6.2.6 Manipulation : formulaire avec jquery

Soit la saisie d'un simple formulaire contenant des informations de connexion (Email, mot de passe) avec une vérification du respect des formats. La gestion des évènements est facilitée via jquery. Reprenez le code suivant et comprendre tous ses détails.

Exercice personnel : Améliorer le rendu visuel en visitant les sites de "jquery UI" (jquery user interface).

```
<!doctype html>
<html lang="en">

<head>
  <meta charset="utf-8">
  <title>jQuery</title>
  <script src="js/jquery-3.3.1.js"></script>
  <script src="js/jquery-ui.js"></script>

  <style>
    * {
      font-size: 14pt;
    }
  </style>
</head>
<body>
  <h1>jQuery UI Example</h1>
  <input type="text" value="Hello World!">
</body>
```

```
}

form {
    width: 600px;
    border: 1px solid black;
    margin: 10px auto 10px auto;
}

label,
input,
button {
    display: inline-block;
    width: 44%;
    height: 30px;
    margin: 10px 1% 10px 1%
}

input:invalid {
    color: red;
}

input:valid {
    color: lime;
}
</style>

</head>

<body>

<body>
<script>
    $("button").click(function () {
        var saisie = "nature envoi :";
        saisie += $("input:hidden").val();
        saisie += " nom et prénom :";
        saisie += $("input:eq(1)").val();
        saisie += " adresse mail : ";
        saisie += $("input:eq(2)").val();
        saisie += " téléphone :";
        saisie += $("input:eq(3)").val();
        saisie += " date début :";
        saisie += $("input:eq(4)").val();
    })
</script>
</body>
```

```
saisie += " durée ( en jours ) :";
saisie += $("input:eq(5)").val();
alert(saisie);
});

</script>

<script>
function fenvoi() {
    var mail1 = $("#mail1").val();
    var mail2 = $("#mail2").val();
    var passe1 = $("#passe1").val();
    var passe2 = $("#passe2").val();

    if (mail1 != mail2) { alert("adresses mail différentes");
return false; }
    if (passe1 != passe2) { alert("mots de passe différents");
return false; }
}
</script>

<h1>Réservation en ligne </h1>
<form action="inscription_trait.htm" method='post'
onsubmit="return fenvoi()">
    <p>Dès qu'une saisie est valide le texte devient vert !</p>
    <label>Tapez votre adresse mail : </label>
    <input type="email" required placeholder="n'oubliez pas @" id='mail1'>
    <label>Confirmez votre adresse mail : </label>
    <input type="email" required id='mail2'>
    <label>Saisissez mot de passe :</label>
    <input type="password" required pattern='[A-z0-9]{6,8}' maxlength='8' placeholder='entre 6 et 8 lettres ou chiffres' id='passe1'>
    <label>Confirmer le mot de passe : </label>
    <input type="password" required pattern='[A-z0-9]{6,8}' maxlength='8' id='passe2'>
    <button type="submit">Envoi</button>
</form>
</body>
</html>
```

6.3 Fiche TP n°3 : Prise en main PHP

Installation EasyPHP : EasyPHP est un pack fonctionnant sous Windows permettant d'installer en un clin d'oeil les éléments nécessaires au fonctionnement d'un site web dynamique développé en PHP. Le pack EasyPHP est disponible sur le site suivant :

<https://www.easypHP.org/>

Il vous suffit dans un premier temps de télécharger la version la plus récente de EasyPHP et l'installer dans un répertoire local, facile à utiliser. Pour vérifier si EasyPHP fonctionne, il vous suffit de taper dans votre navigateur préféré :

<http://localhost>

ou

<http://127.0.0.1>

Pour créer votre site web dynamique avec EasyPHP, il vous suffit de déposer vos créations dans le sous-répertoire /www de EasyPHP. Par exemple créez un fichier texte contenant le texte suivant :

```
<?
phpinfo();
?>
```

On peut tout à fait générer du code HTML avec le script PHP, comme ceci par exemple :

```
<?php
$nom = "Mourad"; // en PHP, toutes les variables sont préfixées par
un dollar ($)
echo "<h1>TP PHP</h1>\n"; // des balises HTML !
echo "<p>Bonjour $nom !<br />...</p>\n"; // encore des balises HTML !
?>
```

Le langage PHP vous fournit la fonction `phpinfo()` qui affiche de nombreuses informations sur la configuration de PHP : options de compilation, extensions, version, informations sur le serveur, et l'environnement (lorsqu'il est compilé comme module), environnement PHP, informations sur le système, chemins, valeurs générales et locales de configuration, en-têtes HTTP et la licence PHP.

```
<html>
<head>
<title>phpinfo.php</title>
</head>
<body>
<?php
phpinfo();
```

```
?>
</body>
</html>
```

6.3.1 Question / variables superglobales

Les variables PHP affichées par la fonction `phpinfo()` sont appelées des variables superglobales en PHP. Depuis la version 4 de PHP, ces variables pré-définies sont stockées dans des tableaux superglobaux tels que `$_GET`, `$_POST` et `$_SERVER`, etc. Pour plus d'informations, lisez la section superglobals du manuel PHP :

www.php.net/manual/fr/language.variables.predefined.php.

Écrire un script PHP qui permet d'obtenir l'affichage suivant :

L'adresse IP du serveur : 192.168.52.1

Votre adresse IP est : 192.168.52.2

Votre système : Linux

6.3.2 Question / typage des variables

Comme d'autres langages de script, PHP utilise un typage dynamique faible pour ses variables. De ce fait, PHP ne nécessite pas de déclararation de type ni d'initialisation pour manipuler des variables. Cette spécificité affectera la sécurité de vos scripts.

Le langage vous fournit plusieurs fonctions pour déboguer les variables à l'exécution :

```
<?php
$a = true;
echo "<pre>"; var_dump($a); echo "</pre>"; echo "<br />";
$a = 10;
echo "<pre>"; var_dump($a); echo "</pre>"; echo "<br />";
$a = 3.141;
echo "<pre>"; var_dump($a); echo "</pre>"; echo "<br />";
$a = "hello $a";
echo "<pre>"; var_dump($a); echo "</pre>"; echo "<br />";
$a = 'bonjour $a';
echo "<pre>"; var_dump($a); echo "</pre>"; echo "<br />";
$a = array('dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi',
'vendredi');
// 'dimanche' est l'index 0 de ce tableau
echo "<pre>"; var_dump($a); echo "</pre>"; echo "<br />";
?>
```

Testez le script ci-dessus. Qu'indiquent les valeurs affichées entre parenthèses pour array

Les tableaux PHP peuvent aussi être multidimensionnel :

```
<?php
    echo "Un tableau multidimensionnel affiché avec var_export() :<br />";
    $a = array(array("a", "b", "c"),array("d", "e", "f"),array("g",
"h", "i"));
    echo "<pre>"; var_export($a); echo "</pre>";
    $jour = array('dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi',
'vendredi', 'samedi');
    $date["jour"] = $jour;
    $date["mois"] = array('janvier', 'février', 'mars', 'avril', 'mai',
'juin',
'juillet', 'aout', 'septembre', 'octobre', 'novembre', 'décembre');
    echo "Un tableau associatif affiché avec var_dump() :<br />";
    echo "<pre>"; var_dump($date); echo "</pre>";
    echo "<br />";
    // Une exploitation de ce tableau
    $j = date("w");
    $m = date("n");
    $message = "Le ".$date["jour"][$j]." ".date("j")."
".$date["mois"][$m-1]."<br />";
    echo "Utilisation d'un tableau associatif :<br />";
    echo $message;
?>
```

Testez le script ci-dessus. En utilisant le manuel PHP pour la fonction date() , que permettent de faire les paramètres passés en argument de cette fonction ?

Écrire un script PHP qui permet d'obtenir l'affichage (en français) suivant :

Bonjour,
On est le samedi 16 février 2013 et il est 11:51:36

6.3.3 Exercice / les fonctions

Soit la fonction PHP suivante :

```
// retourne vrai si l'année $annee passée en argument est bissextile
// sinon faux
function estAnneeBissextile($annee)
{
```

```

$estMultipleDeQuatreCent = ( ($annee % 400) == 0);
$estMultipleDeQuatre = ( ($annee % 4) == 0);
$estPasMultipleDeCent = ( ($annee % 100) != 0);
return ( $estMultipleDeQuatreCent || ( $estMultipleDeQuatre &&
$estPasMultipleDeCent ) );
}

```

Écrire un script qui affiche le nombre d'années bissextiles que vous avez vécu depuis votre naissance en indiquant lesquelles.

Vous avez vécu 13 années bissextiles : 1968 1972 1976 1980 1984 1988 1992 1996 2000 2004
2008 2012 2016

6.3.4 Exercice / passage de paramètres dans l'URL

Cet exercice a pour objectif de montrer comment on récupère des données passées en paramètres de l'url. Cette technique est très utilisée dans la réalisation d'applications PHP.

Jusqu'à la version PHP 4.2.0, les paramètres passés dans l'url étaient automatiquement connus sous forme de variables globales du script destinataire. Mais depuis la version 4.2.0, ce n'est plus le cas à cause du changement de la valeur par défaut (auparavant à On et désormais à Off) du paramètre register_globals du fichier de configuration php.ini du serveur. Ce changement impose de recourir désormais aux tableaux dit superglobaux de PHP (\$_GET[], \$_POST[], etc ...). Ces variables superglobales sont accessibles de partout dans un script php (ne pas mettre global).

Le principe est le suivant :

paramètre id dans url : exemple.php?id=4
alors dans le script exemple.php : \$_GET['id'] sera égal à 4

En phase d'apprentissage ou de débogage, il est recommandé de faire un var_dump(\$_GET);

```

<?php
// vérifie que le paramètre year n'est pas vide
if(!Empty($_GET["year"]))
{
// récupère le paramètre year
$year = $_GET["year"];
// Attention les paramètres d'url sont passées sous forme de chaîne
// de caractères
echo "<pre>Débogage variable year : "; var_dump($year); echo
"</pre>";
// vérifie que le paramètre year est valide

```

```

if(ctype_digit($year)) // cf. is_numeric()
{
echo "est-ce-que $year est bissextile ?<br /><br />";
// est-ce-que cette année est bissextile ?
// TODO
}
else echo "Paramètre year invalide !<br /><br />";
}
else echo "Paramètre year manquant !<br /><br />";
echo "Essayez avec ces paramètres :<br />";
echo "<a href=\"exercice-___.php?year=\"$>vide</a><br />";
echo "<a href=\"exercice-___.php?year=2000\">2000</a><br />";
echo "<a href=\"exercice-___.php?year=2007\">2007</a><br />";
echo "<a href=\"exercice-___.php?year=year\">invalide</a><br />";
?>

```

Testez le script ci-dessus. Que permettent de faire les fonctions Empty() et ctype_digit() utilisées dans ce script ?

Compléter le script ci-dessus afin qu'il affiche toutes les années depuis votre naissance sous forme de lien avec en paramètre l'année en question et qui affichera si cette année est bissextile ou non. Pour cela, utilisez la fonction estAnneeBissextile() écrite à l'exercice précédent.

Exemple d'affichage attendu :

```

2000 est une année bissextile !
1966 1967 1968 1969 1970 1971 1972 1973 1974 1975
1976 1977 1978 1979 1980 1981 1982 1983 1984 1985
1986 1987 1988 1989 1990 1991 1992 1993 1994 1995
1996 1997 1998 1999 2000 2001 2002 2003 2004 2005
2006 2007

```

6.3.5 Exercice / traitement de formulaire

Cet exercice a pour objectif de montrer comment on récupère des données envoyées par un formulaire. Cette technique est très utilisée dans la réalisation d'applications PHP.

Le principe est le suivant :

```

<form action="exemple.php" method="POST" name="form">
<input type="hidden" name="id" value="4">
<input type="submit" value="Envoyer">
</form>

```

alors dans le script exemple.php : `$_POST['id']` sera égal à 4 .

En phase d'apprentissage ou de débogage, il est recommandé de faire un `var_dump($_POST)` ou `var_dump($_GET)`.

```
<?php  
echo "<form action=\"exemple.php\" method=\"POST\" name=\"form\">  
<input type=\"text\" name=\"nom\" value=\"\">  
<input type=\"submit\" value=\"Envoyer\">  
</form>  
"  
?>
```

exemple.php

```
<?php  
if(!Empty($_POST["nom"]))  
{  
    $nom = $_POST["nom"];  
    $heure = date("H");  
    if($heure >= 18)  
        $message = "Bonsoir $nom,<br />";  
    else $message = "Bonjour $nom,<br />";  
    echo $message;  
}  
echo "<br />";  
?>
```

Testez le script ci-dessus.

Écrire un script qui envoie par un formulaire une année choisie dans une liste déroulante et qui affichera si cette année est bissextile ou non. La liste déroulante contiendra toutes les années depuis votre naissance.

Exemple d'affichage attendu :

2000 ▾

Envoyer

6.4 Fiche TP n°4 : Serveur+Client

Dans un premier temps, nous commençons par apprivoiser la communication entre Serveur et Client via l'exemple donné dans la section 3.3.8 où il est question de :

- Créer une base de données
- Un fichier html "ajax.html" pour saisir des données et interroger le serveur
- Un programme "ajax-example.php" qui va réadir aux requêtes du client

Exercice : remplacer l'ancienne façon de se connecter à une BD mysql adopté du côté serveur par PDO, comme expliqué dans la section 3.2.3.

6.5 Fiche TP n°5 : services web

6.5.1 Service web *restful*

Exploiter le service web *restful* décrit dans la section 4.4 dans une application Client qui propose deux formulaires : un formulaire pour inscription ; un deuxième pour le login.

6.5.2 Service web SOAP (WSDL généré)

Implémenter le service web SOAP décrit dans la section 4.5.

6.5.3 Service web SOAP (WSDL disponible)

Implémenter le service web SOAP décrit dans la section 4.6.

7 Contrôles continus

7.1 Fiche Contrôle 1 : réaliser des formulaires

[Test la semaine du dimanche 12 au jeudi 16 avril 2020]

Le test portera sur la conception d'un formulaire pour récupérer des informations sur un sujet précis. Dans la suite de cette fiche, nous introduisons un exemple qui comporte des informations variées pour se familiariser à la conception des formulaires.

(http://www.enseignement.polytechnique.fr/informatique/profs/Olivier.Serre/Memos/Formulaires_CSS/index.html)

Un formulaire est inséré entre les balises `<form ...> ... </form>`. En XHTML, un formulaire ne peut contenir que des balises de type bloc, dont notamment les champs d'entrée ne font pas partie... Une astuce consiste à ajouter un bloc de type `<div>` comme premier élément du formulaire comme suit: `<form ...> <div> ... </div> </form>`.

Un formulaire est ensuite composé de champs d'entrée de texte à une (`<input type="text" ...>`) ou plusieurs lignes (`<textarea ... />`), de menus déroulants (`<select ...> <option> ... </option> ... </select>`) à choix unique ou multiple (`multiple="multiple"`), de cases à cocher (*check boxes*, `<input type="checkbox" ... />`), de cases d'option (*radio buttons*, `<input type="radio" ... />`), de champs de choix de fichier à télécharger sur le serveur, et *last, but not least*, de boutons déclenchant la remise à zéro des entrées effectuées (`<input type="reset" ... />`) ou l'envoi des données au serveur (`<input type="submit" ... />`).

Concernant le formatage des formulaires, il faut en distinguer deux niveaux. D'un côté, l'apparence des différents éléments constituant le formulaire peut être modifiée avec du CSS en changeant couleurs et images de fonds, polices etc. De l'autre côté, il faut agencer dans la fenêtre du navigateur les différents éléments d'entrée proprement dits ainsi que les textes qui les expliquent à l'utilisateur. Pour cela, il est possible d'utiliser des tableaux ou, pour un formatage plus flexible, des blocs de type `<div>`. Dans les deux cas, le formatage par CSS s'impose.

Plusieurs associations organisent une sortie commune auquel leurs membres sont conviés. Vous êtes chargé de gérer les inscriptions au repas. Vous pensez tout de suite à créer une page Web sur laquelle les membres des associations pourront s'inscrire.

On aimeraient arriver au résultat suivant :

Inscriptions

Prénom :

Mot de passe :

Association :

Disponibilités pour la semaine du 22 juin : Lundi Mardi Mercredi Jeudi Vendredi

Contribution : Entrée Plat Dessert

Commentaires :

Exercez-vous : Créez un formulaire permettant à un membre d'une de ces associations de s'inscrire au repas, de donner ses disponibilités dans un ensemble de dates proposées et de spécifier quel type de goûter il préparera. Ce formulaire doit contenir au moins un élément de chaque type d'entrée de formulaire possible (champ texte, menu déroulant, cases à cocher, etc.).

Voilà ci-dessous un exemple de Formulaire à la sauce HTML5 avec le code qui va avec. Testez-le (avec plusieurs navigateurs). Regardez les commentaires dans le code et n'hésitez pas à consulter la page suivante: W3schools.

Formulaire de participation

Complétez le formulaire. Les champs marqué par * sont obligatoires

Contact

Nom *	<input type="text" value="Prénom Nom"/>
Portable	<input type="text" value="06xxxxxxxx"/>
Email *	<input type="text" value="prenom.nom@univoran1.dz"/>

INFORMATION PERSONNELLES

Age*	<input type="text" value="xx"/>
Homme/Femme	<input type="text" value="Homme"/>
Pourquoi voulez-vous vous impliquer dans l'organisation ?	<input type="text"/>

CHOISISSEZ VOTRE "GOÛTER" FAVORIS

- Fruit
- Yaourt
- Omelette
- Sucré
- Salé

Code HTML:

```
<html>

<head>
    <title>This is a page title</title>
    <link href="st.css" media="all" rel="stylesheet" type="text/css">
/>
</head>

<body>

    <h2>Formulaire de participation</h2>
    <form action="#">
        <p><i>Complétez le formulaire. Les champs marqué par * sont obligatoires</i></p>
        <fieldset>
            <legend>Contact</legend>
```

```

        <label for="nom">Nom <em>*</em></label>
        <!-- //placeholder: indication grisée //required: il faut
renseigner le champs sinon la validation est bloquée //autofocus: le
curseur est positionné dans cette case au chargement de la page -->
        <input id="nom" placeholder="Olivier Serre" autofocus="" required=""><br>
        <label for="telephone">Portable</label>
        <!-- // type="tel": bascule le clavier sur un smartphone
// pattern: expression régulière à vérifier pour pouvoir valider -->
        <input id="telephone" type="tel" placeholder="06xxxxxxxx" pattern="06[0-9]{8}"><br>
        <label for="email">Email <em>*</em></label>
        <input id="email" type="email"
placeholder="prenom.nom@polytechnique.edu" required="" pattern="[a-zA-Z]*.[a-zA-Z]*@polytechnique.edu"><br>
    </fieldset>
    <fieldset>
        <legend>Information personnelles</legend>
        <label for="age">Age<em>*</em></label>
        <!-- // type="number": bascule le clavier sur un
smartphone -->
        <input id="age" type="number" placeholder="xx"
pattern="[0-9]{2}" required=""><br>
        <label for="hommefemme">Homme/Femme</label>
        <select id="hommefemme">
            <option value="F" name="hommefemme">Femme</option>
            <option value="H" name="hommefemme">Homme</option>
        </select><br>
        <label for="comments">Pourquoi voulez-vous vous impliquer
dans l'organisation ?</label>
        <textarea id="comments"></textarea>
    </fieldset>

    <fieldset>
        <legend>Choisissez votre "goûter" favoris</legend>
        <label for="fruit"><input id="fruit" type="checkbox"
name="mygouter" value="fruit"> Fruit</label>
        <label for="yaourt"><input id="yaourt" type="checkbox"
name="mygouter" value="yaourt"> Yaourt</label>
        <label for="omelette"><input id="omelette"
type="checkbox" name="mygouter" value="omelette"> Omelette</label>
        <label for="sucree"><input id="sucree" type="checkbox"
name="mygouter" value="sucree"> Sucré</label>
    
```

```
<label for="salee"><input id="salee" type="checkbox" name="mygouter" value="salee"> Salé</label>
</fieldset>
<p><input type="submit" value="Soumettre"></p>
</form>
</body>

</html>
```

Code CSS :

```
p {
    margin-top: 0px;
}

fieldset {
    margin-bottom: 15px;
    padding: 10px;
}

legend {
    padding: 0px 3px;
    font-weight: bold;
    font-variant: small-caps;
}

label {
    width: 110px;
    display: inline-block;
    vertical-align: top;
    margin: 6px;
}

em {
    font-weight: bold;
    font-style: normal;
    color: #f00;
}

input:focus {
    background: #eaeaea;
}

input,
textarea {
```

```
width: 249px;  
}  
  
textarea {  
    height: 100px;  
}  
  
select {  
    width: 254px;  
}  
  
input[type=checkbox] {  
    width: 10px;  
}  
  
input[type=submit] {  
    width: 150px;  
    padding: 10px;  
}
```

7.2 Contrôle 2

7.2.1 Côté client

7.2.2 Côté serveur