

# Architecture et programmation du web - HLIN607

Michel Meynard

14 janvier 2016



# Table des matières

<b>Avant-propos</b>	<b>vii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Prérequis . . . . .	1
1.2 Objectifs . . . . .	1
1.3 HTTP et architecture du Web . . . . .	1
1.3.1 Côté Serveur . . . . .	2
1.3.2 Côté Client . . . . .	2
<b>2 Le langage HTML, XHTML, HTML5</b>	<b>3</b>
2.1 Introduction . . . . .	3
2.1.1 Vocabulaire . . . . .	3
2.2 Syntaxe de HTML5 . . . . .	4
2.2.1 Généralités . . . . .	4
2.3 Le corps du document . . . . .	5
2.3.1 Attributs de base . . . . .	5
2.3.2 Eléments indispensables . . . . .	5
2.4 Les formulaires . . . . .	6
2.4.1 Eléments de formulaires . . . . .	7
2.4.2 La validation des données et l'aide à la saisie . . . . .	8
2.5 Inclusion d'autres objets . . . . .	9
2.5.1 Audio . . . . .	9
2.5.2 Video . . . . .	9
2.5.3 Source . . . . .	10
2.5.4 Object . . . . .	10
2.6 HTML5 . . . . .	10
<b>3 Les feuilles de styles CSS</b>	<b>11</b>
3.1 Introduction . . . . .	11
3.2 Objectifs . . . . .	11
3.3 Syntaxe . . . . .	11
3.4 Quelques exemples simples . . . . .	12
3.5 Localisation des styles . . . . .	13
3.6 Sélecteur . . . . .	14
3.6.1 Correspondance de modèle (pattern matching) . . . . .	14
3.6.2 Sélecteur multiple <input type="checkbox"/> . . . . .	14
3.6.3 classes de style <input type="checkbox"/> . . . . .	14
3.7 Modèle de visualisation . . . . .	16
3.7.1 La propriété css position . . . . .	16
3.7.2 La propriété float . . . . .	17
3.8 Type de media et résolution d'écran . . . . .	17
3.9 Responsive Web Design . . . . .	18

<b>4</b>	<b>Le framework CSS Bootstrap</b>	<b>19</b>
4.1	Introduction . . . . .	19
4.2	Avantages de l'utilisation d'un framework . . . . .	19
4.3	Caractéristiques . . . . .	19
4.4	Installation . . . . .	19
4.4.1	Remarques . . . . .	20
4.5	Premier site . . . . .	20
4.6	La grille . . . . .	20
4.7	Éléments de base . . . . .	21
<b>5</b>	<b>Le langage PHP</b>	<b>25</b>
5.1	Introduction . . . . .	25
5.2	Caractéristiques . . . . .	25
5.3	Structure du langage . . . . .	26
5.3.1	Types . . . . .	26
5.3.2	Typage dynamique . . . . .	26
5.3.3	Existence et valeur . . . . .	26
5.3.4	Constantes . . . . .	26
5.3.5	Le type tableau associatif . . . . .	26
5.3.6	Les expressions . . . . .	27
5.3.7	Structures de contrôle . . . . .	28
5.3.8	Fonctions . . . . .	28
5.3.9	Tableaux super-globaux . . . . .	30
5.4	Gestion de formulaire . . . . .	30
5.5	Le modèle objet depuis PHP5 . . . . .	31
5.5.1	Résumé . . . . .	31
5.5.2	Exemple . . . . .	31
5.5.3	Propriétés . . . . .	31
5.5.4	Constructeur et destructeur . . . . .	31
5.5.5	Méthodes . . . . .	31
5.5.6	Héritage (extends) . . . . .	31
5.5.7	Interfaces . . . . .	32
5.5.8	Classe abstraite . . . . .	32
5.5.9	Méthodes magiques . . . . .	32
5.5.10	Clonage . . . . .	32
5.5.11	Comparaison d'objets . . . . .	32
5.5.12	L'interface iterator . . . . .	32
5.6	Sérialisation . . . . .	33
5.7	Caractères spéciaux . . . . .	33
5.8	Administration PHP . . . . .	33
5.9	Session . . . . .	34
5.9.1	Que mettre comme variable de session ? . . . . .	35
5.9.2	Session et Objets . . . . .	35
5.10	Bases de données avec PDO . . . . .	36
5.10.1	Introduction à PDO . . . . .	36
5.10.2	Connexions et gestionnaire de connexion . . . . .	36
5.10.3	Requêtes uniques . . . . .	37
5.10.4	Requêtes préparées (PDOStatement) . . . . .	37
5.10.5	Transaction . . . . .	38
5.10.6	Synopsis des classes PDO et PDOStatement . . . . .	39
5.10.7	Un exemple de TP . . . . .	40
5.11	MySQL . . . . .	40
5.11.1	phpMyAdmin . . . . .	40
5.11.2	Moteur de stockage . . . . .	40
5.12	Cookies . . . . .	40
5.13	Cookies et session . . . . .	41
5.14	Authentification . . . . .	41
5.14.1	Authentification HTTP . . . . .	41
5.15	Téléchargement . . . . .	42

5.15.1	Déchargement . . . . .	42
5.15.2	Chargement . . . . .	42
5.16	Divers . . . . .	43
5.16.1	SPL : la librairie standard PHP . . . . .	43
5.16.2	Phar : les archives PHP . . . . .	43
5.17	Développement et débogage . . . . .	43
5.17.1	Contexte de débogage . . . . .	43
5.17.2	Gestion des erreurs . . . . .	44
<b>6</b>	<b>Architecture de site</b>	<b>45</b>
6.1	Introduction . . . . .	45
6.2	Analyse des besoins . . . . .	45
6.3	Cahier des charges . . . . .	45
6.3.1	Les exigences . . . . .	45
6.3.2	Le calendrier . . . . .	46
6.3.3	Les ressources . . . . .	46
6.3.4	Maintenance . . . . .	46
6.4	Architecture . . . . .	46
6.4.1	Arborescence du site . . . . .	46
6.4.2	Soignez l'accueil . . . . .	46
6.4.3	Menu clair et accessible . . . . .	47
6.4.4	Plan du site . . . . .	47
6.4.5	Intégrer un moteur de recherche . . . . .	47
6.4.6	Communiquez avec vos visiteurs . . . . .	47
6.4.7	Mentions légales . . . . .	47
6.4.8	Faites des liens . . . . .	47
6.5	La charte graphique . . . . .	47
<b>7</b>	<b>Propel</b>	<b>49</b>
7.1	Introduction . . . . .	49
7.2	Installation de Propel 1.7 . . . . .	49
7.2.1	Composer : un gestionnaire de dépendances . . . . .	49
7.2.2	Installation . . . . .	49
7.3	Fondements de l'utilisation : opérations C.R.U.D. . . . .	50
7.3.1	Création de ligne . . . . .	50
7.3.2	Recherche de lignes . . . . .	50
7.3.3	Mise à jour (Update) . . . . .	51
7.3.4	Suppression (Delete) . . . . .	51
7.4	Construction d'un projet (build) . . . . .	52
7.4.1	Configuration de construction (buid.properties) . . . . .	52
7.4.2	Génération de la BD . . . . .	53
7.4.3	Rétro-ingénierie . . . . .	53
7.4.4	Utilisation à l'exécution de Propel . . . . .	53
7.4.5	Erreurs fréquentes . . . . .	54
<b>8</b>	<b>Doctrine</b>	<b>57</b>
8.1	Introduction . . . . .	57
8.2	Le gestionnaire d'entité et la persistance . . . . .	57
8.2.1	Classe d'entité et métadonnées . . . . .	58
8.2.2	Génération du schéma de BD : la ligne de commande doctrine . . . . .	58
8.2.3	Utilisation de la BD . . . . .	59
8.3	Consultation des entités (query) . . . . .	59
8.3.1	EntityRepository (dépôt d'entités) . . . . .	59
8.3.2	Par clé primaire . . . . .	59
8.3.3	Par des conjonctions d'égalité (where name='x' and id=5) . . . . .	60
8.3.4	Doctrine Query Language (DQL) . . . . .	60
8.4	Associations entre entités . . . . .	61
8.4.1	Association unidirectionnelle "one to one" . . . . .	61
8.4.2	Association bidirectionnelle "one to one" . . . . .	61

8.4.3	Association unidirectionnelle many-to-one . . . . .	62
8.4.4	Association bidirectionnelle one-to-many . . . . .	63
8.4.5	Association unidirectionnelle many-to-many . . . . .	63
8.4.6	Héritages . . . . .	64
8.5	Rétro-ingénierie . . . . .	64
8.5.1	Métadonnées . . . . .	64
8.5.2	Retour d'expériences . . . . .	65
8.5.3	Génération des entités . . . . .	65
8.5.4	Comparaison Propel/Doctrine . . . . .	65
8.6	Les fichiers de configuration . . . . .	65
8.6.1	Installation . . . . .	65
8.6.2	bootstrap.php . . . . .	66
8.6.3	Ligne de commande doctrine . . . . .	66
8.7	Bugs et astuces . . . . .	67
8.7.1	MAMP spécifique . . . . .	67
8.7.2	Ligne de commande . . . . .	67
8.8	DBAL DataBase Access Layer . . . . .	67
8.9	Conclusion . . . . .	67
<b>Solutions des exercices</b>		<b>71</b>

# Avant-propos

Ce cours s'inscrit dans un enseignement d'informatique, discipline scientifique dispensée à la faculté des sciences de l'université de Montpellier. L'évolution des connaissances scientifiques dans l'histoire depuis la Grèce antique jusqu'à nos jours a souvent remis en cause des connaissances plus anciennes ou des dogmes religieux. Au IV<sup>e</sup> siècle avant notre ère, le grec Anaxagore de Clazomènes est le premier savant de l'histoire à être accusé d'impiété par les autorités religieuses de l'époque ["Bruno et Galilée au regard de l'infini" de Jean-Pierre Luminet]. Il ne doit sa liberté qu'à son amitié avec Périclès. Plus tard, Giordano Bruno (1548-1600) invente la cosmologie infinitiste mais également le principe d'inertie. Ses idées et ses écrits contraires à la doctrine chrétienne le conduisent à être incarcéré au total pendant huit ans dans les geôles de l'Inquisition. Après de multiples procès, le 16 Février de l'an de grâce 1600, Giordano BRUNO est torturé et brûlé vif, par l'inquisition catholique, à Rome, sur le Campo dei Fiori, pour avoir refusé d'abjurer ses idées. Plus tard, Galilée, Kepler auront également des problèmes juridiques liés à l'expression de leurs idées scientifiques révolutionnaires.

En France, le siècle des lumières puis la révolution française de 1789 ont permis de donner la liberté d'expression aux scientifiques (et aux autres) afin que leurs travaux de recherche puissent être publiés, discutés, réfutés ou approuvés. La loi de séparation des Églises et de l'État a été adoptée le 9 décembre 1905 à l'initiative du député républicain-socialiste Aristide Briand. Elle prend parti en faveur d'une laïcité sans excès. Elle est avant tout un acte fondateur dans l'affrontement violent qui a opposé deux conceptions sur la place des Églises dans la société française pendant presque vingt-cinq ans.

La liberté de pensée et d'expression constituent donc les fondements d'un enseignement universitaire de qualité. D'autres part, les scientifiques étant des citoyens comme les autres, il convient de rappeler quelques lois françaises qui nous gouvernent.

## Quelques lois fondamentales

**Art. 1 de la constitution du 4 octobre 1958** La France est une République indivisible, laïque, démocratique et sociale. Elle assure l'égalité devant la loi de tous les citoyens sans distinction d'origine, de race ou de religion. Elle respecte toutes les croyances. Son organisation est décentralisée. La loi favorise l'égal accès des femmes et des hommes aux mandats électoraux et fonctions électives, ainsi qu'aux responsabilités professionnelles et sociales.

**Art. 4 de la Déclaration des Droits de l'Homme et du Citoyen de 1789** La liberté consiste à pouvoir faire tout ce qui ne nuit pas à autrui : ainsi, l'exercice des droits naturels de chaque homme n'a de bornes que celles qui assurent aux autres Membres de la Société la jouissance de ces mêmes droits. Ces bornes ne peuvent être déterminées que par la Loi.

**Art. 5 de la Déclaration des Droits de l'Homme et du Citoyen de 1789** La Loi n'a le droit de défendre que les actions nuisibles à la Société. Tout ce qui n'est pas défendu par la Loi ne peut être empêché, et nul ne peut être contraint à faire ce qu'elle n'ordonne pas.

**Art. 10 de la Déclaration des Droits de l'Homme et du Citoyen de 1789** Nul ne doit être inquiété pour ses opinions, même religieuses, pourvu que leur manifestation ne trouble pas l'ordre public établi par la Loi.

**Art. 11 de la Déclaration des Droits de l'Homme et du Citoyen de 1789** La libre communication des pensées et des opinions est un des droits les plus précieux de l'Homme : tout Citoyen peut donc parler, écrire, imprimer librement, sauf à répondre de l'abus de cette liberté dans les cas déterminés par la Loi.





# Chapitre 1

## Introduction

### 1.1 Prérequis

Ce cours est destiné à des étudiants ayant les connaissances de bases du langage HTML ainsi que du protocole HTTP. Les notions de serveur HTTP, par exemple Apache, et de client léger, navigateur Web, ainsi qu'un minimum de connaissance de Javascript sont supposées acquises. L'ouvrage [1] est indiqué pour s'initier à ces techniques de base.

L'étudiant devra être familier avec la conception de sites statiques à l'aide d'outils de son choix : emacs, Dreamweaver, ... En TP, il devra être capable de rechercher les références concernant HTML, Javascript, les feuilles de style CSS.

### 1.2 Objectifs

Les objectifs de ce cours sont nombreux :

- apprentissage d'un langage de programmation côté serveur PHP ;
- étude de la dynamique d'une application Web : initialisation, saisie, traitement, ...
- technologies avancées : cookies, sessions, téléchargement, ...
- réalisation d'un projet.

### 1.3 HTTP et architecture du Web

Quelques définitions de base sont nécessaires :

**HTTP** **H**yper**T**ext **T**ransfer **P**rotocol est un protocole de communication **client-serveur** développé pour le World Wide Web (www). HTTP a été inventé par Tim Berners-Lee avec les adresses web (URL) et le langage HTML pour créer le World Wide Web. Il utilise généralement le port 80 et est un protocole **non orienté connexion**. Schématiquement, le client envoie une requête au serveur qui lui retourne une réponse généralement sous la forme d'un fichier HTML. Le client affiche alors la réponse. Cette réponse contient généralement des **liens** hypertextes ou des formulaires, qui une fois cliqués ou soumis génèrent une requête au serveur ...

**serveur HTTP** par exemple, Apache, IIS, ... sont des serveurs installés sur des machines.

**client HTTP** appelé aussi client léger ou navigateur Web, les plus connus sont Firefox, Internet Explorer, Opera, ...

**URL** de l'anglais *Uniform Resource Locator* est une chaîne de caractères codée en ASCII pour adresser les ressources du World Wide Web : document HTML, image, son, forum Usenet, boîte aux lettres électroniques, etc. Elle est informellement appelée une **adresse web**. Exemples : `http://www.google.fr/`, `mailto:toto@titi.fr`, `http://www.lirmm.fr/~meynard/Ens2/rubrique.php3?id_rubrique=36&x=1`, ...

**URL relative** à l'intérieur d'une page HTML, des liens vers d'autres pages du même site peuvent être définis avec une écriture relative au répertoire courant de la page : par exemple, `../images/toto.jpg` est une URL relative.

Ce qu'il faut retenir absolument, c'est que HTTP n'est pas orienté connexion c'est-à-dire que le serveur ne mémorise aucune information à propos du client et qu'entre 2 requêtes du client, il peut se passer 2 secondes ou un temps infini ! Par conséquent, l'écriture des applications côté serveur doit prendre en compte cette absence de mémoire.

D'autre part, une part non négligeable du traitement dans une application peut être effectué côté client afin d'effectuer des vérifications (contrôles) qui éviteront de surcharger le serveur. Cependant, des règles de sécurité interdisent aux scripts côté client d'accéder aux ressources du poste client (fichiers, imprimante, ...).

### 1.3.1 Côté Serveur

L'application côté serveur utilise deux technologies possibles :

**cgi** le programme est un script (python, bash) ou un binaire (compilé par g++) qui est chargé dans un processus externe au serveur http. La sortie standard de ce processus est redirigé dans la réponse envoyée par le serveur au client. L'inconvénient principal des cgi est la perte de temps nécessitée par la création d'un processus pour chaque nouvelle requête ;

**module** certains serveurs dont Apache ont intégré des modules interprétant des langages de programmation tels que Perl, Python, PHP. Ainsi, l'interprétation des scripts est beaucoup plus rapide.

### 1.3.2 Côté Client

L'application côté client peut utiliser plusieurs technologies possibles :

**javascript** langage de script interprété par le navigateur et permettant de manipuler l'arborescence du document(DOM) ;

**applet Java** mini application Java permettant d'utiliser toute la puissance du langage Java (API, structures de données, ...)

**ActionScript** langage de script compatible avec JavaScript et permettant de réaliser des animations Flash ...

# Chapitre 2

## Le langage HTML, XHTML, HTML5

### Cours 1

#### 2.1 Introduction

“HyperText Markup Language”, abrégé HTML, est le langage conçu pour représenter les pages web. En août 1991, Tim Berners-Lee (CERN) annonce publiquement le web sur Usenet, en donnant l’URL d’un document de suffixe .html. A l’origine, HTML est basé sur SGML jugé trop complexe. Différentes versions de HTML ont vu le jour jusqu’à la version 4.01. “eXtensible HyperText Markup Language”, abrégé XHTML 1.0, était le langage destiné à remplacer HTML 4 et est une application XML. Une nouvelle norme HTML5 est apparue depuis 2006 sous la houlette du World Wide Web Consortium (W3C) et du Web Hypertext Application Technology Working Group (WHATWG). Beaucoup des recommandations de cette norme sont implémentées sur les navigateurs récents mais certaines de ses fonctionnalités ne le sont pas. Nous utiliserons le HTML5 sans examiner toutes les capacités de ce langage.

A l’origine, contenu et format de document étaient mélangés dans un même fichier HTML. C’est-à-dire que l’on pouvait trouver dans un document HTML, des éléments de :

- structuration du contenu tels que `div`, `h1`, `h2`, ..
- mise en forme tels que `b` (bold), `i` (italic), `center`, ...

Actuellement, on utilise des feuilles de style (en cascade) “Cascading Style Sheets” pour le format. La plupart du temps, la feuille de style est externe c’est-à-dire stockée dans un fichier différent référencé (lié) par le fichier HTML. Dans certains cas, un style en-ligne (au fichier) peut être associé.

Actuellement, c’est le W3C (*World Wide Web Consortium*) qui est chargé de rédiger des recommandations (sorte de norme) concernant les technologies du web. L’adresse web de leur site est <http://www.w3.org/>. On peut notamment valider ses documents en ligne à l’adresse <http://validator.w3.org/>. Un autre site d’importance et qui n’a aucun lien avec le W3C est <http://www.w3schools.com/> qui est un site pédagogique très pratique même s’il est commercial.

##### 2.1.1 Vocabulaire

- document : l’ensemble du texte composé du fichier principal et des fichiers inclus ou référencés (script, feuille de style, image, ...);
- langage à balisage : `<balise>contenu ...</balise>`;
- élément : composé d’une balise ouvrante puis d’un contenu (possédant éventuellement d’autres éléments) puis de la balise fermante correspondante ;
- élément vide : ne possédant pas de contenu, la balise ouvrante est également fermante comme dans `<br />`;
- attribut : information de la forme `nom='valeur'` présente uniquement dans une balise ouvrante ;
- validité : un document HTML est valide s’il correspond aux règles édicté par le “World Wide Web Consortium” (w3c) ;
- URI : *Uniform Resource Identifier* adresse d’une ressource Internet composé d’un protocole, d’un nom de domaine complètement qualifié (FQDN), d’un chemin, et d’autres paramètres possibles ; Par exemple, `http://www.lirmm.fr/~meynard/ProjetInfoL3/index.php?rubrique=Projet&action=liste` est une URI. Autre exemple, `mailto:toto@tutu.fr` désigne une adresse email ;

## 2.2 Syntaxe de HTML5

### 2.2.1 Généralités

HTML5 est extrêmement laxiste vis-à-vis de XHTML 1.0. Par exemple, les balises `html`, `head`, `body` sont optionnelles ! Les valeurs d'attribut ne sont pas forcément entourées de guillemets, les éléments vides ne sont pas forcément fermés ... Cependant, pour des raisons de lisibilité du code, nous continuerons à utiliser une syntaxe stricte héritée de `xhtml`.

- les noms d'attribut sont en minuscules ;
- les valeurs d'attribut doivent être entre apostrophes ou bien entre guillemets ;
- les éléments vides sont définis par une seule balise, comme dans `<br />` et dans `` ;
- les attributs présentsiels n'ont pas de valeur puisque leur présence suffit (pseudo valeur booléenne) ; voir l'exemple suivant avec `autofocus` ou `multiple`.

```
<input type="text" name="prenom" autofocus />
<select multiple name="ues" ...
```

#### Exemple 1 (Exemple de document HTML minimum : modele.html)

```
<!doctype html>
<html lang="fr">
<head>
<meta charset="utf-8" />
<!-- <meta charset="iso-latin-1" / -->
<title>mon premier site</title>
</head>
<body>
<header>
<h1></h1>

</header>
<h1>Titre de niveau 1</h1>
<p>hello <b>World</b> ! éâè</p>
</body>
</html>
```

#### Doctype

Une DTD (Document Type Definition) définit la syntaxe d'un type de document : c'est une grammaire formelle indiquant les imbrications possibles d'éléments, les attributs obligatoires, optionnels, ...

La déclaration de la DTD `<!DOCTYPE ...` était auparavant complexe puisqu'elle référençait l'URI du fichier de DTD. Il y avait plusieurs DTD pour XHTML 1 (strict, transitional, frameset), et c'était compliqué !

#### L'élément `html`

C'est la racine du document qui possède l'attribut de langue. Il "peut" contenir à sa suite 2 éléments :

**head** l'en-tête du document ;

**body** le corps du document qui sera affiché sur la page ;

#### L'en-tête du document

L'élément `head` peut et **doit** contenir d'autres éléments que le titre du document `title` :

- l'élément `meta` fournit des méta-informations (ou métadonnées) sur la page en cours qui pourront être utilisées par les moteurs de recherche ou par le navigateur. Ses attributs :

**charset** indique l'encodage du fichier HTML. Pour le français les 2 encodages les plus fréquents sont l'`iso-latin-1` (1 octet par lettre accentuée), ou l'`utf-8` (codage multi-octets mais universel).

**name** est optionnel et peut prendre les valeurs `author`, `description`, `keywords`. La valeur associée à ce nom sera dans l'attribut `content`.

**http-equiv** permet d'envoyer des en-têtes HTTP au navigateur pour l'aider à interpréter le document. Par exemple, `<meta http-equiv="refresh" content="5" />` permet de rafraîchir la page au bout de 5 secondes. La valeur associée à `http-equiv` sera dans l'attribut `content`.

**content** définit la valeur de la méta-information. Par exemple,

```
<meta name="keywords" content="fromage, camembert, produit frais" />
```

- l'élément **script** permet de référencer un fichier script externe et/ou de définir des fonctions JavaScript locales. Par exemple,

```
<script charset="iso-8859-1" src="monscript.js"></script>
```

**Attention** à ne pas créer un élément vide `<script ... />` lorsque tout le code est dans un fichier externe. Cela ne fonctionne pas !

- l'élément **link** permet de lier un autre document. Par exemple, pour définir la feuille de style associée :  

```
<link rel="stylesheet" href="messtyles.css" />
```
- l'élément **style** permet de définir des styles en ligne sans utiliser de fichier externe.

## 2.3 Le corps du document

Le corps (*body*) du document HTML constitue ce qui va être affiché sur l'écran. Par défaut, les éléments sont affichés selon leur type :

- les éléments en ligne (*inline*) sont placés de gauche à droite jusqu'à ce qu'il n'y ait plus de place dans leur bloc et un retour à la ligne automatique est géré par le navigateur ; ainsi le texte d'un paragraphe est-il affiché avec des coupures de lignes dépendant de la taille de la fenêtre du navigateur ; de même les ancres sont des éléments en ligne ;
- les éléments de type **bloc** sont affichés avec une coupure de ligne avant et après eux ; les paragraphes sont de type bloc ainsi que les en-têtes.

Un grand nombre de balises *sémantiques* ont été ajoutées en HTML5 afin d'éviter la multiplication des blocs (**div**) pour des usages différents.

### 2.3.1 Attributs de base

Un certain nombre d'attributs de base communs à quasiment tous les éléments (*core attributes*) peuvent être utilisés.

**class** classe CSS de l'élément. Cela permet de formater l'élément selon un style défini dans une classe déjà définie ; par exemple, `class='monvert'` où `monvert` est une classe de style ;

**style** style en ligne ; par exemple, `style='color:green;'`

**id** identifiant **unique** de l'élément permettant sa manipulation en JavaScript grâce à la méthode :  
`document.getElementById(id) ;`

**title** *tooltip* affiché lors du survol de l'élément ;

De plus, une gestion des événements souris et clavier permettent d'associer des scripts JavaScript. Cette association est réalisée grâce à des attributs tels que `onclick` ou `onkeypress` qui peuvent être associés à un gestionnaire d'événements. Attention à ne pas confondre l'identifiant (`id`) qui est utilisé pour la manipulation côté client et le nom (`name`) des champs de formulaire qui lui caractérise les paramètres des requêtes passées au serveur lors de la soumission.

### 2.3.2 Éléments indispensables

**a** *anchor* les ancres mettent en oeuvre l'hypertexte et peuvent être utilisées de 2 façons :

1. lien hypertexte référençant une autre page web grâce à l'attribut `href`. Par exemple, pour aller sur le site du `w3schools` :

```
<a href='http://www.w3schools.com/'>CLIQUEZ ICI</a>
```

2. marque-page indiquant une cible potentielle d'un lien :

```
<a id='toto'>Texte cible</a>
```

Plus loin dans le même document, on pourra référencer cette cible par le lien suivant :

```
<a href='#toto'>aller au texte cible</a>
```

Bien entendu, on peut faire une référence externe sur une cible en suffixant le chemin de l'url par `#toto` ;

Enfin, la valeur d'un `href` peut-être une pseudo-URL contenant du code JavaScript qui sera exécuté lors de l'activation du lien comme dans l'exemple suivant :

```
<a href="javascript:alert('Bonjour le monde !')">Mon Lien</a>
```

Remarquons que le code est préfixé par `javascript:` afin d'indiquer le langage ;

**script** exécution de code `javascript` dans le corps de page ; par exemple :

```
<script>document.writeln('Bonjour le monde !');</script>
```

**p** paragraphe contenant une suite de phrases ;

**br** *break* élément vide qui permet de passer à la ligne suivante ; en HTML les espaces, tabulations et retour ligne (CR et/ou LF) multiples ne sont perçus que comme un espace séparateur lorsqu'ils sont situés entre deux mots.

**h1** *header* de niveau 1 est un titre de section ; on trouve également **h2** ...

**ul** *unordered list* liste d'items non numérotés ;

**ol** *ordered list* liste d'items numérotés ;

**li** *list item* élément d'une liste numérotée ou non ;

**div** *division* est une section logique du document permettant de regrouper plusieurs blocs dans un même formatage ; on l'utilise souvent dans la mise en page du document pour scinder les différentes parties (bandeau haut, menu de gauche, page centrale, bandeau du bas) ;

**table** les tables HTML sont un moyen d'afficher les tableaux mais pas de mettre en page un document (utiliser plutôt les `div`). Les tables peuvent ou doivent contenir :

**caption** la légende de la table ;

**tr** *table row* une ligne ;

**td** *table delimiter* une case ; `colspan` et `rowspan` sont des attributs indiquant le nombre de cases à fusionner avec la case courante vers la droite et vers le bas ;

**th** *table header* une case de titre de colonne ;

**hr** *horizontal rule* ligne de séparation horizontale ;

**img** *image* en format gif, png ou jpg ; les attributs indispensables :

**src** *source* chemin du fichier image ;

**alt** *alternative* texte utilisé si le navigateur ne sait pas afficher les images ;

**width** largeur en pixels ;

**height** hauteur en pixels ;

**form** les formulaires sont décrits dans la section suivante.

**header** HTML5 à ne pas confondre avec `head`, cet élément contient l'entête visuel de votre page (appelé aussi bandeau haut).

**footer** HTML5 contient le pied de page visuel de votre page (appelé aussi bandeau bas).

**nav** HTML5 contient une liste de liens externes ou internes. Il est typiquement utilisé pour implémenter un menu ou un fil d'Ariane :

```
rayon informatique > clé USB > par marque
```

**article** HTML5 portion de page indépendante du reste de la page ;

**section** HTML5 groupement de contenu incluant généralement un titre ;

**aside** HTML5 groupement de contenu non primordial ;

**time** HTML5 pour spécifier une date compréhensible par javascript ;

**figure**, **figcaption** HTML5 pour insérer un **flottant** composé d'images de textes et d'une légende,

## 2.4 Les formulaires

Les formulaires constituent des éléments indispensables dans la saisie d'informations à destination d'une application web. Un ou plusieurs formulaires peuvent être présents dans une même page. L'attribut `name` de chaque champ de saisie correspond au nom du "paramètre" qui contiendra l'information. Plusieurs champs peuvent avoir le même nom, dans ce cas le paramètre sera de type tableau. L'attribut `tabindex` permet d'ordonner les champs de saisie quand l'utilisateur appuiera sur la touche de tabulation.

### 2.4.1 Éléments de formulaires

Dans HTML5, de nouveaux éléments de saisie sémantiques sont apparus associés à des contrôles de validité automatiques. Ils sont plus ou moins bien implémentés selon l'âge du navigateur.

**form** élément racine d'un formulaire contenant les attributs suivants :

**action** est l'URI où sera soumis le formulaire (généralement un script php ou un programme cgi). Cette URI peut être absolue ou relative au répertoire courant. La norme requiert cet attribut qui doit être une URI, or une URI ne peut être vide mais, la norme HTML4.0 admet l'exception d'attribut vide pour désigner le document courant !

**method** soit get, soit post ; la première (get) insère les champs saisis par l'utilisateur à la fin de l'url après un point d'interrogation sous la forme `http://localhost/index.php?nom1=val1&nom2=val2` ; les noms sont les valeurs des attributs **name** des champs tandis que les valeurs sont les contenus saisis par l'utilisateur. La seconde méthode post "cache" les contenus saisis ;

**target** avec la valeur `_blank`, une nouvelle fenêtre sera ouverte, avec `_self` (par défaut) on recouvrira la fenêtre actuelle.

**onsubmit** code javaScript à exécuter avant la soumission ; Si le code JavaScript retourne faux, la soumission est annulée !

**name** ou **id** nom ou id du formulaire pouvant être utile pour le référencement des champs de saisie dans le code javaScript de vérification ;

**label** texte préfixant le champ de saisie et lié à lui par l'attribut **for**. Indispensable pour l'accessibilité.

**input** élément **vide** définissant un champ de saisie ayant :

- un attribut **name**, le nom du champ ;
- un attribut **type** permettant d'indiquer la forme du champ de saisie parmi :

**text** champ de type texte libre sur une ligne ; autres attributs : **size** la taille du champ de saisie à l'écran, **maxlength** le nombre maximum de caractères à saisir, **value** valeur par défaut ;

**password** champ de mot de passe caché ;

**button** bouton permettant de déclencher un script javascript ; autres attributs : **value** valeur affichée à l'écran, **onclick** code javaScript à déclencher ;

**checkbox** case à cocher ; autres attributs : **value** valeur affectée au nom dans le cas où cette case est cochée, **checked="checked"** si on veut que la case soit cochée par défaut ;

**radio** case à cocher **exclusive** ; tous les boutons radio mutuellement exclusifs doivent avoir le même attribut **name** ; autres attributs : **value** valeur affectée au nom dans le cas où cette case est cochée, **checked="checked"** si on veut que la case soit cochée par défaut ;

**hidden** champ caché n'apparaissant pas dans le formulaire ; historiquement, les champs cachés permettaient de faire transiter l'information passée de proche en proche lors de la navigation ; autres attributs : **value** ; Actuellement, le mécanisme de session est plus pratique à utiliser.

**submit** bouton de soumission permettant d'exécuter l'**action** du formulaire ; plusieurs boutons de soumission peuvent coexister dans un même formulaire ;

**image** bouton de soumission utilisant une image comme visuel ; autres attributs : **src**, **alt**, **width**, **height** ;

**reset** réinitialisation des champs du formulaire ;

**file** pour envoyer au serveur un fichier localisé chez le client ; le formulaire doit posséder un attribut **enctype** ayant la valeur `multipart/form-data` et sa méthode doit être `post` ; Il est également possible de limiter la taille du fichier à envoyer en ajoutant un champs caché nommé `max_file_size` et de valeur la taille maximal acceptée en octets.

**date** HTML5 permet de saisir une date grâce à un calendrier ;

**time** HTML5 permet de saisir un horaire ;

**datetime-local** HTML5 permet de saisir un horaire et un jour ;

**time**, **week**, **month**, **datetime** HTML5 les dates sont retournées selon la norme RFC 3339, par exemple : `1985-04-12T23:20:50.52Z` ;

**number** HTML5 permet de saisir un nombre flottant dans un intervalle ;

```
<input type="number" name="qte" min="10" max="20" step="2" value="12">
```

**range** HTML5 permet de saisir un nombre flottant grâce à un curseur ;

```
<input type="range" name="son" min="10" max="11" step="0.2" value="11">
```

**color** HTML5 permet de saisir une couleur au format hexadécimal #1234bb (RVB) grâce à un nuancier de couleur.

```
<input type="color" name="coul" value="#1234bb">
```

**email** HTML5 permet de saisir une adresse email valide.

**url** HTML5 permet de saisir une url valide.

**tel** HTML5 permet de saisir un numéro de téléphone valide.

**search** HTML5 champ de texte assez classique dont la sémantique est d'indiquer un champ destiné à la recherche d'information.

L'attribut pseudo-booléen **disabled** permet de désactiver un champ de saisie (**input**) qui ne permet alors plus la saisie. Il ne sera pas soumis.

**textarea** élément non vide contenant une zone de texte multi-ligne; attributs : **name**, **rows** nb de lignes, **cols** nb de colonnes;

**select** élément non vide contenant une liste déroulante d'options; attributs : **name**, **size** nb de lignes visibles, **multiple** pseudo-booléen indiquant que plusieurs options sont possibles;

**option** élément non vide contenu dans **select**; attributs : **value** valeur qui sera associée au **name** du **select** conteneur, **selected** de valeur **selected** indique si cette option est présélectionnée; le contenu de l'élément option sera le texte affiché dans la liste déroulante;

**datalist** élément contenant une liste de choix possibles mais non limitatifs. Il permet de suggérer certaines valeurs habituelles pour un champ texte (avec auto-complétion) en permettant une saisie libre. Son attribut **id** devra être référencé en tant que valeur de l'attribut **list** du champ de texte.

```
<datalist id="lprenoms">
  <option value="Pierre">Pierre Durand</option>
  <option value="Michel">Michel Meynard</option>
  <option value="Paul">
</datalist>
<label for="pre">Prénom : </label><input type="text" id="pre" list="lprenoms" name="prenom">
```

### Exemple 2 (Un formulaire de login)

```
<form action="login.php" method="post" >
<label for="log">Nom : </label><input type="text" id="log" size="20" name="login" /><br />
<label for="pwd">Mot de passe : </label><input type="password" id="pwd" size="20" name="passwd" /><br />
<input type="submit" value="Valider">
</form>
```

## 2.4.2 La validation des données et l'aide à la saisie

Avant HTML5, la validation des données côté client devait être exécutée par du code Javascript. Désormais, si cela reste possible, il est plus simple d'utiliser des éléments de formulaires sémantiques (**email**, **url**, **date**, **tel**, **color**, **number**, **range**) qui, par nature, sont vérifiés par le navigateur avant l'envoi du formulaire. De plus, d'autres éléments de langages permettent d'assister l'utilisateur lors de la saisie.

N'oublions pas que la validation côté client permet de ne pas submerger le serveur avec des données erronées ou incomplètes MAIS qu'en aucun cas, elle ne peut être suffisante : il faut absolument vérifier la correction des données soumises côté serveur avant de les traiter. En effet, un pirate peut recopier un formulaire, en supprimer certaines parties permettant de le valider puis le soumettre. C'est encore pire avec la méthode GET qui permet de soumettre des données dans l'URL.

### Expressions régulières

L'attribut **pattern** permet de définir une expression régulière qui doit correspondre au texte saisi dans un champ de type **text** lorsque la soumission est demandée. Dans le cas où la saisie n'est pas correcte, un message "Veuillez respecter le format requis" sera affiché par le navigateur.

### Exemple 3

```
<input type="text" name="nomvar" pattern="[A-Za-z_][A-Za-z_0-9]*">
```



### Renseigner un élément de saisie

En raison des règles d'accessibilité, chaque élément de saisie doit être associé à un élément de `label` ayant un attribut `for` dont la valeur est l'`id` du champ de saisie :

Afin d'indiquer ce que doit contenir un champ, on peut utiliser des attributs tels que :

**placeholder** affiche un texte indicatif grisé dans le champ avant la saisie.

**title** affiche une info-bulle (*tooltip*) à côté du champ.

#### Exemple 4

```
<label for="netu">Numéro d'étudiant sur 8 chiffres</label>
<input type="text" name="numetu" id="netu" pattern="[0-9]{8}" placeholder="Numéro
d'étudiant" title="par exemple : 20131234">
```

### Champ obligatoire et ordre des éléments

Il est maintenant possible, avec l'attribut pseudo-booléen **required** d'imposer la saisie de cet élément. A la validation, les éléments requis non saisis empêcheront la soumission et seront encadrés en rouge.

Afin qu'un élément ait le focus dès l'ouverture de la page, il suffit de lui ajouter (et seulement à lui) l'attribut pseudo-booléen **autofocus**. Pour les autres champs, on utilise l'attribut **tabindex** avec une valeur entière (entre 1 et n) pour indiquer l'ordre logique de saisie. Evidemment, l'élément ayant l'**autofocus** devra avoir un **tabindex="1"**.

### Pour les vieux navigateurs

Afin d'assurer une même interface visuelle et la validation aux différents utilisateurs de vieux navigateurs, on peut utiliser une librairie javascript telle que **H5F** ou **Webforms2** qui se chargent de vérifier la compatibilité du navigateur courant et d'émuler les fonctionnalités manquantes avec du code javascript.

## 2.5 Inclusion d'autres objets

### 2.5.1 Audio

L'élément **audio** (HTML5) permet d'inclure un lecteur audio pour écouter un son au format mp3, wav, ou ogg. Ses attributs :

**src** url du fichier son ;

**controls** pour afficher les boutons de pause, play, ...

**autoplay** pour lancer le son dès le chargement ;

**loop** pour répéter le son ;

**preload** pour charger le son avant la page ;

### 2.5.2 Video

L'élément **video** (HTML5) permet d'inclure un lecteur video pour voir un fil au format mp4, ogg ou wbm. Ses attributs :

**src** url du fichier ;

**controls** pour afficher les boutons de pause, play, ...

**autoplay** pour lancer le film dès le chargement ;

**loop** pour répéter ;

**preload** pour charger le film avant la page ;

**muted** muet ;

**poster** url de l'image de remplacement pendant le téléchargement du film ou avant sa lecture ;

**height, width** taille de l'écran ;

### 2.5.3 Source

Des balises **sources** peuvent être utilisées afin d'indiquer différents fichiers supports en différents formats (audio ou vidéo). Cela assure une plus grande compatibilité car certains navigateurs ne supportent pas certains formats.

#### Exemple 5

```
<video width="320" height="240" controls>
  <source src="movie.mp4" type="video/mp4">
  <source src="movie.ogv" type="video/ogg">
  Votre navigateur ne supporte pas la balise video !
</video>
```

### 2.5.4 Object

L'élément **Object** permet d'inclure un objet typé dans une page HTML. Ses attributs sont :

**type** indique le type de l'objet tel que : "text/html", "application/x-shockwave-flash", "video/x-msvideo", ...

**data** url du fichier tel que : ../commun.html, ma\_video.avi, animflash.swf, ...

**width** et **height** indique la taille de l'inclusion ;

Cet élément n'est pas vide et peut ou doit contenir des sous-éléments **paramètres** utilisés par le navigateur pour un type d'objet particulier.

#### Exemple 6

```
<param name="autostart" value="true" />
<param name="loop" value="true" />
Texte alternatif
```

## 2.6 HTML5

Attention, HTML5 fournit bien d'autres fonctionnalités qui ne sont pas décrites dans ce cours introductif. Nous citerons par exemple :

- Canvas et son API pour dessiner et traiter des images ;
- drag and drop ;
- XMLHttpRequest level 2 ou AJAX 2 ;
- géolocalisation ;
- applications offline (cache) ;
- WebSocket qui offre une connexion et la prog. événementielle ;
- Microdata pour décrire et marquer des parties de pages afin que des programme tiers (robots) puissent extraire automatiquement de l'information ;

# Chapitre 3

## Les feuilles de styles CSS

### 3.1 Introduction

Le langage CSS (Cascading Style Sheets : feuilles de style en cascade) sert à décrire la présentation des documents HTML et XML. Les standards définissant CSS sont publiés par le World Wide Web Consortium (W3C). Introduit au milieu des années 1990, CSS 2.1 est couramment utilisé dans la conception de sites web et est bien pris en charge par les navigateurs web actuels. La documentation de référence absolue est à l'adresse <http://www.w3.org/TR/CSS21/>. La spécification de CSS 2.1 est basée sur le modèle arborescent de document html : **Document Object Model (DOM)**.

CSS3, la nouvelle norme est rétrocompatible. Elle est découpée en modules :

- Selectors ;
- Box Model ;
- Backgrounds and Borders ;
- Text Effects ;
- 2D/3D Transformations ;
- Animations ;
- Multiple Column Layout ;
- User Interface ;

### 3.2 Objectifs

- séparer la structure d'un document de ses styles de présentation ;
- définir le rendu d'un document en fonction du média de restitution et de ses capacités (type de moniteur ou de dispositif vocal), de celles du navigateur textuel, ...
- permettre la cascade des styles, c'est-à-dire un héritage multiple entre les origines (auteur, utilisateur), le média, la localisation des définitions de style (externe, en ligne, ...);

### 3.3 Syntaxe

Un style est défini par : `selecteur {prop1:val1; prop2:val2 val3 val4; prop3: v1, v2, v3; ...}`.

Le **sélecteur** définit l'ensemble des balises affectées par le style. Chacune des 53 **propriétés** est répartie dans un groupe parmi les 6 suivants :

**font** contenant font-family, font-size, font-weight, ...

**color, background** contenant color, background-color, background-repeat, background-image, ...

**text** contenant text-align, text-indent, ...

**box, layout** contenant padding, border, margin, width, height, display, float, ...

**list** contenant list-style-type, list-style-position, ...

**table** contenant caption-side, padding, border-collapse, ...

Les **valeurs** de propriétés sont réparties dans les catégories suivantes :

**mot-clé** non sensible à la classe tel que : red, underline, bold, top, collapse ...

**longueur** nombre décimal absolu ou relatif (démarrant par un signe) suivi par une unité sur 2 lettres. Les unités suivantes sont possibles :

**cm, mm** centimètre, millimètre;  
**in** inch (2.54cm);  
**pt** point (1/72 inch);  
**pc** pica (12pt ou 4.2mm);  
**px** pixel (dépend de la résolution de l'écran);  
**em** hauteur de la police courante;  
**ex** hauteur d'un x de la police courante;

Par exemple, +0.25mm, -3em, 100px, sont des longueurs correctes dont les deux premières indiquent un agrandissement et un rétrécissement par rapport à la valeur par défaut;

**pourcentage** longueur relative telle que `width :50%`, `line-height :120%`, ...

**couleur** exprimée en notation RGB (Red, Green, Blue) par :

**6 chiffres hexadécimaux** tels que `#FFFFFF` pour le blanc;

**3 chiffres hexadécimaux** tels que `#000` pour le noir; `#F00` pour le rouge (chaque chiffre est répété);

**rgb(128,0,128)** pour le pourpre (également `rgb(50%,0,50%)`);

**url** avec la notation `url(http://toto.fr/chemin)`;

### 3.4 Quelques exemples simples

#### Exemple 7 (styles dans l'en-tête)

Ce premier exemple utilise des styles définis dans l'en-tête html du document. Remarquons que pour `p`, la liste des polices est séparée par des virgules indiquant qu'un seul des éléments de la liste sera choisi. Pour la propriété `border` de `div` en revanche, la liste de valeur est une concaténation (sans virgule) de valeur (`color`, `width`, `style`).

```
<style type="text/css">
<!--
p {
  font-family : cursive, fantasy, monospace;
  line-height : 100%;
}
h1 {
  text-align: center;
  font-variant : small-caps;
}
div {
  float: left;
  width: 30%;
  border: blue 1px solid;
}
-->
</style>
</head>
<body>
...
```

Les commentaires html entourant la définition des styles est destinée aux navigateurs ne comprenant pas les styles CSS. Il est utile de valider sa feuille de style en faisant appel au validateur du W3C : <http://jigsaw.w3.org/css-validator/>. Les divisions flottantes permettent de placer les divisions dans le flot des boîtes affichées. Ici, trois divisions ( $3 \times 30% < 100%$ ) pourront être placées horizontalement dans la page.

#### Exemple 8 (positionnement fixe de l'en-tête)

Voici un ensemble de styles permettant de définir une division d'en-tête fixée en haut du navigateur et une division principale qui lorsqu'elle défilera, passera sous l'en-tête. La propriété `z-index` définit l'empilement des boîtes à l'affichage (par défaut 0).

```

<style type="text/css">
<!--
h1 {
  text-align: center;
  color: rgb(20,152,12);
}
div.entete {
  position: fixed;
  z-index: 1;
  background-color:aquamarine;
  top : 0px;
  height: 100px;
  width: 100%;
  border: red 1px solid
}
div.principale {
  position: absolute;
  top : 100px; // sous l'entete
  border: red 1px solid
}
div {
  float: left;
  border: blue 1px solid
}
-->
</style>
</head>
<body>

```



Dans le corps du document, on aura une structure en division telle que ce qui suit :

```

<div class="entete">
<h1>Titre de niveau 1</h1>
</div>

<div class="principale">

<div>
<p>hello World ! abcde abcde abcde abcde abcde
</div>

<div>
<p>hello World ! abcde abcde abcde abcde abcde
</div>
...
</div> <!-- principale -->

```

### 3.5 Localisation des styles

Les styles peuvent être définis de différentes façons et avec une priorité **décroissante** :

**en ligne** à éviter car le style est un attribut de balise et la séparation contenu et forme n'est plus assurée ;

**dans l'en-tête** comme dans le premier exemple, l'élément style peut être placé dans l'élément **head** du document html ; sa portée concerne **seulement** le document où il est défini ;

**dans un autre fichier** de style d'extension **.css** qui sera lié aux documents html par un élément **link**. C'est la meilleure méthode ! On peut également lier le document html à plusieurs fichiers de styles : en cas de conflit, c'est le dernier fichier lié qui est prépondérant ;

**Exemple 9 (un exemple de liaison à deux fichiers de style)**

```
<head>
<meta http-equiv="content-type" content="text/html; charset=ISO-8859-1" />
<title>Fichier CSS externe</title>
<link rel="stylesheet" type="text/css" href="css3.css">
<link rel="stylesheet" type="text/css" href="css32.css">
</head>
```

Supposons que dans le fichier `css3.css`, on a `h1 {color: red;}` et que dans le fichier `css32.css`, on a `h1 {color: blue;}`. Alors, les éléments `h1` seront bleu.

## 3.6 Sélecteur

Dans un style, le sélecteur définit les éléments affectés par les propriétés. Dans les exemples précédents les sélecteurs simples tels que `p` ou `div` ou `h1` permettent de désigner tous les éléments de cette catégorie. Mais on peut raffiner la sélection en utilisant de nombreux procédés.

### 3.6.1 Correspondance de modèle (pattern matching)

Avec CSS, des expressions régulières permettent de sélectionner certains éléments du DOM. Ces expressions régulières utilisent des opérandes (des éléments notés `E`, `F` tels que `div` ou `p` ou des sélecteurs complexes) et des opérateurs que nous allons décrire succinctement :

\* correspond à tous les éléments de n'importe quel type ;

`E F` correspond aux élément `F` descendant de `E` ; par exemple : `h1 a {color : red}` correspond aux liens situés dans un titre de niveau 1 qui seront en rouge ;

`E > F` correspond aux éléments `F` fils de `E` ; `ol > li {font-weight: bolder;}` met en gras les listes numérotées mais pas les autres (`ul`) ;

`E + F` correspond aux `F` qui sont un frère suivant d'un `E` ; par exemple : `h2 + p {color: yellow}` met en jaune le paragraphe qui suit un titre de niveau 2 ;

`E[nom="val"]` correspond aux éléments `E` ayant un attribut `nom` de valeur `val` ; par exemple :

`input[type="text"] {color: blue}` met les champs de saisie en bleu ; On peut ne pas mentionner la valeur afin de sélectionner tous les éléments ayant un attribut : `*[id]` sélectionne tous les éléments possédant l'attribut `id`.

### 3.6.2 Sélecteur multiple ,

Pour avoir tous les éléments de titre centrés :

```
h1, h2, h3 {text-align: center}
```

### 3.6.3 classes de style .

Une classe de style permet d'affecter différents styles à un même élément HTML selon le contexte où il est utilisé. Par exemple, un paragraphe de résumé pourra être écrit en italique alors que les paragraphes "normaux" ne le seront pas. Pour cela, il suffit de définir un attribut `class` de l'élément à styliser.

```
p {
  font-family : monospace;
}
p.resume {
  font-style : italic;
  line-height : 80%;
}
```

Remarquons que les paragraphes de résumé héritent de la famille de police monospace. Voici l'utilisation des deux styles.

```
<p class="resume">
  hello World ! abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
</p>
<p>abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde abcde
</p>
```

Cette façon de procéder est très fréquente pour définir des divisions sémantiques du document avec des `div` en leur affectant des classes différentes (voir l'exemple 8).

### Classe générique

On peut créer une classe sans l'associer à un élément html. Dans ce cas, le sélecteur est composé d'un point suivi du nom de classe. Cette classe peut être affectée à différents éléments :

```
.italique {font-style: italic;}
...
<p class="italique">bla bla </p>
<h4 class="italique">titre</h4>
```

### Identifiant d'élément `#`

On peut utiliser (avec `#`) l'attribut `id` d'un élément html pour lui affecter un style mais ce style ne pourra être affecté qu'à l'unique élément qui possèdera cet identifiant !

```
#valider { color : yellow;}
h1#premier { font-style : italic;}
```

### Pseudo-classe `:`

Les pseudo-classes et pseudo-éléments permettent de sélectionner des objets ou des classes qui ne sont pas des noeuds du DOM :

**pseudo-élément** la première ligne d'un paragraphe, ou un élément généré par la feuille de style ;

**pseudo-classe** classe acquise dynamiquement ou déduite d'un parcours du DOM ;

**:first-child** sélectionne seulement les paragraphes premiers fils : `p:first-child { text-indent: 0 }`

**:link** sélectionne les liens avant qu'ils aient été visités : `a:link { color: blue }`

**:visited** sélectionne les liens après qu'ils aient été visités : `a:visited { color: blue }`

**:focus** sélectionne l'élément ayant le focus : `input[type="text"]:focus {color: red}`. Durant la frappe du texte, celui-ci est rouge ;

**:hover** sélectionne l'élément ayant le pointeur dessus (souvent utilisé pour les liens) :

`input[type="text"]:hover {color: yellow}`. Durant le survol, le texte est jaune ;

**:active** sélectionne l'élément en train d'être activé (clic souris) : `input[type="text"]:active {color: black}`.

**:first-line** première ligne d'un paragraphe par exemple ;

**:first-letter** première lettre ;

**:before** pour générer du texte avant un certain élément :

```
body {
  counter-reset: chapter;      /* Crèe un compteur de chapitre */
}
h1:before {
  content: "Chapitre " counter(chapter) ". "; /* contenu généré */
  counter-increment: chapter; /* chapter++ */
  counter-reset: section;     /* Crèe un compteur de section */
}
h2:before {
  content: counter(chapter) "." counter(section) ". ";
  counter-increment: section;
}
```

Ce style permettra de précéder chaque titre de niveau 1 d’un “Chapitre 12. ” et chaque titre de niveau 2 d’un “Section 12.3.”.

**:after** pour générer un contenu (texte, image, ...) après un élément ;

```
body:after {
  content: "Fin du body";
  display: block;
  margin-top: 2em;
  text-align: center;
}
```

Dans cette exemple, le texte “Fin du body” sera affiché en fin de page.

A noter que les 4 derniers types de sélecteurs sont des pseudo-éléments et non des pseudo-classes et qu’ils doivent donc être à la fin d’un sélecteur.

## 3.7 Modèle de visualisation

Le flux des éléments du document est affiché selon un modèle de visualisation utilisant le principe des “boîtes” TeX. Les éléments peuvent être classés en deux familles :

- Les éléments de type block (h1, p, ul, ol, dl, table, blockquote, etc.);
- Les éléments de type inline (a, img, strong, abbr, etc.);

Un élément de type block se différencie des éléments de type en ligne sur différents points :

- Il occupe l’entiereté de la largeur de son conteneur ;
- Il permet l’attribution de marges verticales ;
- Il permet la modification de sa hauteur et largeur ;

Tout élément peut être “reclassé” dans la famille opposée grâce à la propriété `display`. Le calcul des dimensions et de la position de ces boîtes est dynamique au fur et à mesure du chargement de la page et/ou de la dynamique interactive.

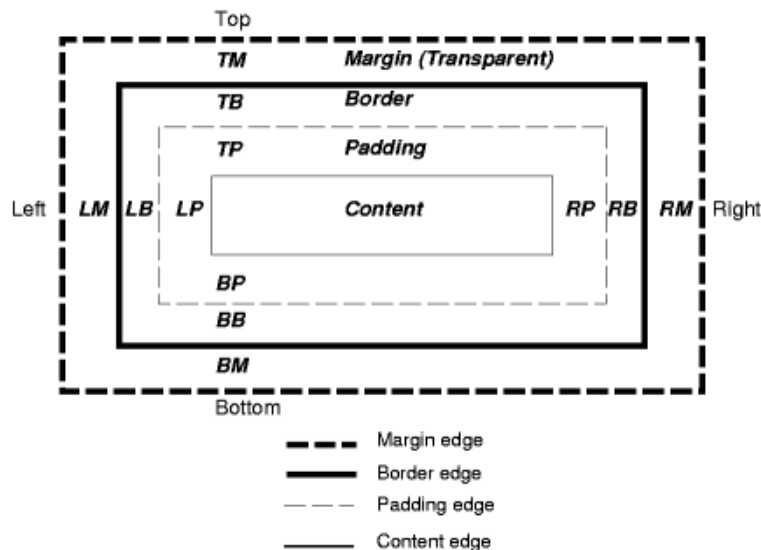


FIGURE 3.1 – une boîte et son entourage

On peut sortir un élément bloc du flux lorsqu’on envisage une mise en page un peu sophistiquée ...

### 3.7.1 La propriété css position

C’est la propriété vers laquelle l’on se doit de se tourner en premier lieu dès que l’on considère une mise en page pour laquelle le flux ne suffit plus. Une fois le type de `position` fixée, ce sont les propriétés `left`, `right`, `top`, `bottom` qui définiront les décalages.



### La position statique

La position statique (`position:static`) correspond simplement à la valeur par défaut d'un élément du flux. Il n'y a que peu d'intérêt à la préciser, si ce n'est dans le but de rétablir dans le flux un élément en particulier parmi une série qui serait positionnée hors du flux.

### La position fixe

Le positionnement fixe (`position:fixed`) positionne l'élément par rapport à la fenêtre du navigateur. L'élément est fixé à un endroit et ne pourra se mouvoir, même lors de la présence d'une barre de défilement. En d'autres termes, la position initiale est fixée au chargement de la page, le fait qu'une éventuelle scrollbar puisse être utilisée n'a aucune influence sur le positionnement de l'élément : il ne bouge plus de la position initialement définie.

### La position absolue

La position absolue (`position:absolute`) ressemble à la position fixe sauf qu'elle positionne l'élément par rapport à son premier ancêtre ayant une position autre que statique (`body` s'il n'en existe pas). `fixed`, `absolute` permettent la superposition. Un élément bénéficiant d'une position absolue ne bougera pas de sa position initiale tant que l'une des propriétés `top`, `bottom`, `left` ou `right` n'a pas été précisée ; il s'agit d'ailleurs là d'un comportement applicable à toutes les positions.

Dans le cas où un élément est en position absolue mais n'a pas de coordonnées (`left`, `top`, ...), il est placé à la suite comme s'il était dans le flux, mais ses successeurs qui sont dans le flux viennent l'écraser (superposition).

### La position relative

La position relative (`position:relative`) permet de décaler un élément par rapport à une position de référence : celle qu'il avait dans le flux. Les éléments qui le suivent et le précèdent ne sont pas influencés par ce décalage puisqu'ils considèrent que l'élément est toujours dans le flux à sa position initiale. Attribuer à un élément une position relative peut être pratique dans les situations suivantes :

- Servir de référent à un élément enfant positionné en absolu (rappelons qu'un élément positionné absolument grâce aux propriétés `top`, `left`, ... le fera par rapport à la fenêtre du navigateur à défaut d'avoir un parent lui-même positionné) ;
- Bénéficier de la possibilité d'utiliser la propriété `z-index` pour gérer des superpositions d'éléments (propriété inopérante pour des éléments du flux) ;

## 3.7.2 La propriété float

La propriété `float` existe avant tout pour répondre à un besoin typographique précis : la création d'habillages. Un habillage est une pratique courante dans le média print consistant à "enrouler" un texte autour d'un élément (graphique ou texte).

À l'instar du positionnement absolu, un élément flottant adopte par défaut la largeur qu'occupe son contenu. Le principe de base est simple : un élément flottant est ôté partiellement du flux et placé à l'extrême gauche (`float:left`) ou droite (`float:right`) de son conteneur, forçant par la même occasion tout contenu du flux qui suit à l'envelopper. Deux objets flottants dans la même direction se rangeront côte à côte, seul un contenu demeuré dans le flux qui les succède *immédiatement* initiera l'habillage.

## 3.8 Type de media et résolution d'écran

Depuis CSS2, on peut spécifier une feuille de style différente selon le media d'affichage de la page web :

```
<link rel="stylesheet" media="screen" href="screen.css" type="text/css" />
<link rel="stylesheet" media="print" href="print.css" type="text/css" />
```

On peut également spécifier des propriétés à l'intérieur d'une même feuille :

```
@media print {
  #menu, #footer, aside {
    display:none;
  }
  body {
    font-size:120%;
  }
}
```

```

    color:black;
  }
}

```

Les différents media sont : screen, handheld (mobiles), print, aural (Synthèse vocale), braille, projection, tty (police à pas fixe), tv, all.

Afin de prendre en compte, les résolutions d'écran différentes (moniteur, tablette, smartphone), CSS3 a introduit les *media queries* qui permettent de spécifier un style conditionnel en fonction de la taille de la fenêtre. Un exemple suit :

```

.precisions span {
display: none; // par défaut n'apparait pas
}
@media screen and (min-width: 1024px) and (max-width: 1280px) {
.wide {
background: #f11a57; // change couleur et fond
color:#fff;
}
.precisions span.regle3 {
display: block; // sauf si grand écran
}
}

```

D'autres éléments de CSS3 telles que les grilles fluides dépassent l'objectif de ce cours!

### 3.9 Responsive Web Design

Les tailles d'écrans de visualisation se démultipliant (mobile, tablette, TV, ...), une conception adaptée consiste à prendre en compte ces différents supports afin de permettre à chaque utilisateur de pouvoir utiliser le site web. Une solution simple consiste à définir son site pour l'écran le moins disant au niveau de sa définition, par exemple en définissant de manière fixe les éléments importants de votre blog :

```

article {
width: 300px;
height: 600px;
}

```

Une autre solution, bien meilleure, consiste à utiliser un framework CSS tel que bootstrap qui redéfinit le css en fonction de la largeur de l'écran grâce à l'élément méta à insérer dans le l'entête (head) de la page :

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

# Chapitre 4

## Le framework CSS Bootstrap

### 4.1 Introduction

Bootstrap est un framework CSS donc côté client (*frontend*) créé pour Twitter. Il embarque également des composants HTML, JavaScript, JQuery. Il comporte un système de grille simple pour organiser l'aspect visuel d'une page web. Il apporte du style pour les boutons, les formulaires, la navigation ... Il permet ainsi de concevoir un site web rapidement et avec peu de lignes de code. Il utilise le "responsive design" qui consiste à adapter la page Web au média utilisé.

### 4.2 Avantages de l'utilisation d'un framework

- les navigateurs ont des comportements très différents malgré leur lente convergence vers les standards. Les frameworks assurent une présentation similaire quel que soit le navigateur utilisé et une parfaite compatibilité;
- bootstrap assure une accessibilité pour les utilisateurs handicapés;
- les frameworks CSS font gagner du temps de développement **une fois qu'on les maîtrise**;
- ils proposent un ensemble homogène de styles;
- ils proposent en général une grille pour faciliter le positionnement des éléments;
- ils offrent souvent des éléments complémentaires : boutons esthétiques, fil d'ariane, etc...
- la grande variété des nouveaux moyens de visualisation du web (smartphones, tablettes...) impose désormais la prise en compte de tailles d'écran très variées; les frameworks CSS prennent généralement en compte cette contrainte.

Inconvénients :

- temps d'apprentissage;
- version 3 non compatible avec version 2!

### 4.3 Caractéristiques

- système de grille de 12 colonnes;
- sous licence Apache, actuellement à la version 3 orientée mobile;
- du code fondé sur HTML5 et CSS3;
- une bonne documentation;
- une architecture basée sur LESS, un outil qui étend les possibilités de CSS;

### 4.4 Installation

- se rendre à l'url `http://getbootstrap.com`;
- télécharger bootstrap sous forme d'un fichier zip contenant le code css compilé et minimisé ainsi que du javascript et des polices;
- décompresser l'archive à la racine du site et renommer le répertoire en `bootstrap` (supprimer le numéro de version);
- insérer dans l'entête (*head*) des pages html ou php conforme au langage HTML5 le code suivant :

```
<link href="bootstrap/css/bootstrap.min.css" rel="stylesheet">
<script src="bootstrap/js/jquery.js"></script>
```

```
<script src="bootstrap/js/bootstrap.min.js"></script>
```

*Il faudra bien entendu télécharger le le fichier `jquery.js` dans le répertoire concerné!*

#### 4.4.1 Remarques

Les fichiers minimisés (`.min.cs` ou `.min.js`) sont des versions compressées illisibles dans lesquelles les commentaires ont été supprimés. Ils sont destinés à être chargés plus rapidement en mode production. En mode développement, il est conseillé d'utiliser les versions non minimisées.

Les CDN (Content delivery network) sont des serveurs qui mettent à disposition des bibliothèques. Il devient ainsi inutile de stocker ces bibliothèques sur son propre serveur, il suffit de "pointer" vers eux. L'avantage à utiliser un CDN est de diminuer l'utilisation de ressources (stockage et bande passante) sur son propre serveur mais surtout de factoriser ces ressources dans le cache du navigateur ce qui accélère le chargement des pages.

### 4.5 Premier site

Bootstrap propose sur son site des exemples de dispositions de pages. Le site <http://www.bootply.com/> propose même un éditeur graphique en ligne afin de prototyper l'apparence de son projet. Mais voyons les concepts essentiels.

### 4.6 La grille

Une grille est tout simplement un découpage en cellules de **même largeur** de la page Web. Les différents éléments de la page (bannière, barre de menu, ...) seront disposés sur un nombre donné de ces cellules. La hauteur d'une ligne sera déterminée par la hauteur de son plus gros élément. Le nombre maximum de colonnes est de 12.

4 types d'écrans sont définis par les tailles (de vêtements!) suivantes :

**xs** largeur < 768 px pour les mobiles;

**sm** 768 px <= largeur < 992 px pour les tablettes;

**md** 992 px <= largeur < 1200 px pour ordinateurs portables;

**lg** 1200 px <= largeur pour les grands écrans;

On va définir une mise en page de la grille spécifique pour un type d'écran, par exemple **md**. Si l'utilisateur possède un écran **md** ou **lg**, la disposition prévue sera respectée. Par contre, sur les écrans plus petits, un empilement vertical des éléments aura lieu afin de permettre leur lisibilité sur des mobiles par exemple. On peut combiner différentes dispositions pour plusieurs types d'écran pour chaque élément afin de prévoir différentes disposition selon le type d'écran.

#### Exemple 10

3 dispositions distinctes de blocs

##### 1. Le code source HTML

```
<body>
<div class="container">
<h3>Responsive design</h3>
<p>combinaisons de dispositions : xs pour les mobiles et étendu aux tablettes,
  md pour les écrans de portables et d'ordinateurs de bureau.
<div class="row">
  <div class="col-xs-12 col-md-8">(col-xs-12 col-md-8) plein sur mobile, 2/3
    sur écran</div>
  <div class="col-xs-6 col-md-4">(col-xs-6 col-md-4) moitié sur mobile, 1/3
    sur écran</div>
</div>
<div class="row">
  <div class="col-xs-6 col-md-4">(col-xs-6 col-md-4) moitié sur mobile, 1/3
    sur écran</div>
  <div class="col-xs-6 col-md-4">(col-xs-6 col-md-4) moitié sur mobile, 1/3
    sur écran</div>
  <div class="col-xs-6 col-md-4">(col-xs-6 col-md-4) moitié sur mobile, 1/3
    sur écran</div>
</div>
```

```

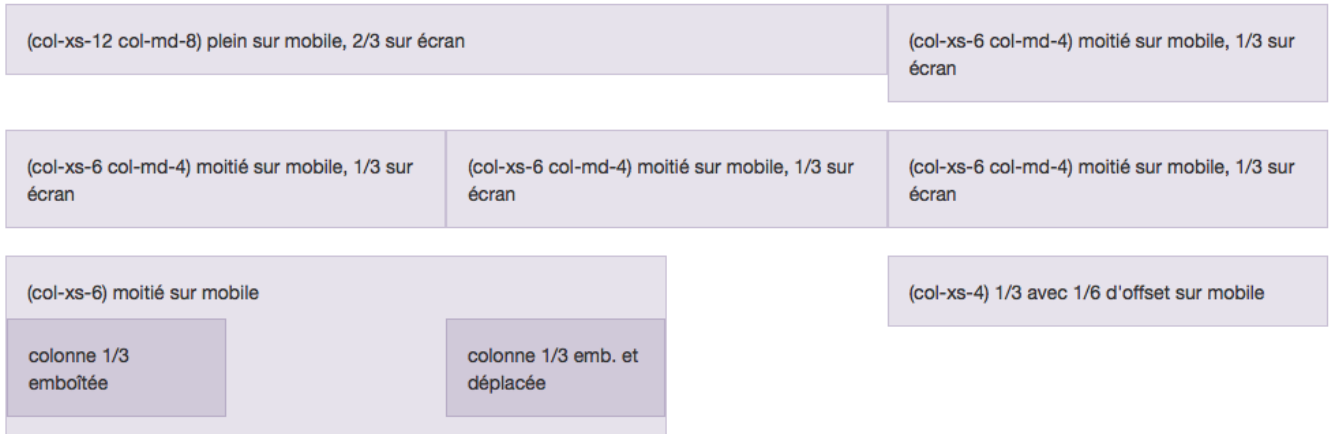
<div class="row">
  <div class="col-xs-6">(col-xs-6) moitié sur mobile</div>
  <div class="col-xs-4 col-xs-offset-2">(col-xs-4) 1/3 avec 1/6 d'offset
    sur mobile</div>
</div>

```

## 2. Affichage sur grand écran

### Responsive design

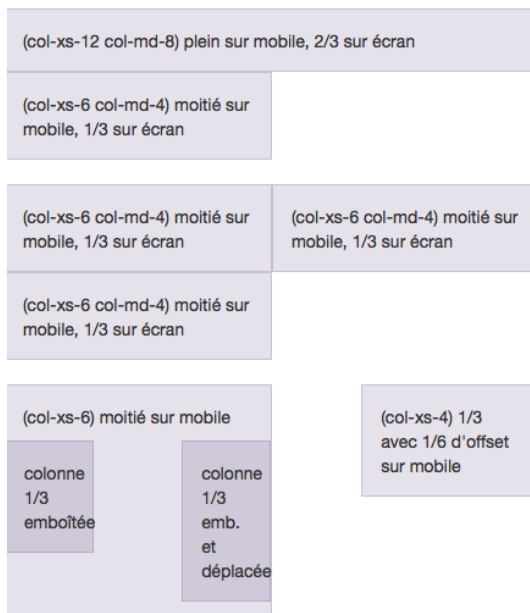
combinaisons de dispositions : xs pour les mobiles et étendu aux tablettes, md pour les écrans de portables et d'ordinateurs de bureau.



## 3. Affichage sur mobile

### Responsive design

combinaisons de dispositions : xs pour les mobiles et étendu aux tablettes, md pour les écrans de portables et d'ordinateurs de bureau.



Bien entendu, le principe de la grille de 12 est récursif, ce qui fait qu'on peut emboîter des blocs à l'intérieur ...

## 4.7 Eléments de base

Afin de créer un rendu visuel cohérent pour que tous les éléments cohabitent de façon esthétique, Bootstrap impose un certain nombre de principes :

- Tous les éléments sont englobés dans une racine `<div class="container">` fille de `body` ; Pour les applications (sans marges latérales) la classe `container-fluid` est plus adaptée ;
- l'emboîtement (row in col), le déplacement (offset), le changement d'ordre des colonnes (push) sont possibles ;
- des titres de h1 à h6 sont disponibles avec un sous-titre en ligne grâce à la classe `small` ;
- la taille de police par défaut du `body` est de 14px, la hauteur de ligne de 1,428 soit près de 20px ;
- diverses balises sont stylées pour les listes `ul`, `ol`, `dl` (description list=dictionnaire), les abbréviations (`abbr`), les citations (`blockquote`), les alignements gauche droite justifié,
- les tables html possèdent différentes classes intéressantes :

**.table-bordered** une bordure simple encadre chaque cellule ;

**.table-hover** la ligne est surlignée lorsque la souris survole ;

**.table-responsive** emboîter une table dans une table-responsive ajoutera à ascenseur horizontal sur les petits écrans ;

- les formulaires sont divisés en plusieurs classes :

**par défaut** les contrôles (`input`, `select`, ...) ont une largeur de 100% (formulaires destinés au téléphones mobiles) ; `label` et `input` sont emboîtés verticalement dans une `div` de classe `form-group` ;

**form-inline** permet de placer les labels et leur contrôle en ligne ;

**form-horizontal** permet d'utiliser le principe de la grille afin de placer le label dans la colonne de gauche, et le contrôle dans la colonne de droite ; en cas d'utilisation avec petit écran, l'empilement reprend le dessus et l'aspect redevient celui de la classe par défaut.

Les différents contrôles HTML doivent posséder la class `form-control` et doivent être emboîtés dans une `div` de class `form-group`. De plus, une autre classe peut être ajoutée (en javascript ou PHP) pour indiquer la qualité de la validation des données saisies : `has-success`, `has-warning`, `has-error`. De nombreux types de boutons existent (forme, couleur) et l'on peut représenter différents éléments html tels que les liens (`a href`), les `button`, les `input` de type `button` et les `input` de type `submit` avec la même représentation graphique !

- Des menus dropdown, des groupes de boutons, ...
- Une navigation est une liste non numérotée (`ul`) possédant la classe `nav` et une sous-classe précisant la visualisation des `li` : `nav-tabs` pour des onglets, `nav-pills` pour des boutons accolés. Chaque item de liste peut contenir un simple lien ou un un dropdown-toggle permettant de basculer un menu ;
- Une barre de navigation est un bloc `<nav>` possédant la classe `navbar` et offre un widget réactif (responsive) pour réaliser le menu principal d'un site. Il peut réunir un logo, des dropdown, un formulaire de recherche, un autre de connexion, ...
- un fil d'ariane (breadcrumb) affiche la hiérarchie de navigation en séparant les liens par des slash. Il est réalisé avec une liste ordonnée (`ol`) possédant la classe `breadcrumb`.

## Eléments textuels

```

texte <mark>surligné</mark> grâce à la balise mark
texte <del>supprimé</del> grâce à la balise del
texte <u>souligné</u> grâce à la balise u
texte <small>petit</small> grâce à la balise small
texte <strong>en gras</strong> grâce à la balise strong
texte <em>en emphase</em> grâce à la balise em
texte de programme : <code> if (i<5*j) printf(... </code>
taper <kbd> cd ~</kbd> pour vous rendre sur votre répertoire d'accueil
<pre> pour plusieurs lignes de code

```

```

texte surligné grâce à la balise mark
texte supprimé grâce à la balise del
texte souligné grâce à la balise u
texte petit grâce à la balise small
texte en gras grâce à la balise strong
texte en emphase grâce à la balise em
texte de programme : if (i<5*j) printf(...)
taper cd ~ pour vous rendre sur votre répertoire d'accueil

```

## Responsive design

combinaisons de dispositions : xs pour les mobiles et étendu aux tablettes, md pour les écrans de portables et d'ordinateurs de bureau.

(col-xs-12 col-md-8) plein sur mobile, 2/3 sur écran

(col-xs-6 col-md-4) moitié sur mobile, 1/3 sur écran

(col-xs-6 col-md-4) moitié sur mobile, 1/3 sur écran

(col-xs-6 col-md-4) moitié sur mobile, 1/3 sur écran

(col-xs-6 col-md-4) moitié sur mobile, 1/3 sur écran

(col-xs-6) moitié sur mobile

colonne 1/3  
emboîtée

colonne 1/3 emb. et  
déplacée

(col-xs-4) 1/3 avec 1/6 d'offset sur mobile





# Chapitre 5

## Le langage PHP

### 5.1 Introduction

PHP (acronyme récursif pour PHP : Hypertext Preprocessor) est un langage de script (interprété) et Open Source, spécialement conçu pour le développement d'applications web. Il doit être **intégré** au HTML. La référence du langage est sur un site Web [5] qu'il faut toujours avoir en face des yeux quand on programme !

#### Exemple 12

```
<html><head><title>premier prg php</title></head><body>
<?php
    echo "Bonjour le monde !";
?>
</body></html>
```

Remarques :

- différence avec les autres scripts CGI (Perl, C, ... ) : page HTML avec du code inclus à l'intérieur :
  - pas besoin de générer le code HTML ;
  - prototypage du squelette du site par des spécialistes graphiques ;
- différence avec Javascript : le code est exécuté sur le serveur ;
- langage extrêmement simple et fonctionnalités avancées ;
- l'interprète PHP peut être appelé par :
  - le serveur Web comme module ou comme CGI ;
  - l'interpréteur de commandes de votre système : `php hello.php`.
  - en mode **interactif** (`php -a`), une session d'interprétation s'ouvre et vous pouvez tester vos instructions !

### 5.2 Caractéristiques

Différentes versions sont apparues au cours du temps (PHP2FI, PHP3, PH4, PHP5). Nous ne traiterons ici que de **PHP5**.

L'interprète PHP fonctionne sur le fichier interprété comme un filtre laissant passer de l'entrée standard vers la sortie standard tout ce qui n'est pas entre `<?php` et `?>`. Quant au reste, il exécute les instructions, fonctions, définitions incluses. Dans le cas où plusieurs *blocs* php existent, les définitions du premier bloc sont utilisables dans tous les autres.

#### Exemple 13

```
<html><head><title>deuxieme prg php</title></head><body>
<?php
$i=123;
?>
bla bla bla
<?php
echo "la variable i vaut : $i";
?>
</body></html>
```

Le fichier d'extension `.php` doit être lisible par le serveur HTTP (`chmod a+r hello.php`). Le droit d'exécution n'est pas nécessaire.

## 5.3 Structure du langage

Au niveau syntaxique, le langage PHP ressemble fortement au langage C à ceci près qu'il n'est pas typé! Toutes les variables doivent être **préfixées par un \$** : c'est la cause principale d'erreur syntaxique. Il faut donc répéter que TOUTES LES VARIABLES DOIVENT ÊTRE PRÉFIXÉES PAR UN \$!

### 5.3.1 Types

Les types scalaires sont :

- integer avec des littéraux tels que : 10, -454, 0777, 0xABCD;
- float avec des littéraux tels que : 1.24, -1.2e3, 5E-6;
- string avec des littéraux tels que : 'hello world', "hello \$i world"; Entre guillemets, les variables scalaires sont substituées par leur valeur (pas entre apostrophes)! De nombreuses fonctions strxxx existent!
- boolean : true et false (insensibles à la casse);
- null : la valeur spéciale NULL représente l'absence de valeur (insensible à la casse).

### 5.3.2 Typage dynamique

Une variable n'est pas typée statiquement mais dynamiquement au fur et à mesure de ses affectations successives. La conversion de type est automatique et dépend du contexte.

#### Exemple 14

```
<?php
$x = "0"; // $x est une chaîne de caractères (ASCII 48)
$x += 2; // $x est maintenant du type entier (2)
$x = $x + 1.3; // $x est maintenant du type double (3.3)
$x = 5 + "10 roses"; // $x est du type entier (15)
?>
```

On peut forcer le type d'une variable avec :

- la fonction `settype()`;
- ou le transtypage à la C : `$x = (int) 5.6;`

On peut tester le type d'une variable avec les fonctions booléennes suivantes : `bool is_int(mixed $var)`, `bool is_string(mixed $var)`, ...

### 5.3.3 Existence et valeur

La fonction `isset($x)` permet de tester l'existence d'une variable ce qui s'avère primordial dans le cadre de la soumission de formulaire HTML pour savoir si un paramètre a été posté.

Pour "tuer" une variable, il faut utiliser la fonction `unset($x)`.

Attention, il faut distinguer `isnull($x)` qui teste l'égalité de `$x` avec la valeur NULL.

### 5.3.4 Constantes

Comme en C, les constantes sont des macros, elles sont définies par la fonction `define()` et utilisées **sans \$**!

```
define("REP_TELECHARGEMENT", "Telechargement/");
define("TAILLE_MAXI_FICHER", 1000000);
echo TAILLE_MAXI_FICHER;
```

### 5.3.5 Le type tableau associatif

En PHP, le type tableau associatif est constamment utilisé, il faut donc bien comprendre son fonctionnement. Un tableau PHP est une association (Map) de couples (clé, valeur). Les clés sont uniques et sont des entiers positifs ou des chaînes. Les valeurs sont quelconques (type et valeur). Si l'on a besoin d'un tableau à la C indicé par des entiers débutant à 0, il faut considérer ces indices comme des clés.

Lorsqu'on ajoute des valeurs sans clé dans le tableau, la clé est calculée comme un entier immédiatement supérieur à la plus grande clé entière du tableau! **Attention**, l'ordre chronologique des ajouts est conservée lorsqu'on parcourt (`foreach`) le tableau.

**Exemple 15**

```
<html><head><title>tableaux php</title></head><body>
<?php
$tab=array("un"=>123, 2=>"deux", "abc"=>12.3);
$tab[]="toto";$tab[1]="un";
echo "taille tab : ".count($tab)."<br/>\n";
foreach($tab as $c=>$v){echo "$c => $v <br/>\n";}
?>
</body></html>
```

affiche :

```
<html><head><title>tableaux php</title></head><body>
taille tab : 5<br/>
un => 123 <br/>
2 => deux <br/>
abc => 12.3 <br/>
3 => toto <br/>
1 => un <br/>
</body></html>
```

Quelques fonctions utiles :

count(\$tab)	retourne le nombre de couples
unset(\$tab['deux'])	supprime la clé et la valeur
isset(\$tab['deux'])	teste si défini
empty(\$_POST['valider'])	teste si non défini ou vide ou 0
\$tab[]='nouveau'	ajoute un couple à la fin
print_r(\$tab)	affiche récursivement le tableau
\$compact=array_values(\$tab)	compacte tab dans nouveau indicé de 0 à n-1
foreach(\$tab as \$cle=>\$valeur){...}	parcours du tableau
bool in_array ("toto",\$tab)	recherche de valeur
\$t=array(array(1,"toto"=>3),array("tutu",1=>2))	multi-dimensionnel
bool array_xxx(\$tab)	innombrables fonctions avec xxx valant sort, merge, diff, ...

TABLE 5.1 – fonctions utiles sur les tableaux

Avec une syntaxe utilisant des accolades, on peut substituer des éléments de tableaux dans une chaîne littérale :  
 echo "bonjour {\$tab['michel'][5]}";

### 5.3.6 Les expressions

#### Opérateurs

Très utilisés avec tous les types, ils ont une priorité (“precedence”) et une associativité. Voici une liste non exhaustive avec priorités décroissantes :

++, -- inc et déc : ++\$x;\$y--;

**unaires** ~ (not binaire), - (moins arithmétique), ! (not logique);

**arithmét. et concat.** (\*,/,%) puis (+, -, .) (point de concaténation);

**décalages** (<<,>>)

**comparaison** (<,<=,>=,>) puis (==, !=, ===, !==) (comparaison typée). ATTENTION : fonctionne avec les chaînes mais attention aux chaînes numériques : "1"==" 01.0" mais "1"!==" 01.0"

**bit à bit** (&, ^, |)

**logiques** (&&, ||)

**\$i==\$j? 1 : 2** expression conditionnelle

**affectation** (=, .=, +=, -=, \*=, /=, %=)

**logiques** (and, xor, or) moins prioritaire que les affectations, utilisé après les appels risqués : \$x=danger() or die("erreur

Remarquons qu'il existe 2 types de comparaisons == et ===". Ce dernier n'est vrai que si les deux expressions sont du même type et sont égales. Cela peut être utile pour éviter les égalités dues à des conversions.

### Exemple 16

```
<html><head><title>comparaison</title></head><body>
<?php
if (0==" && null==false && $tab==0 && "2"=="02"){
    echo "BIZARRE";
}
?>
</body></html>
    affiche : BIZARRE
```

La fonction `strpos($chaine, $facteur)` retourne `false` si le facteur est absent, la position comprise entre 0 et `strlen($chaine)-1` si le facteur est trouvé. Le test `if(strpos('abc','ab')){...}` est incorrect car `false==0` ! Il faut donc absolument faire le test suivant : `if (strpos('abc','ab') !== false) {...}`

### Particularités

Certaines particularités historiques sont encore présentes :

**\$\$s** valeur de la variable dont le nom est dans `s` ; on ne peut pas tripler le `$` !

**eval("...")** évaluation d'une chaîne : TRES dangereux !

**commande syst.** 'ls'

L'affect. de chaîne longue (heredoc) est possible :

```
$s<<<FIN
bla bla..
etc
FIN; // en début de ligne et avec un '',''
```

### 5.3.7 Structures de contrôle

Les coupures de lignes sont traitées comme les espaces. Les structures de contrôle classique à la C existent en PHP :

**alternative** `if (expr) {instons1} else {instons2}`

**choix** `if (e1) {ins1} elseif (e2) {ins2} else {ins3}`

**choix** `switch ($i) {case e0: instons0; break; case e1: instons1; break; default: instons; }`

**répétitives** — `while (e) {instons}`

— `do { instons } while (e)`

— `for (exp-init; exp-cond; exp-itér) { instons }`

— ruptures possibles : `break;`, `continue;`

**parcours de tableau** — `foreach($tab as $val) { instons utilisant $val}`

— `foreach($tab as $cle => $val) { instons utilisant $cle et/ou $val }`

**inclusion** `include 'monfic.php'`; inclusion de fichier

**inclusion unique** `include_once 'monfic.php'`; inclusion au plus une fois!

**require, require\_once** même effet que `include` sauf qu'en cas d'inexistence du fichier requis, une erreur fatale est envoyée (seulement `warning` pour `include`)

### 5.3.8 Fonctions

Un nombre impressionnant de fonctions prédéfinies ! Passage des paramètres scalaires ou tableaux par valeur (par référence si précédé de `&` dans l'appel et la déclaration). Les variables globales ne sont pas accessibles directement. Il faut utiliser la notation : `$GLOBALS['glob']`. `$GLOBALS` est l'asso. des variables globales. On peut aussi déclarer les variables globales dans la fon par `global $i,$j`; Les arguments peuvent avoir une valeur par défaut (optionnels) et être en nombre variable. Les arguments ayant une valeur par défaut doivent être à droite de ceux n'en ayant pas. Les valeurs de retour peuvent être scalaire ou tableaux. On peut supprimer l'affichage des erreurs produites par l'appel à une fonction `f` par : `@f($i)`

**Fonctions sur les tableaux**

**int** count(\$tab) taille du tableau  
**int** array\_push(\$pile,\$elem) empile à la fin et ret la taille  
**mixed** array\_pop(\$pile) dépile le sommet  
**int** array\_unshift(\$fifo,\$prem) ajoute au début  
**mixed** array\_shift(\$fifo) retire le premier  
**array** array\_values(\$asso) resp. array\_keys  
**void** shuffle(\$tab) mélange les valeurs

**Fonctions sur les tris de tableau**

**void** sort(\$tab) tri croissant (décroissant avec rsort) selon les valeurs  
**void** ksort(\$asso) tri croissant selon les clés (ksort)  
**void** asort(\$asso) tri croissant selon les valeurs (arsort)  
**void** usort(\$tab, moncmp) tri croissant selon la fon de cmp définie par l'utilisateur  
**uksort, uasort** tri utilisateur pour les asso

**Fonctions sur les chaînes**

De très nombreuses fonctions dont voici les principales. \$s est supposée être une chaîne.

**int** strlen(\$s) taille  
**int** strcmp(\$s1,\$s2) comparaison -1, 0, 1  
**string** substr(\$s, 2, 10) sous-chaîne à partir de 2, de taille 10  
**string** strstr(\$s,\$facteur) ret tout s à partir de la 1ère occurrence de facteur  
**array** split(';', \$s, 10) ret un tableau des 10 (maxi) premiers champs séparés par des ;  
**string** join(';', \$tab) ret la chaîne constituée des valeurs séparées par ;  
**string** trim(\$s) retire les blancs de début et de fin de chaîne. Blancs : \n, \r, \t, \v, \0, espace  
**string** chop(\$s) supprime les blancs de fin de chaîne  
**string** ltrim (\$s) supprime les blancs de début  
**string** strtolower(\$s) met en minuscules (resp. strtoupper)  
**string** nl2br(\$s) remplace les \n par des <br>

**Fonctions sur les expressions régulières**

**correspondance** **int** ereg( '^[^.]\*(.\*)\$', \$s, \$tabr) met dans tabr la partie à droite d'un point dans s à partir de l'indice 1. Retourne TRUE si trouvé au moins une.  
**remplacement** **string** ereg\_replace("[a-z]+", "[\1]", \$s); encadre les mots minuscules de s

**Entrées/Sorties : système de fichier**

De nombreuses fonctions de gestion du système de fichier à la C.

**echo string, string ...**; affiche plusieurs arguments **chaînes**  
**print(string)**; affiche un arg. (**classé** dans les fons chaînes)  
**\$desc=fopen("fic.txt", "r+");** ouverture en lecture et écriture. Modes (r, w (création ou raz), w+, a (append), a+).  
**\$d=fopen("c : \\data\\info.txt", "r");** Windows  
**\$d=fopen("ftp ://user :password@example.com/", "w");** ouverture de session ftp  
**fclose(\$desc)** fermeture  
**bool** feof(\$desc) test la fin  
**\$ligne=fgets(\$desc, 4096)** lecture de la ligne (maxi 4096 octets)  
**\$ligne=readline("un entier SVP");** ne fonctionne pas sur le Web!  
**\$car=fgetc(\$desc);** lecture d'un car;  
**fputs(\$desc, "chaine");** écriture d'une chaîne;

## Fonctions utilisateur

La référence en avant est possible.

```
définition function f($arg0, $arg1="toto"){instons; return array(1, 4);}
pointeur de fonction $pf='f';list($i,$j)=$pf(1,2);
```

## Diverses fonctions prédéfinies

**echo** exp1, exp2, ... ; affiche les expressions ;

**print(exp1)** affiche une expression ;

**print\_r(\$var)** affiche la variable scalaire ou tableau ou objet récursivement ;

**void exit()** termine le script

**\$l=system("ls")** ; exécute une commande

**void die ("message ")** affiche le message puis termine

**void eval("instons PHP")** il faut échapper les caractères spéciaux : `eval('$' . f($i) . '= "toto";'` ; Ne pas oublier le “;”

### 5.3.9 Tableaux super-globaux

Des tableaux super-globaux sont prédéfinis et permettent d'accéder à diverses informations.

**\$GLOBALS** Contient une référence sur chaque variable qui est en fait disponible dans l'environnement d'exécution global. Les clés de ce tableau sont les noms des variables globales définies par `global $x`; Ainsi, `$GLOBALS['x']` permet de connaître ou d'affecter la variable globale `$x`.

**\$\_SERVER** Les variables fournies par le serveur web, ou bien directement liées à l'environnement d'exécution du script courant, notamment `$_SERVER['PHP_SELF']` qui est le chemin du script en cours d'exécution par rapport à la racine web (sans les paramètres GET) ;

**\$\_GET** Les variables fournies au script via la chaîne de requête URL.

**\$\_POST** Les variables fournies par le protocole HTTP en méthode POST.

**\$\_COOKIE** Les variables fournies par le protocole HTTP, dans les cookies.

**\$\_FILES** Les variables fournies par le protocole HTTP, suite à un téléchargement de fichier.

**\$\_ENV** Les variables fournies par l'environnement

**\$\_REQUEST** Les variables fournies au script par n'importe quel mécanisme d'entrée et qui ne doivent recevoir qu'une confiance limitée. Note : lorsque vous exécutez un script en ligne de commande, cette variable ne va pas inclure les variables `argv` et `argc`. Elles seront présentes dans la variable `$_SERVER`.

## 5.4 Gestion de formulaire

Les champs de **formulaire HTML** sont accessibles via les tableaux superglobaux `$_GET` ou `$_POST` selon la méthode utilisée par le formulaire. Par exemple :

```
<form name='F' method='get' action='<?php echo $_SERVER['PHP_SELF']; ?>'>
<input type='text' name='nom'>
<select multiple name="biere[]">
  <option value="blonde">blonde</option>
  <option value="brune">brune</option>
</select>
<input type='submit' name='bouton' value='ok' />
</form>
<?php print_r($_GET['biere']); ?>
```

Après saisie et validation, l'url suivante est requise :

```
essai.php?nom=toto&biere[]=blonde&biere[]=brune&bouton=ok
```

L'affichage suivant aura lieu :

```
Array ( [0] => blonde [1] => brune )
```

**Exercice 1** Ecrire :

- une page HTML “Multiplication” contenant un formulaire ayant deux champs de texte  $x$  et  $y$  et un bouton **Multiplier** !;
- un script php appelé par ce formulaire et qui affiche le résultat de la multiplication  $x * y$ ;

**Exercice 2** Réécrire le même exercice en un seul fichier !

## 5.5 Le modèle objet depuis PHP5

### 5.5.1 Résumé

Depuis PHP 5, le modèle objet a été réécrit. Il est proche de celui du C++. Une classe peut contenir ses propres constantes, variables (appelées "propriétés" ou "attributs"), et fonctions (appelées "méthodes"). Une classe est introduite par le mot-clé `class` suivi de son nom. Le constructeur se nomme `__construct()` et le destructeur `__destruct()`. Par défaut, l'appel au constructeur de la classe parente n'est pas fait ; pour le faire : `(parent::__construct())`. Les méthodes magiques `__toString()` et `__clone()` sont également très utiles. L'affectation et le passage de paramètre objet est effectué **par référence** : il n'y a pas de duplication de l'objet mais la variable affectée ou le paramètre formel contient l'adresse de l'objet.

### 5.5.2 Exemple

```
class Point{
    public static $nbPoints=0;
    public $x,$y;
    public function __construct($px=0,$py=0){ // constructeur
        $this->x=$px;$this->y=$py;
        self::$nbPoints+=1;
    }
    public function getxy(){
        return array($this->x, $this->y);
    }
}
$p1=new Point(); // (0,0)
$p2=new Point(3,4);
list($a,$b)=$p2->getxy();
```

### 5.5.3 Propriétés

Les attributs d'instance ou propriétés doivent être déclarés à l'aide de `public`, `protected` ou `private`. L'ancien déclarateur `var` est obsolète et est remplacé par `public`. Une déclaration initialisante est permise en utilisant une valeur constante. Dans les méthodes d'instance, ces propriétés sont accessibles via la notation “fléchée” `$this->x`.

Une propriété statique (attribut de classe) peut être définie grâce au mot-clé `static`. L'accès à cette propriété à l'intérieur d'une classe se fait par `self::$nbPoints`.

### 5.5.4 Constructeur et destructeur

Les constructeurs d'une classe se nomment `__construct(...)` et le destructeur `__destruct()`. Par défaut, l'appel au constructeur de la classe parente n'est pas fait ; pour le faire : `(parent::__construct())`. En l'absence de constructeur explicite, toute classe admet un constructeur par défaut qui ne fait rien.

### 5.5.5 Méthodes

Les méthodes des classes peuvent être définies en tant que publiques, privées ou protégées. Les méthodes sans déclaration seront automatiquement définies comme étant publiques.

### 5.5.6 Héritage (extends)

Une classe peut hériter des méthodes et des membres d'une autre classe en utilisant le mot-clé `extends` dans la déclaration. Il n'y a pas d'héritage multiple.

Les méthodes et attributs hérités peuvent être redéfinis en les redéclarant avec le même nom que dans la classe parente (sauf si la classe parente a défini la méthode comme `final`). Il est possible d'accéder à une méthode ou un membre parent avec l'opérateur `parent::`.

Lors de la redéfinition de méthodes, la signature doit rester la même sinon PHP génèrera une erreur de niveau `E_STRICT` (**pas de surcharge**). Ceci ne s'applique pas au constructeur, qui accepte la surcharge (avec des paramètres différents).

### 5.5.7 Interfaces

Une interface permet de spécifier quelles méthodes une classe doit implémenter. Les interfaces sont définies en utilisant le mot-clé `interface`, de la même façon qu'une classe standard mais sans aucun contenu de méthode. Toutes les méthodes déclarées dans une interface doivent être publiques.

Pour implémenter une interface, l'opérateur `implements` est utilisé. Toutes les méthodes de l'interface doivent être implémentées dans une classe ; si ce n'est pas le cas, une erreur fatale sera émise. Les classes peuvent implémenter plus d'une interface en séparant chaque interface par une virgule.

### 5.5.8 Classe abstraite

On ne peut créer une instance d'une classe définie comme `abstract`. Toutes les classes contenant au moins une méthode abstraite doivent également être abstraites. Pour définir une méthode abstraite, il faut simplement déclarer la signature de la méthode (précédée de `abstract` et ne fournir aucune implémentation).

Lors de l'héritage d'une classe abstraite, toutes les méthodes marquées comme abstraites dans la déclaration de la classe parente doivent être définies par l'enfant ; de plus, ces méthodes doivent être définies avec la même visibilité, ou une visibilité moins restreinte. Par exemple, si la méthode abstraite est définie comme protégée, l'implémentation de la fonction doit être définie comme protégée ou publique, mais non privée.

### 5.5.9 Méthodes magiques

Les méthodes : `__construct`, `__destruct`, `__toString`, `__clone`, `__get`, `__set`, `__call`, ... sont magiques en PHP. Vous ne pouvez pas utiliser ces noms de méthode dans vos classes, sauf si vous voulez implémenter le comportement associé à ces méthodes magiques.

**toString** détermine comment l'objet doit réagir lorsqu'il est traité comme une chaîne de caractères (`echo` ou `print`);

**clone** une fois le clonage effectué, si une méthode `__clone()` est définie, celle-ci sera appelée sur le nouvel objet ;

**get, set void \_\_set(string \$name, mixed \$value)** sera appelé lorsque l'on essaie d'affecter une valeur à un attribut inaccessible. Cela permet d'ajouter dynamiquement de nouveaux attributs (prototype).

**call, callStatic** appelé lorsque l'on essaie d'appeler une méthode inaccessible. Cela permet d'ajouter dynamiquement de nouvelles méthodes (prototype).

### 5.5.10 Clônage

Lorsqu'un objet est cloné, PHP effectue une copie **superficielle** de toutes les propriétés de l'objet. Toutes les propriétés qui sont des références à d'autres variables demeureront des références. L'opérateur `clone` est utilisé comme suit :

```
$copie = clone $objet;
```

### 5.5.11 Comparaison d'objets

Lors de l'utilisation de l'opérateur de comparaison `==`, deux objets sont égaux s'ils ont les mêmes attributs et valeurs, et qu'ils sont des instances de la même classe. Lors de l'utilisation de l'opérateur d'identité `===`, les objets sont identiques uniquement s'ils font référence à la même instance de la même classe.

### 5.5.12 L'interface iterator

Cette interface permet d'utiliser la structure de contrôle `foreach` afin de parcourir tous les élément dans une itération. Voici la liste des méthodes à implémenter :



```

Iterator extends Traversable {
/* Methods */
abstract public mixed current ( void )
abstract public scalar key ( void )
abstract public void next ( void )
abstract public void rewind ( void )
abstract public boolean valid ( void )
}

```

## 5.6 Sérialisation

La sérialisation permet de transformer toute variable complexe (objet, tableau) de PHP en une chaîne de caractères qui pourra être désérialisée ultérieurement. La chaîne sérialisée peut être sauvée dans un fichier ou émise dans un réseau.

```

string serialize ( mixed $value )
mixed unserialize ( string $str )

```

La sérialisation permet également de réaliser une copie **profonde** d'un objet : `$copieprof=unserialize(serialize($object))`. Attention, les attributs de type ressource (fichier, connexion) ne sont pas sérialisables.

## 5.7 Caractères spéciaux

Sources de séances de débogages longues et pénibles, il est important de comprendre les transformations des caractères spéciaux.

- Certains caractères (anti-slash, guillemet, NULL et apostrophe) saisis dans des champs de formulaires : (`<input ... name="champ">`) sont automatiquement **échappés** par un anti-slash par PHP (magic\_quotes). La variable PHP correspondante `$_POST['champ']` (ou GET) doit donc être traitée par : `stripslashes($_POST['champ'])` pour supprimer **tous** les anti-slash générés.
- **Attention**, ceci est valable pour tous les tableaux super-globaux : GET, POST, COOKIE. En particulier, un objet sérialisé dans un cookie aura été échappé si `get_magic_quotes_gpc()` est vrai ! Il faut donc enlever les slashes avant la désérialisation !
- Lors de l'affichage d'une chaîne contenant des caractères html spéciaux (`&"<'>`), il faut les transformer en entité HTML (`&quot;`). Pour afficher proprement une chaîne contenant ces caractères, il faut au préalable la traiter avec `htmlspecialchars($champ, ENT_QUOTES)`.
- **Attention**, si le champ posté doit être remis en tant que "value" dans un `<input type='text'` de formulaire, il faut donc enlever les anti-slash **puis** transformer ces caractères en entités HTML
- Par conséquent, pour un champ interactif, on écrira :  

```
echo '<input name="champ" value="'.htmlspecialchars(stripslashes($_POST['champ'], ENT_QUOTES)).'>';
```
- La suppression des slashes sur un tableau, par exemple de checkbox, pose problème ! Il faut les supprimer sur chaque élément du tableau !
- Lors de l'envoi d'une requête à un SGBD, il faut parfois échapper les caractères spéciaux tels que : ', \, ", NULL. La fonction `addslashes($chaine)` effectue ce travail.
- De même en JavaScript, la fonction `addslashes()` permettra d'échapper du code.
- Remarquons que dans une session, les caractères spéciaux ne sont pas échappés.

## 5.8 Administration PHP

Pour visualiser la configuration PHP, il suffit d'appeler la fonction `phpinfo()` pour voir :

- la version de php ;
- la localisation du fichier de configuration de php : `php.ini` ;
- les bibliothèques incluses et leur configuration (mysql, gd, ...);
- de nombreuses autres choses importantes.

La configuration de l'interprète PHP est réalisé dans le fichier `php.ini`. Ce fichier `php.ini` contient des directives (sous forme d'affectation de variables) fondamentales :

- register\_globals=On** si **on** alors les variables d'Environnement, Get, Post, Cookie, Serveur sont globales ; si **off** alors les variables sont accessibles via `$_POST[]`, ... (Sécurité +);
- variables\_order="EGPCS"** ordre de résolution des conflits de noms de variables Env, GET, POST, COOKIE, Serveur ;

**magic\_quotes\_gpc=On** permet d'anti-slasher les caractères anti-slash, guillemet et apostrophe dans les variables GPC.

## 5.9 Session

HTTP n'étant pas un protocole orienté connexion, chaque nouvelle requête semble être la première. Depuis PHP4, Les **sessions** PHP permettent de conserver de l'information **du côté serveur** sur la suite de requêtes émises par le même client (navigateur). Les variables enregistrées dans le tableau super-global `$_SESSION` peuvent être de type quelconque. Pour les variables de type tableau ou objet, elles seront sérialisées et désérialisées automatiquement dans le fichier de session côté serveur. Attention cependant à définir la classe d'un objet enregistré en session **avant** l'appel à `session_start()`.

La communication des identifiants de sessions est réalisée :

- soit par cookie ce qui est le plus simple ;
- soit par URL de manière quasi-transparente pour le programmeur.

Par défaut, la durée de vie du cookie de session (0) est égale à la durée de vie du navigateur.

**bool session\_start()** démarre une session ; doit être réalisé en tout **début de script** avant tout en-tête !

**string session\_id()** retourne l'identifiant de session qui est une suite de chiffres hexadécimaux.

**\$\_SESSION['z']**="contenu" ; ajoute une variable de session ;

**echo \$\_SESSION['z']** ; affiche le contenu d'une variable de session ;

**bool session\_is\_registered("mavar")** teste si \$mavar a été sauvé dans la session ;

**bool session\_unregister("x")** supprime la variable de session x ;

**\$\_SESSION=array()** ; réinitialise la session !

**string serialize(mixed)** retourne une chaîne composée des différents champs de l'objet ou du tableau ;

**mixed unserialize(string)** retourne la valeur de la variable d'origine ;

**bool session\_destroy()** supprime toutes les données de la session ; le cookie de session sera supprimé dès la prochaine page.

**bool session\_register("x", "y", ...)** démarre une session si pas encore fait et y enregistre les variables \$x et \$y ;

Afin de gérer les sessions de manière transparente, le script PHP doit tenir compte du fait que le navigateur client peut accepter ou refuser les cookies. Pour ce faire, il suffit d'indiquer dans chaque référence interne du site (href d'ancre, action de formulaire, ...), la constante SID comme paramètre de l'URL afin de transmettre l'identifiant de session. En effet, la constante SID a comme valeur :

- " chaîne vide lorsque le navigateur accepte les cookies ;
- 'PHPSESSID=0c92dbd...51ff2' lorsque le navigateur ne les accepte pas ;

Ainsi dans un formulaire, l'attribut action devra avoir la forme suivante :

`action="<?php echo "{$_SERVER['PHP_SELF']}".(strlen(SID)??'.SID:'); ?>"` L'exemple suivant illustre les sessions avec un historique de multiplications.

### Exemple 17 (Multiplication avec mémorisation des résultats dans une session)

```
<?php
session_start();
?>
<html><head><title>Multiplication et session</title></head><body>
<h1>Multiplication et session</h1>
<?php
if(isset($_SESSION['historique'])){ //
    $historique=unserialize($_SESSION['historique']); // tableau
    //print_r($historique);
    foreach($historique as $tab){ // tableau 'x'=>2, 'y'=>3, 'r'=>6
        echo "{$tab['x']} * {$tab['y']} = {$tab['r']}<br/>\n";
    }
    echo "<hr/>\n"; // pour délimiter l'historique
}else { // début de session
    $historique=array(); // init tableau de tableau
    $_SESSION['historique']=serialize($historique);
```

```

}

if ($_POST['mult']){ // nouvelle mult
    echo "Mult. courante : {$_POST['x']} * {$_POST['y']} = " . $_POST['x'] * $_POST['y'] .
        " !<br/>";
    $tab=array('x'=>$_POST['x'], 'y'=>$_POST['y'], 'r'=>$_POST['x'] * $_POST['y']);
    $historique[]=$tab; // ajout dans l'historique
    $_SESSION['historique']=serialize($historique);
    echo "<hr/> Nouvelle Multiplication :<br/>";
}
?>
<form action="<?php echo "{$_SERVER['PHP_SELF']}".(strlen(SID)??'.SID:'); ??"
    method="post">
X<input type="text" name="x" size="10"><br>
Y<input type="text" name="y" size="10"><br>
<input type="submit" value="Multiplier !" name="mult">
</form>
</body></html>

```

*La page affichée ressemblera à :*

```

Multiplication et session
2 * 3 = 6
4 * 5 = 20
-----
Mult. courante : 6 * 7 = 42 !
-----
Nouvelle Multiplication :
X ----
Y ----
Multiplier!

```

*Avec cookies non acceptés, voici le contenu de la barre d'adresse :*

```
.../multsession.php?PHPSESSID=1d5192e149789a9a52b10f948bbf6843
```

### 5.9.1 Que mettre comme variable de session ?

Si l'on utilise une BD pour stocker les informations, il est souhaitable de ne mettre en session qu'un identifiant afin de récupérer à chaque fois les informations courantes. Si l'on stocke un objet, il faut comprendre que tous les objets référencés par cet objet sont représentés dans la variable de session (faire un `print_r`). Aussi, les mises-à-jour dans la BD d'objets référencés peuvent devenir invisibles depuis l'objet de session !

Si l'on n'utilise pas de BD, alors toute l'information étant dans l'objet de session, il conviendra de le mettre à jour.

### 5.9.2 Session et Objets

Si l'on souhaite mettre un objet en variable de session, il faut que la définition de la classe soit chargée avant le début de session (`session_start()`). Ceci peut se faire par un `include` classique ou bien par l'auto-chargement (`autoload`) qui permet de charger le fichier définissant la classe à la demande.

```

// classLoader.php
function __autoload($class_name) {
    include 'MesClasses/' . $class_name . '.php';
}

// testClass.php dans le rép MesClasses
<?php
class testClass {
    private $prop1;

    function __construct($propValue) {

```

```

    $this->prop1 = $propValue;
}

function showProp() {
    return $this->prop1;
}
}
?>

// page1.php
<?php
require_once('classLoader.php');
session_start();
$_SESSION['testObj'] = new testClass('foo');
echo '<a href="page2.php">Go to page 2</a>';
?>

// page2.php
<?php
require_once('classLoader.php');
session_start();
echo $_SESSION['testObj']->showProp(); // displays foo
?>

```

## 5.10 Bases de données avec PDO

Les fonctions PHP d'accès aux Système de Gestion de Bases de Données sont nombreuses. Il existait des APIs spécialisées natives pour des sgbds tels que MySQLi ou Oracle OCI8 ou PostgreSQL, mais depuis PHP 5.1, l'extension PHP Data Objects (PDO) est installée par défaut et **doit être utilisée** car elle est indépendantes du sgbd cible. Cette interface d'abstraction à l'accès de données signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quelque soit la base de données utilisée.

### 5.10.1 Introduction à PDO

PDO fournit une interface d'abstraction à l'accès de données, ce qui signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quelque soit la base de données utilisée (MySQL, Oracle, ...). PDO est fourni avec PHP 5.1 et requiert les nouvelles fonctionnalités OO fournies par PHP 5.

### 5.10.2 Connexions et gestionnaire de connexion

Les connexions sont établies en créant des instances de la classe de base de PDO quel que soit le gestionnaire (driver) de SGBD utilisé. Le constructeur nécessite un paramètre pour spécifier la source de la base de données (Data Source Name) et optionnellement, le nom d'utilisateur et le mot de passe (s'il y en a un). Le DSN est une chaîne de caractères composée :

- d'un préfixe indiquant le gestionnaire, par exemple `mysql:` ou `pgsql:` ou ...;
- d'une suite de paramètres séparés par des **point-virgules** de la forme `param1=val1;param2=val2;...` Ces paramètres peuvent être : `host`, `port`, `dbname`, `user`, `password`

Remarquons que le couple (nom d'utilisateur, mot de passe) peuvent être fournis dans le DSN (prioritaire) ou comme paramètres du constructeur.

#### Exemple 18 (Connexion à MySQL)

```

<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
?>

```

S'il y a des erreurs de connexion, un objet PDOException est lancé. Vous pouvez attraper cette exception si vous voulez gérer cette erreur, ou laisser le gestionnaire global d'exception défini via la fonction `set_exception_handler()` la traiter.

**Exemple 19 (Connexion à PostgreSQL avec gestion des erreurs)**

```
<?php
try {
    $dbh = new PDO('pgsql:dbname=$dbname;host=$host;username=$username;password=$password');
    foreach($dbh->query('SELECT * from etudiant') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Erreur !: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Pour fermer la connexion, il suffit de déréférencer l'objet PDO en affectant la variable à nul : `$dbh=null`;

Les connexions **persistantes** ne sont pas fermées à la fin du script, mais sont mises en cache et réutilisées lorsqu'un autre script demande une connexion en utilisant les mêmes paramètres. Le cache des connexions persistantes vous permet d'éviter d'établir une nouvelle connexion à chaque fois qu'un script doit accéder à une base de données, rendant l'application web plus rapide.

**Exemple 20 (Connexion persistante)**

```
<?php
$dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass, array(
    PDO::ATTR_PERSISTENT => true
));
?>
```

**5.10.3 Requêtes uniques**

Pour des requêtes individuelles, on peut utiliser la méthode `query` pour une consultation ou la méthode `exec` pour une modification. Il faut noter que l'objet `PDOStatement` retourné par `query` est `Traversable`, c'est-à-dire qu'on peut itérer directement dessus avec `foreach` (fetch automatique).

**Exemple 21 (Requête unique)**

```
<?php
try {
    $sql = 'SELECT nom, ue FROM utilisateur u, etudiant e WHERE
    u.id=e.id ORDER BY ue, nom';
    foreach($dbh->query($sql) as $row) {
        print($row['nom'] . "\t" . $row['ue'] . "<br />");
    }
    $nb=$dbh->exec("DELETE FROM utilisateur WHERE id NOT IN (SELECT id FROM etudiant)");
    print("Nb de suppressions : " . $nb);
}
?>
```

**5.10.4 Requêtes préparées (PDOStatement)**

Une requête préparée est une sorte de modèle compilé pour la(es) requête(s) SQL que vous voulez exécuter. Les requêtes préparées offrent deux fonctionnalités essentielles :

- La requête ne doit être analysée (ou préparée) qu'une seule fois, mais peut être exécutée plusieurs fois avec des paramètres identiques ou différents. L'optimisation de son plan d'exécution permet à la requête préparée d'utiliser moins de ressources et de s'exécuter plus rapidement.
- Les paramètres pour préparer les requêtes (`:nomparam`) n'ont pas besoin d'être entre guillemets; le driver le gère pour vous. Si votre application utilise exclusivement les requêtes préparées, vous pouvez être sûr qu'aucune injection SQL n'est possible.

PDO émule les requêtes préparées pour les drivers qui ne les supportent pas. Ceci assure de pouvoir utiliser la même technique pour accéder aux données, sans se soucier des capacités de la base de données.

**Exemple 22 (Insertions répétées en utilisant les requêtes préparées)**

Cet exemple effectue une requête `INSERT` en y substituant un nom et une valeur pour les **marqueurs nommés**.

```

<?php
$stmt = $dbh->prepare("INSERT INTO REPERTOIRE (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name); // association paramètre avec variable PHP
$stmt->bindParam(':value', $value); // association paramètre avec variable PHP

// insertion de deux lignes
$name = 'Dupont';
$value = 0612345678;
$stmt->execute();

$name = 'Durand';
$value = 0468901234;
$stmt->execute();
?>

```

### Exemple 23 (Insertion avec marqueur anonyme)

```

<?php
<$stmt = $dbh->prepare("INSERT INTO REPERTOIRE (name, value) VALUES (?,?)");
$stmt->bindParam(1, $name); // association paramètre avec variable PHP
$stmt->bindParam(2, $value);

// insertion
$name = 'Martin';
$value = 0612435678;
$stmt->execute();
?>

```

### Exemple 24 (Consultation paramétrée avec marqueur anonyme)

```

<?php
$stmt = $dbh->prepare("SELECT * FROM REPERTOIRE where name = ?");
if ($stmt->execute(array($_POST['nom']))) { // liaison tardive
    while ($row = $stmt->fetch()) {
        print($row['name']." : ".$row['value']);
    }
}
?>

```

La récupération (fetch) des lignes résultant d'une consultation est réalisée séquentiellement selon la valeur du paramètre optionnel de la méthode fetch. Par défaut, la valeur PDO::FETCH\_BOTH permet de retourner la ligne suivante en tant que tableau indexé par le nom et le numéro de la colonne (0 à n-1). Mais on peut également :

- récupérer un objet dont les attributs correspondront aux colonnes ;
- se déplacer selon une orientation (vers le bas ou le haut) ;
- indiquer un déplacement différent de 1 dans la séquence de récupération.

## 5.10.5 Transaction

Les transactions offrent 4 fonctionnalités majeures : Atomicité, Consistance, Isolation et Durabilité (ACID). Le travail d'une transaction (suite de requêtes) peut être annulé (abort) à votre demande ou bien validé (commit). Malheureusement, toutes les bases de données ne supportent pas les transactions, donc, PDO doit s'exécuter en mode "autocommit" lorsque vous ouvrez pour la première fois la connexion. Le mode "autocommit" signifie que toutes les requêtes que vous exécutez ont leurs transactions implicites, si la base de données le supporte ou aucune transaction si la base de données ne les supporte pas.

Si vous avez besoin d'une transaction, vous devez utiliser la méthode PDO::beginTransaction() pour l'initialiser. Si le driver utilisé ne supporte pas les transactions, une exception PDO sera lancée. Une fois que vous êtes dans une transaction, vous devez utiliser la fonction PDO::commit() ou la fonction PDO::rollback() pour la terminer.

**Attention**, PDO ne vérifie la possibilité d'utiliser des transactions qu'au niveau du pilote. Si certaines conditions à l'exécution empêchent les transactions de fonctionner, PDO::beginTransaction() retournera tout de même TRUE sans erreur si le serveur accepte de démarrer une transaction. Ce sera le cas en utilisant le pilote MySQL sur des tables au format MyISAM car ce dernier ne supporte pas les transactions (utiliser plutôt InnoDB).

Lorsque le script se termine ou lorsque la connexion est sur le point de se fermer, si vous avez une transaction en cours, PDO l'annulera automatiquement. Ceci est une mesure de sécurité afin de garantir la consistance de vos données dans le cas où le script se termine d'une façon inattendue. Si vous ne validez pas explicitement la transaction, alors, on présume que quelque chose s'est mal passé et l'annulation de la transaction intervient afin de garantir la sécurité de vos données.

### Exemple 25 (Transaction)

```
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass,
        array(PDO::ATTR_PERSISTENT => true));
    echo "Connecté\n";
} catch (Exception $e) {
    die("Impossible de se connecter: " . $e->getMessage());
}
try {
    $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // pour lancer une exception sur chaque erreur
    $dbh->beginTransaction();
    $dbh->exec("insert into utilisateur (id, nom, prenom) values (12, 'Dupont', 'Jean')");
    $dbh->exec("insert into etudiant (id, ue) values (12, '13')");
    $dbh->commit();
} catch (Exception $e) {
    $dbh->rollBack();
    echo "Impossible d'ajouter l'étudiant : " . $e->getMessage();
}
?>
```

### 5.10.6 Synopsis des classes PDO et PDOStatement

```
PDO {
public __construct ( string $dsn [, string $username [, string $password [, array $driver_options ]]] )
public bool beginTransaction ( void )
public bool commit ( void )
public mixed errorCode ( void )
public array errorInfo ( void )
public int exec ( string $statement )
public mixed getAttribute ( int $attribute )
public static array getAvailableDrivers ( void )
public bool inTransaction ( void )
public string lastInsertId ([ string $name = NULL ] )
public PDOStatement prepare ( string $statement [, array $driver_options = array() ] )
public PDOStatement query ( string $statement )
public string quote ( string $string [, int $parameter_type = PDO::PARAM_STR ] )
public bool rollBack ( void )
public bool setAttribute ( int $attribute , mixed $value )
}
PDOStatement implements Traversable {
/* Propriétés */
readonly string $queryString;
/* Méthodes */
public bool bindColumn ( mixed $column , mixed &$param [, int $type [, int $maxlen [, mixed $driverdata ]]] )
public bool bindParam ( mixed $parameter , mixed &$variable [, int $data_type = PDO::PARAM_STR [, int $length ] ] )
public bool bindValue ( mixed $parameter , mixed $value [, int $data_type = PDO::PARAM_STR ] )
public bool closeCursor ( void )
public int columnCount ( void )
public void debugDumpParams ( void )
public string errorCode ( void )
public array errorInfo ( void )
public bool execute ([ array $input_parameters ] )
```

```

public mixed fetch ([ int $fetch_style [, int $cursor_orientation = PDO::FETCH_ORI_NEXT [, int $cursor_offset
public array fetchAll ([ int $fetch_style [, mixed $fetch_argument [, array $ctor_args = array() ]]] )
public string fetchColumn ([ int $column_number = 0 ] )
public mixed fetchObject ([ string $class_name = "stdClass" [, array $ctor_args ] ] )
public mixed getAttribute ( int $attribute )
public array getColumnMeta ( int $column )
public bool nextRowset ( void )
public int rowCount ( void )
public bool setAttribute ( int $attribute , mixed $value )
public bool setFetchMode ( int $mode )
}

```

### 5.10.7 Un exemple de TP

#### Exemple 26

```

<html><head> <title>Test de PDO et du driver MySQL</title>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
</head>
<body>
  <h1>Test de PDO</h1>
<?php
try {
  $dbh = new PDO('mysql:host=venus;dbname=mmeynard', "mmeynard", "XXXX");
  foreach($dbh->query('SELECT * from test') as $row) {
    print_r($row);
  }
  $dbh = null; // fermeture connexion
} catch (PDOException $e) {
  print "Erreur !: " . $e->getMessage() . "<br/>";
  die();
}
?>
</body>
</html>

```

## 5.11 MySQL

### 5.11.1 phpMyAdmin

phpMyAdmin est un outil d'administration d'une BD MySQL écrit en PHP. Cet outil est accessible via le web et permet donc d'administrer à distance une BD. On peut : créer, supprimer, modifier (alter) des tables, interroger, ajouter, supprimer, modifier des lignes, importer des fichiers textes contenant les données ... Le site de référence où l'on peut tester en direct est : [http://www.phpmyadmin.net/home\\_page/index.php](http://www.phpmyadmin.net/home_page/index.php).

### 5.11.2 Moteur de stockage

MySQL supporte plusieurs moteurs de stockage, qui gère différents types de tables. Les moteurs de tables MySQL peuvent être transactionnels ou non-transactionnels. Historiquement et par défaut, le moteur de stockage est MyISAM : il ne permet ni contraintes d'intégrité référentielles (clés étrangères), ni transactions mais est très rapide. Le moteur de stockage InnoDB est une alternative à privilégier puisqu'elle offre ces fonctionnalités indispensables (transactions, références) en plus d'autres.

## 5.12 Cookies

Les cookies sont stockées du côté client par le navigateur et contiennent des variables. Quand celui-ci envoie une requête à une URI d'un domaine et d'un chemin dont il possède un cookie, toutes les variables du cookie sont envoyées au serveur. Un cookie a une durée de vie (expire) par défaut égale à la durée de vie de la session. Pour accepter/afficher les cookies sur Firefox, utiliser le menu Outlis, Option, Vie Privée, Accepter/Afficher les cookies.



```

int setcookie('x','Hello!',int expire, string path, string domain, int secure) doit être exécutée avant toute chose et permet de positionner la variable $x avec la valeur 'Hello!'; attention le cookie ne sera renvoyé au serveur qu'à la prochaine requête;
$_COOKIE['x'] accès à une variable de cookie;
setcookie('x',",time()-1000) efface la variable x du cookie;
setcookie('y',$value,time()+3600); expire dans une heure;
setcookie('x','345'); expire en fin de session c'est-à-dire lorsque le navigateur sera fermé (expire=0);

```

## 5.13 Cookies et session

Les variables de session sont conservées côté serveur. Du côté client, si les cookies sont acceptées, une variable de cookie, généralement nommée PHPSESSID est présente. Elle contient un identifiant qui est envoyé au serveur à chaque requête ce qui permet de l'associer à la session correspondante. La variable de configuration `session.cookie_lifetime` spécifie la durée de vie du cookie en secondes. Par défaut, la valeur de 0 signifie : "Jusqu'à ce que le navigateur soit éteint". Cependant, une autre variable de configuration `session.gc_maxlifetime` qui vaut généralement 1440 soit 24 minutes, indique le temps maximum dont jouissent les données de session sur le serveur. Par conséquent, au bout de 24 minutes d'INACTION, les données de session seront supprimées même si le cookie de session perdure côté navigateur.

## 5.14 Authentification

L'authentification selon le mode courant (login, password) peut être réalisé de bien des façons différentes sur le web. Dans la plupart des cas, on peut réaliser l'authentification par un formulaire HTML classique puis interrogation d'une base de données des utilisateurs contenant les couples (login, password **crypté**). Une fois authentifié, il suffira de conserver dans le tableau de session les informations concernant l'utilisateur loggé (id, nom, type, ...) : `$_SESSION['uid']`, ...

Cependant, il faut également connaître les autres types d'authentification !

### 5.14.1 Authentification HTTP

La procédure d'authentification HTTP est associée à un **nom de domaine** (realm ou AuthName) et à un répertoire. Elle peut être déclenchée :

- soit par Apache, indépendamment de PHP ;
- soit en utilisant PHP.

#### Authentification HTTP via Apache

L'authentification est valable pour toute une **arborescence**. Un fichier `.htaccess` spécifie les règles d'authentification valables pour ce répertoire et tous ses **descendants** :

```

AuthType Basic
AuthUserFile "/auto/.../AuthApache/.htpasswd"
AuthName "Le Domaine Privé"
<Limit GET POST>
  require valid-user
</Limit>

```

Le fichier `.htpasswd` contient la liste des utilisateurs et leurs mots de passe cryptés. Il est obtenu grâce à l'utilitaire `htpasswd` fourni par Apache. Par exemple : `htpasswd -c .htpasswd un`; crée un fichier avec l'utilisateur `un`.

Lors de tout accès à un fichier descendant de `AuthApache`, Apache va envoyer un en-tête au navigateur client qui va afficher une fenêtre popup d'authentification. Par la suite, l'utilisateur reste authentifié pour "Le Domaine Privé" jusqu'à la fin du navigateur ou si une nouvelle authentification PHP est lancée.

#### Authentification HTTP via PHP

PHP doit être un module d'Apache. On utilise alors la fonction `header` pour demander une authentification ("WWW-authenticate") au client, générant ainsi l'apparition d'une fenêtre de demande d'utilisateur et de mot de passe. Une fois que les champs ont été remplis, l'URL sera de nouveau appelée, avec les variables `$_SERVER['PHP_AUTH_USER']`, `$_SERVER['PHP_AUTH_PW']` et `$_SERVER['PHP_AUTH_TYPE']` contenant respectivement le nom d'utilisateur, le mot de

pas et le type d'authentification. Actuellement, seule l'authentification de type "Basic" est supportée. Si l'authentification est réalisée via Apache (.htaccess), la variable `$_SERVER['REMOTE_USER']` est égale à `$_SERVER['PHP_AUTH_USER']`.

### Exemple 27 (Exemple d'authentification HTTP par PHP)

```
<?php
if (!isset($_SERVER['PHP_AUTH_USER']) || !verifierAuth())
    header("WWW-Authenticate: Basic realm=\"Le Domaine Privé\"");
    header("HTTP/1.0 401 Unauthorized");
    echo "Texte à envoyer si le client annule\n";
    exit;
} else {
    echo "Bonjour ", $_SERVER['PHP_AUTH_USER'];
    // suite de la page privée
}
?>
```

La fonction booléenne `verifierAuth()` utilisera tout moyen nécessaire à la vérification du nom et du mot de passe (Base de données, ...). Remarquons que les variables `$_SERVER['PHP_AUTH_...']` sont utilisables même si l'authentification n'a pas été effectuée par le module PHP.

**Les variables d'authentification PHP sont valables pour tous les fichiers descendants du répertoire.** Par contre, tout fichier html ou php ne testant pas l'authentification est accessible, contrairement à l'authentification par .htaccess (Apache) qui sécurise tout le répertoire.

### Désauthentification PHP

Le navigateur écrase le cache d'authentification client d'un domaine quand il reçoit une nouvelle demande d'authentification. Cela permet de déconnecter un utilisateur, pour le forcer à entrer un nouveau nom et son mot de passe. Si l'utilisateur annule l'authentification, il est alors désauthenticé! Penser à recharger la page. Certains programmeurs changent dynamiquement le nom de domaine pour donner un délai d'expiration, ou alors, fournissent un bouton de réauthentification.

## 5.15 Téléchargement

- du site web vers le client : download ou téléchargement ;
- du client vers le site web : upload ou chargement ;

### 5.15.1 Déchargement

Réaliser un lien référençant le fichier à télécharger, par exemple : `<a href="Public/toto.ps">Cliquer ici</a>`. Si le type du fichier est affichable par le navigateur, il sera affiché (donc téléchargé), sinon il sera proposé à l'utilisateur soit de sauver le fichier, soit de lancer l'application associée. Remarquons que tout lien peut être enregistré en utilisant le bouton droit de la souris sur le lien.

### 5.15.2 Chargement

Réaliser un formulaire de chargement envoyant une requête POST. L'action effectuée (script) après soumission doit vérifier que le fichier chargé est du bon type et de la bonne taille puis l'afficher depuis la zone temporaire (tmp/) ou il est chargé. Le fichier temporaire sera **automatiquement effacé** de la zone à la fin du script, s'il n'a pas été déplacé ou renommé. Par exemple, le formulaire sera :

```
<FORM ENCTYPE="multipart/form-data" ACTION="traitement.php" METHOD="POST">
    <INPUT TYPE="hidden" NAME="MAX_FILE_SIZE" value="1000">
    Envoyez ce fichier : <INPUT NAME="fichier" TYPE="file" SIZE=15>
    <INPUT TYPE="submit" VALUE="Envoyer le fichier">
</FORM>
```

Dans le script `traitement.php` on a accès à différentes variables (différent selon les versions de PHP) :

- `$_FILES['fichier']['name']` le nom du fichier original chez le client ;
- `$_FILES['fichier']['type']` le type mime du fichier "image/gif, text/plain, ...";
- `$_FILES['fichier']['size']` la taille du fichier chargé;
- `$_FILES['fichier']['tmp_name']` le nom du fichier temporaire sur le serveur ;
- `$_FILES['fichier']['error']` le code d'erreur (PHP 4.2.0).

Remarquons que tous les navigateurs ne vérifient pas le `MAX_FILE_SIZE`.

## 5.16 Divers

### 5.16.1 SPL : la librairie standard PHP

La librairie SPL fournit par défaut (sans include, sans modifier le `php.ini`) des interfaces et des classes permettant de gérer des collections de données. Par exemple, suit une liste de quelques classes utiles de la SPL : `SplDoublyLinkedList`, `SplStack`, `SplQueue`, `SplHeap`, `SplPriorityQueue`, `SplObjectStorage`.

### 5.16.2 Phar : les archives PHP

Phar est un format de fichier archive (PHp ARchive) permettant de sauvegarder une application (arborescence) dans un unique fichier d'extension `phar`. C'est l'équivalent des fichiers `jar` de Java. L'interprète PHP en ligne de commandes est capable d'exécuter un fichier `phar` directement : `$ php monappli.phar`

On peut également définir une bibliothèque dans un fichier `phar` et n'inclure que certaines parties grâce à une syntaxe appropriée :

```
<?php
include 'malib.phar'; // inclusion de tous les fichiers
include 'phar://malib2.phar/Commun/inc-Html.php'; // Commun est un répertoire de malib2
```

Le format interne des fichiers de l'archives peut utiliser différents formats de compression (`zip`, `phar` ou `tar`).

## 5.17 Développement et débogage

Les données complexes (objets, tableaux) peuvent être affichés récursivement de manière indentée grâce à `print_r($tab)` ou `var_dump($tab)`.

### 5.17.1 Contexte de débogage

La fonction `array debug_backtrace()` permet de récupérer le contexte de débogage sous la forme d'un tableau de tableau. Chaque case correspond à un appel de fonction emboîté depuis la courante à l'indice 0 (celle qui appelle `debug_backtrace`) jusqu'à la plus lointaine (*main*). Chaque case contient un tableau associatif :

- function** nom de la fonction courante. Voir aussi `__FUNCTION__` ;
- line** numéro de la ligne courante. Voir aussi `__LINE__` ;
- file** nom du fichier courant. Voir aussi `__FILE__` ;
- class** nom de la classe si on est dans une méthode ;
- object** objet courant si on est dans une méthode ;
- type** type d'appel : méthode d'instance "->", méthode statique " : : " ;
- args** liste des arguments ou liste des fichiers inclus.

Certaines constantes magiques peuvent être utiles à des fins de débogage :

- `__FUNCTION__` nom de la fonction courante ;
- `__FILE__` nom du fichier exécuté ;
- `__LINE__` numéro de ligne courante ;

La fonction `debug_print_backtrace()` permet d'afficher les appels emboîtés de fonction.

### 5.17.2 Gestion des erreurs

On doit fixer un niveau de rapport exigeant à l'interprète afin de vérifier tous les avertissements et erreurs possibles : `error_reporting(E_ALL)`; Cependant, une directive de configuration dans `php.ini` permet d'afficher ou de cacher les erreurs de quelque niveau que ce soit : `display_errors = On`. Si cette directive est à `Off`, ce qui est souhaitable en mode production, les erreurs PHP sont loggées dans un fichier `php_error.log` qu'il faut ouvrir à chaque fois que l'on suspecte une erreur !

Visualiser le source HTML sur le navigateur (Ctrl U) est très souvent utile notamment pour la gestion des caractères spéciaux.

La console Javascript ou console d'erreurs permet également de visualiser les problèmes locaux aux navigateur : javascript et feuille de style.

Certaines extensions de navigateur, telle que **Firebug** pour **Firefox**, sont indispensables afin d'étudier le code produit.

# Chapitre 6

## Architecture de site

### 6.1 Introduction

Ce chapitre est destiné à indiquer la ligne de conduite à tenir lors de la création d'un site Web.

### 6.2 Analyse des besoins

Lors de la création d'un nouveau site Web ou de la mise à jour d'un site Internet existant, il faut commencer par analyser les besoins :

- Quelle est la cible du site ?
- Choix du vocabulaire exact de la cible, sans ambiguïtés ni imprécisions.
- Choix des fonctionnalités à intégrer au site.
- Choix des contenus : Quels sont ceux qui seront les plus opportuns ?
- Choix d'une communication (image de marque, charte graphique, argumentaire, ...).
- Structuration du contenu et de la navigation. Ergonomie du site web.
- Règles de rédaction du contenu ;
- Comment améliorer le référencement.
- Rédaction du cahier des charges.

### 6.3 Cahier des charges

#### 6.3.1 Les exigences

**Contexte** décrire les caractéristiques fondamentales de votre groupe ou organisme (histoire, buts, principes fondamentaux, fonctionnement, budget, etc) ;

**Type du site** définir ici ce que vous attendez de votre site. S'agira t'il d'un site vitrine, d'un site outil ? vous placez vous dans une logique de communication (site-vitrine) ou d'information (site-service) ? souhaitez vous un site de vente ou un site d'informations ?

**Rubriques** il s'agit de l'architecture de votre information, des grandes rubriques et sous rubriques qui structureront le contenu du site.

**Description des fonctionnalités** Voulez vous un forum, un agenda, des listes de discussion, des base de données consultables en ligne ? tous ces exemples sont autant de fonctionnalités types d'un site Internet, que vous aurez à définir au préalable.

**La cible** préciser ici le public que vous souhaitez atteindre. L'aspect général du site, son graphisme et ses animations en dépendront.

**L'image** quelle image souhaitez vous donner de votre groupe, association ou organisme ? Les objectifs du site : il peut s'agir de la constitution d'une base de donnée, de la diffusion d'une information particulière ou encore de la mise en réseau de compétences propres à votre groupe.

**Périodicité et types de mises à jour** définir ici la façon dont les contenus futurs devront être mis en ligne : Souhaitez vous gérer en interne ou en externe les mises en ligne d'informations, de photos ou des autres contenus ?

**Bénéfice attendu** décrire en quelques mots le bénéfice que vous attendez de ce site. Il sera aussi important de définir les personnes les plus concernées par ce bénéfice attendu.

### 6.3.2 Le calendrier

**La dead line** la description du projet pourra fixer une date clé pour la mise en ligne et la remise des sources. Prévoir les modalités des renvois d'informations et la description des navettes et des validations.

### 6.3.3 Les ressources

**Ressources humaines** en externe, désigner une personne chargée de la coordination pour la réalisation et le suivi de votre site. Ce correspondant aura autorité pour prendre ou relayer les décisions vers le prestataire, et saura assurer les échanges mutuels. En interne, désigner un chef de projet et une équipe ayant les compétences dans les différents domaines (BD, prog, graphisme, ...).

**Budget** Plusieurs éléments sont à prévoir : conception, réalisation, suivi ; formation, embauche ; achat de matériel, de logiciel ou de livres ; coûts de connexion et d'hébergement...

**Contenus** vous créez un site pour y apporter du contenu. Vous pouvez dès à présent penser aux différentes photos, textes ou modules que vous placerez.

**Impératifs techniques** Matériel informatique disponible en interne ; Types de connexions et pratique de l'Internet. Matériel prévu spécialement pour le site. Hébergement interne ou externe ?

### 6.3.4 Maintenance

Un site Internet engendre des coûts annuels, liés essentiellement à sa connexion au réseau, à l'hébergement des pages chez le fournisseur d'accès et au maintien du nom de domaine. A ces frais fixes s'ajoutent ceux concernant la mise à jour régulière des données, les réponses rapides aux demandes qui vous parviennent et les actions de promotion de votre site.

## 6.4 Architecture

### 6.4.1 Arborescence du site

Définir clairement l'arborescence du site. Choisissez de concevoir plusieurs pages, avec de nombreuses sous thématiques, plutôt d'une page à contenu multiple trop longue et difficile à consulter. Faire un grand nombre de pages n'est pas un vice rédhibitoire, bien au contraire. Essayez de diviser votre site en thématique et sous thématique pour gagner en lisibilité et en clarté.

L'arborescence de votre site Web doit ressembler à un arbre généalogique avec la page d'accueil en haut et les différentes rubriques qui se répartissent en dessous en suivant une logique de navigation. Pour le moment, il est simplement question d'un plan du site succinct pour se donner une base de départ cohérente. Un bon début est un site de 10-15 pages clair et bien structuré. De toutes façons, votre site se doit d'être vivant et si vous vous impliquez correctement dans le projet, le nombre de pages ira vite en croissant. Ne soyez pas trop ambitieux au départ pour privilégier de solides fondations à votre site plutôt qu'un site mis en ligne à moitié achevé.

Attention quand même à ne pas tomber dans l'effet inverse qui est de créer trop de pages en diluant l'information sans raison. Une règle de base à adopter est celle des **3 clics** : il faut que l'internaute puisse accéder à l'information qu'il cherche en 3 clics de souris.

### 6.4.2 Soignez l'accueil

Votre page d'accueil est le carrefour de vos visites. Il faut la soigner pour faire une page attractive et structurée. Identifiez posément le thème de votre site et définissez clairement vos différentes rubriques. **Evitez** la page d'accueil constituée d'une animation flamboyante qui fait **patienter** l'internaute et engendre un clic de plus pour rien. Il vaut mieux aller directement aux faits en donnant un accès direct aux rubriques et sous rubriques du site dans une page plaisante à visualiser et bien structurée. De plus, évitez aussi les longs discours que vous garderez pour vos sous rubriques.

L'internaute est toujours pressé, alors ne risquez pas qu'il fuit votre site en alourdissant votre page d'accueil. Pour un chargement décent de votre page d'accueil en connexion bas débit, il faut qu'elle tombe en dessous de 40-50Ko, images comprises.

### 6.4.3 Menu clair et accessible

Votre menu va évoluer dans le temps, mais il faut le concevoir avec soin dès le départ. Définissez des effets de couleurs ou de soulignage lorsque l'internaute passe avec sa souris sur les liens. De plus, privilégiez les liens texte aux liens image ou zone réactive pour optimiser le référencement par les moteurs de recherche.

L'idéal serait d'avoir les différentes rubriques et sous rubriques accessibles depuis toutes les pages du site. L'internaute et le moteur de recherche apprécieront l'attention.

### 6.4.4 Plan du site

Il est intéressant à plusieurs titres d'intégrer un plan de votre site accessible depuis toutes les pages du site. D'une part, un internaute perdu pourra facilement retrouver son chemin et d'autre part le robot de crawl des moteurs de recherche aura plus de facilité à trouver les pages enterrées sous plusieurs niveaux de sous rubriques.

### 6.4.5 Intégrer un moteur de recherche

Si on garde à l'esprit que le but d'un site Web est de procurer à l'internaute l'information qu'il recherche dans les meilleures conditions et le plus rapidement possible, il est intéressant d'adjoindre un moteur de recherche à votre site. Plusieurs solutions s'offrent à vous, mais elles requièrent des connaissances en langage de programmation comme le PHP ou l'ASP. Si votre site est correctement référencé sur Google, c'est à dire que toutes les pages du site sont présentes dans son index, vous pouvez utiliser le moteur de recherche Google adaptable à votre site (pour savoir si toutes les pages de votre site Internet sont présentes dans Google, tapez la commande site :www.monsite.com dans Google).

### 6.4.6 Communiquez avec vos visiteurs

Toujours dans l'optique de fidéliser les internautes qui visitent votre site, il est important d'inclure un moyen de vous contacter. Au minimum, il faut que l'internaute puisse vous envoyer un eMail. Pour cela, faites un lien vers votre eMail, accessible depuis toutes les pages du site. Il s'agit d'un hyperlien de type mailto, que tous les éditeurs HTML sauront faire pour vous.

Pour aller plus loin, vous pouvez créer un formulaire de contact qui semble être un support plus convivial.

Parmi les autres moyens qui existent pour fidéliser les internautes, il faut citer la newsletter, les forums de discussion, les livres d'or, les Weblogs et les Chats.

### 6.4.7 Mentions légales

Désormais, la Commission Informatique et Libertés (CNIL) dicte à tous les sites français de s'identifier clairement. Il faut donc indiquer, sur la page d'accueil ou sur une page spéciale accessible depuis toutes les pages du site, certaines mentions obligatoires comme le propriétaire du site, la personne à contacter en cas de besoin de rectification sur les données personnelles et aussi l'hébergeur du site.

De plus, il faut déclarer son site Internet à la CNIL qui se chargera de vous attribuer un numéro.

En tout cas, il est utile de présenter une certaine transparence par rapport à votre site, augmentant ainsi le capital confiance que l'internaute attribuera à votre site Web.

### 6.4.8 Faites des liens

Un des composants essentiels à la réussite d'un site sur le Web est son référencement. Google et d'autres moteurs de recherche attribuent une attention toute particulière aux liens hypertexte qui renvoient d'un site à un autre. Les sites qui reçoivent beaucoup de liens entrants sont jugés comme populaires par les moteurs de recherche et ils sont ainsi favorisés dans les résultats de recherche.

Il est ainsi intéressant de demander des échanges de liens avec d'autres sites Web, de préférence à thèmes similaires ou complémentaires. Prévoyez donc une page d'échanges de liens pour placer les URLs des sites qui sont d'accord pour parler de vous, à condition que vous parliez d'eux.

## 6.5 La charte graphique

L'aspect visuel du site devra être homogène. Pour cela, on rédige une charte graphique, c'est à dire qu'on définit une mise en page à respecter sur l'ensemble du site. La cohérence graphique doit être respectée quels que soient les différents intervenants de la production (graphiste, directeur artistique, programmeur, ...).

Pour l'aspect graphique, on peut s'inspirer d'autres sites ou des modèles fournis par les logiciels de création de sites web, ainsi que de la mise en page de certains magazines ou d'affiches publicitaires.

Quelques règles de mise en page sont à respecter :

- L'oeil parcourt les pages en diagonal, du coin supérieur gauche au coin inférieur droit. Les éléments importants de la page sont à placer sur cet axe.
- On évitera que l'internaute ait à utiliser sa barre de défilement horizontale. L'affichage le plus répandu étant le 800x600 et 1024x768.
- En ce qui concerne la longueur de la page, il est recommandé de ne pas dépasser 3 fois la hauteur d'écran. Les informations importantes sont à placer en haut des pages : certains internautes n'utilisent pas le défilement vertical !
- Pour faciliter la lecture, les lignes ne doivent pas contenir plus d'une dizaine de mots, sinon l'oeil perd le fil de la lecture. Cela conduit souvent à utiliser des colonnes pour les textes.
- Pour des raisons esthétiques, se limiter à 3 polices. Préférer les polices sans empattement qui sont plus lisibles sur un écran. Arial et Verdana sont des bons choix. N'utiliser que des polices courantes qui seront disponibles chez l'internaute. Si une police exotique doit être employée (pour un titre par exemple) le mieux est d'utiliser un logiciel de retouche d'images pour transformer le texte en une image que l'on insérera dans la page.
- Le choix des couleurs : il existe des règles universelles régissant l'harmonie des couleurs, dûes à des propriétés physiques de l'oeil. En effet, lorsqu'il contemple une couleur, l'oeil crée automatiquement, sur son contour, un filtre de la couleur complémentaire. On parle de "contraste simultané". Avec ce mécanisme, la perception des couleurs dépend des couleurs avoisinantes. Ainsi, le jaune paraîtra plus orangé lorsqu'il est associé à du bleu et le bleu paraîtra plus violet. Un bleu à côté d'un rouge apparaîtra vert, etc.  
Par ailleurs, des couleurs voisines sur le diagramme chromatique créent une sensation d'équilibre pour l'oeil, en vertu de l'absence de contraste, on parle ainsi "d'harmonie d'analogie". Il existe donc globalement deux façons principales de choisir des couleurs harmonieuses :
  - en choisissant des nuances d'une même couleur, soit des couleurs de même teinte dont les tons sont proches ;
  - en mêlant des couleurs complémentaires (chaudes et froides), c'est-à-dire des couleurs éloignées sur le diagramme chromatique. Pour deux couleurs, il suffit de choisir des couleurs complémentaires, diamétralement opposées ; pour trois couleurs, les couleurs choisies doivent former un triangle équilatéral, etc.



# Chapitre 7

## Propel

### 7.1 Introduction

Propel est un ORM pour PHP 5. Il est possible de l'utiliser avec les frameworks Symfony et Symfony2. Un mapping objet-relationnel (en anglais object-relational mapping ou ORM) est une technique de programmation qui permet de manipuler une base de données relationnelle en utilisant la programmation par objets. Ainsi le code PHP est plus homogène, ce qui facilite sa programmation et son débogage. Propel est basé sur PDO pour l'accès au SGBD.

Propel existe actuellement dans une version 1.7 qui est stable et que nous utiliserons. Il existe également en version 2.0 en  $\alpha$  mais nécessite PHP 5.4 qui n'est pas installé dans les salles machines.

### 7.2 Installation de Propel 1.7

Plusieurs types d'installations sont possibles ; nous utiliserons Composer et nous installerons Propel sous la racine publique `public_html`.

#### 7.2.1 Composer : un gestionnaire de dépendances

Historiquement `pear` fut un gestionnaire de paquets PHP centralisé qui permettait de nombreuses fonctionnalités dont la gestion des dépendances entre paquets mais qui était un peu rigide. Aussi, les auteurs de paquets ou de bibliothèques PHP sont maintenant hébergés sur différents gestionnaires de version (`GitHub`, `GoogleCode`, ...) et proposent leur bibliothèque sous différents formats (`zip`, `tgz`, `phar`, ...).

Composer est un gestionnaire de dépendances de paquets. Il est associé à un gestionnaire de paquetage `Packagist` qui lui maintient en dépôt (`git`) tous les paquets utilisables avec Composer.

Propel dépend du projet Phing qui est un portage de Ant en PHP. Ant, make, Phing sont des constructeurs de projet qui permettent d'automatiser les tâches de fabrication d'une application.

#### 7.2.2 Installation

```
cd public_html/  
mkdir Propel  
cd Propel  
curl -sS https://getcomposer.org/installer | php  
xemacs composer.json  
{  
    "require": {  
        "propel/propel1": "~1.7"  
    }  
}  
  
php composer.phar install
```

La commande `curl` permet de télécharger l'exécutable `php composer.phar`. Puis on édite un fichier JSON indiquant ce que l'on veut installer et enfin on installe le paquet `propel` et les paquets dont il dépend. On obtient alors la sous-arborescence suivante :

```

vendor
  bin
  composer
  phing
  propel
  propel1
    generator
    runtime
    test

```

### Installation sous Wamp

Wamp est une distribution Apache, MySQL, Php pour Windows. L'installation est très semblable mais par défaut l'extension openssl n'est pas activée dans le fichier de configuration (php.ini) de l'interprète php en ligne de commande (Command Line Interpreter). Ce fichier se situe généralement en :

```
C:\wamp\bin\php\php5.4.xx\php.ini
```

Il faut alors décommenter la ligne `extension=php_openssl.dll`.

## 7.3 Fondements de l'utilisation : opérations C.R.U.D.

C.R.U.D. (Create, Retrieve, Update, Delete) sont des opérations de base sur les tables. Nous allons voir grâce à des exemples comment les réaliser grâce à des méthodes PHP.

### 7.3.1 Création de ligne

#### Exemple 28 (Création)

```

<?php
/* initialize Propel, etc. */
$author = new Author(); // nouvel objet PHP
$author->setFirstName('Jane'); // affectation de ses attributs : firstName, ...
$author->setLastName('Austen');
$author->save(); // réalise la requête SQL suivante
// INSERT INTO author (first_name, last_name) VALUES ('Jane', 'Austen');

```

*Attention, le nom des attributs de la table correspond au nom des accesseurs et mutateurs qui sont au format "UpperCamelCase". Les attributs eux peuvent utiliser les "underscore" plus classiques en bases de données : `FirstName` correspond donc à `first_name`.*

#### Exemple 29 (Lecture)

```

<?php
echo $author->getId(); // 1
echo $author->getFirstName(); // 'Jane'
echo $author->getLastName(); // 'Austen'

```

*Remarquons que l'attribut `id` a été créé automatiquement lors du `save` car cet attribut est auto-incrémenté.*

*D'autres méthodes existent pour lire tous les attributs d'un objet dans un format spécifique : `toArray()`, `toXML()`, `toYAML()`, `toJSON()`, `toCSV()`, `__toString()`. On peut bien entendu importer dans un objet une chaîne formatée : `fromArray()`, `fromXML()`, `fromYAML()`, `fromJSON()`, and `fromCSV()`.*

### 7.3.2 Recherche de lignes

L'hydratation d'objets (1 ou plusieurs) grâce à une consultation (SELECT) de la BD est une des opérations les plus courantes (Retrieve).

#### Exemple 30 (Hydratation d'1 objet grâce à la clé primaire)

```

<?php
$q = new AuthorQuery(); // objet de consultation
$firstAuthor = $q->findPK(1); // $firstBook est soit NULL, soit l'auteur de clé 1

```

*Si la clé primaire est composée de plusieurs colonnes, la méthode `findPK` contiendra autant de paramètres.*

Une version raccourcie utilise la méthode statique `create` de `XXXQuery` qui retourne un objet de consultation (patron `Factory`).

```
$firstAuthor = AuthorQuery::create()->findPK(1);
```

On peut aussi récupérer plusieurs lignes grâce à leurs clés primaires :

```
$selectedAuthors = AuthorQuery::create()->findPKs(array(1,2,3,4,5,6,7));
// $selectedAuthors is a collection of Author objects
```

D'autres recherches peuvent être effectuées sur d'autres critères que la clé primaire.

### Exemple 31 (Sélection de lignes)

La méthode `find` de l'objet `XXXQuery` permet par défaut de récupérer toutes les lignes de la table `XXX`.

```
<?php $allAuthors = AuthorQuery::create()->find(); // collection de tous les auteurs
    foreach($authors as $author) { echo $author->getFirstName();}
```

Pour ajouter une condition (`WHERE`) sur une colonne, il suffit d'utiliser la méthode `filterByCCC(valeur)`, où `CCC` est le nom de la colonne.

```
$authors = AuthorQuery::create()
->filterByFirstName('Jane') // where name='Jane'
->orderByTitle()           // order by title
->limit(10)                 // les 10 premières
->find();
```

Bien entendu, les méthodes de filtrage peuvent être chaînées et elles acceptent des conditions complexes :

- méta-caractères (%?) pour les chaînes (`LIKE`);
- intervalle pour les champs numériques et temporels si la condition est un tableau associatif de clé min et max;
- ensemble de valeurs pour les champs entiers si la condition est un tableau d'entiers;
- une date, une estampille ou un objet PHP `DateTime` pour les champs temporels.

`find` retourne toujours une collection d'objets sur lesquels on peut itérer (`foreach`). Si l'on ne souhaite qu'une seule ligne (la première), il faut utiliser `findOne`.

Il existe bien sûr une méthode `filterBy` pour chaque clé étrangère et celle-ci admet une valeur correspondant à la clé ou bien un objet PHP correspondant à la ligne étrangère. On peut enfin, réaliser des requêtes imbriquées en utilisant `useXXXQuery()` à l'intérieur d'une requête ...

Enfin, de nombreuses autres méthodes de `XXXQuery` existent permettant de faire :

- jointure (`join`);
- critères complexes (`where`);
- récupération d'objet lié par une Contrainte d'intégrité référentielle (clé étrangère) : `joinWith('Book.Author')`;

### 7.3.3 Mise à jour (Update)

La mise à jour de la BD peut être effectuée via la recherche, la mise à jour d'objet suivie de leur sauvegarde (`find`, `set`, `save`). Mais on peut également utiliser la méthode `update` pour agir directement sur la BD.

```
$author = AuthorQuery::create()->findOneByFirstName('Jane'); // récup. 1 objet
$author->setLastName('Austen'); // māj
$author->save(); // sauvegarde
```

```
AuthorQuery::create() // sans passer par les objets
->filterByFirstName('Jane')
->update(array('LastName' => 'Austen'));
```

### 7.3.4 Suppression (Delete)

Comme la mise à jour, la suppression peut être effectuée via des objets ou directement sur la BD.

```
$author = AuthorQuery::create()->findOneByFirstName('Jane');
$author->delete();
```

```
AuthorQuery::create()
->filterByFirstName('Jane')
->delete();
```

## 7.4 Construction d'un projet (build)

Le générateur propel utilise un **fichier central** `schema.xml` décrivant le schéma de la BD relationnelle : database, table, column, foreign-key, unique, index, ... Ce schéma peut :

- soit être écrit à la main ;
- soit être fabriqué à partir d'une BD existante ;
- soit être initialisé à partir d'une BD existante puis modifié à la main !

L'exemple suivant illustre la BD utilisée tout au long de ce chapitre.

### Exemple 32 (fichier Propel/schema.xml)

```
<?xml version="1.0" encoding="UTF-8"?>
<database name="bookstore" defaultIdMethod="native">
  <table name="book" phpName="Book">
    <column name="id" type="integer" required="true" primaryKey="true" autoIncrement="true"/>
    <column name="title" type="varchar" size="255" required="true" />
    <column name="isbn" type="varchar" size="24" required="true" phpName="ISBN"/>
    <column name="publisher_id" type="integer" required="true"/>
    <column name="author_id" type="integer" required="true"/>
    <foreign-key foreignTable="publisher" phpName="Publisher" refPhpName="Book">
      <reference local="publisher_id" foreign="id"/>
    </foreign-key>
    <foreign-key foreignTable="author">
      <reference local="author_id" foreign="id"/>
    </foreign-key>
  </table>
  <table name="author" phpName="Author">
    <column name="id" type="integer" required="true" primaryKey="true" autoIncrement="true"/>
    <column name="first_name" type="varchar" size="128" required="true"/>
    <column name="last_name" type="varchar" size="128" required="true"/>
  </table>
  <table name="publisher" phpName="Publisher">
    <column name="id" type="integer" required="true" primaryKey="true" autoIncrement="true" />
    <column name="name" type="varchar" size="128" required="true" />
  </table>
</database>
```

*Quelques remarques concernant cet exemple :*

- l'attribut `defaultIdMethod` indique la façon dont les identifiants numériques auto-incrémentés doivent l'être : `native` signifie que c'est le SGBD qui gère ;
- les attributs `phpName` ne sont pas nécessaires puisque par défaut propel générera des noms de classes Camel-Case !
- les contraintes d'intégrité de clé primaire et référentielles sont précisées elles aussi ;
- les types des colonnes peuvent être : `boolean`, `tinyint`, `smallint`, `integer`, `bigint`, `double`, `float`, `real`, `decimal`, `char`, `varchar`, `longvarchar`, `date`, `time`, `timestamp`, `blob`, `clob`, `object`, `array` ; ils seront associés aux types natifs du SGBD ;

### 7.4.1 Configuration de construction (`buid.properties`)

Avant de générer la BD et/ou le modèle objet PHP à partir du schéma, il faut définir la configuration d'exécution dans `buid.properties`. C'est un fichier texte contenant des directives (`x.y.z=toto`) dans différentes sections (General Build Settings, Database Settings, Customizing Generated Object Model, ...). Un fichier minimal `Propel/build.properties` suit :

```
# Database driver
propel.database = mysql

# Project name
propel.project = bookstore
```

La **génération du modèle objet** (om comme Object Model) peut maintenant être lancée :

```
vendor/propel/propel1/generator/bin/propel-gen om
```

L'examen des classes PHP générées peut être effectué dans le répertoire `Propel/build/classes/bookstore` qui contient 3 fichiers par objet métier (`book`, `author`, `publisher`). Par exemple, on trouve `Book.php`, `BookQuery.php` et `BookPeer.php` (Peer pour rétrocompatibilité). Ces classes héritent des classes de bases situées dans le répertoire `om` (Object Model) sous-jacent. Elles sont vides et peuvent être spécialisées en y ajoutant des méthodes ad hoc. Lors d'une régénération, seules les classes de base seront reconstruites !

Attention, ce modèle objet ne peut pas être utilisé tant que la base de données n'a pas été créée !

### 7.4.2 Génération de la BD

On peut générer la BD grâce à un script SQL ou bien en demandant à propel de se connecter au SGBD.

Pour générer le script sql de création de la BD :

```
vendor/propel/propel1/generator/bin/propel-gen sql
```

Le fichier généré `Propel/build/sql/schema.sql` est un script de création de la BD que l'on pourrait exécuter en ligne de commande (`mysql`) ou avec `phpMyAdmin`. Un fichier `sqldbmap` est également généré et est nécessaire à la communication avec le serveur.

Une fois le script généré, on peut également demander à Propel de communiquer directement avec le serveur `mysqld` ! Il faut rajouter les paramètres suivant au fichier `build.properties` :

```
# Connection parameters
propel.database.url = mysql:host=venus;dbname=mmeynard
propel.database.user = mmeynard
propel.database.password = ???????
```

Puis on lancera les commandes :

```
vendor/propel/propel1/generator/bin/propel-gen sql
vendor/propel/propel1/generator/bin/propel-gen insert-sql
```

Après cela, la BD a été créée !

**Sous Windows**, il faut utiliser le script `propel-gen.bat` et positionner la variable d'environnement `Path` afin que le script `phing.bat` soit accessible.

### 7.4.3 Rétro-ingénierie

On peut également concevoir le schéma XML à partir d'une base de données existantes. De ce schéma, on pourra ensuite générer le modèle objet.

```
vendor/propel/propel1/generator/bin/propel-gen reverse
```

Si la BD contient des données, on peut également les copier dans un fichier XML :

```
vendor/propel/propel1/generator/bin/propel-gen datadump
```

La création du fichier SQL de création de la BD contenant des données sera alors réalisée par :

```
vendor/propel/propel1/generator/bin/propel-gen datasql
```

### 7.4.4 Utilisation à l'exécution de Propel

Pour utiliser le modèle objet dans des pages PHP, il faut écrire un fichier de configuration XML permettant aux pages de communiquer avec la BD par l'intermédiaire du modèle objet !

`Propel/runtime-conf.xml`

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
<!-- <log>
  <type>file</type>
  <name>/path/to/propel.log</name>
  <ident>propel-bookstore</ident>
  <level>7</level>
</log>
```

```
-->
<propel>
  <datasources default="bookstore">
    <datasource id="bookstore">
      <adapter>mysql</adapter> <!-- sqlite, mysql, mssql, oracle, or pgsql -->
      <connection>
        <dsn>mysql:host=venus;dbname=mmeynard</dsn>
        <user>mmeynard</user>
        <password>??????</password>
      </connection>
    </datasource>
  </datasources>
</propel>
</config>
```

Puis, il faut convertir ce fichier en php (dans build/conf) afin qu'il soit plus rapide à l'exécution :

```
vendor/propel/propel1/generator/bin/propel-gen convert-conf
```

Enfin, il ne vous reste plus qu'à écrire votre premier script `index.php` utilisant Propel :

```
<?php
// Include the main Propel script
require_once("vendor/propel/propel1/runtime/lib/Propel.php");

// Initialize Propel with the runtime configuration
Propel::init("build/conf/bookstore-conf.php");

// Add the generated 'classes' directory to the include path
set_include_path("build/classes" . PATH_SEPARATOR . get_include_path());

$b=new Book();
$b->setTitle("Les Misérables");
$b->setIsbn(1234);

$a=new Author();
$a->setFirstName("Victor");
$a->setLastName("Hugo");
$a->save();
$b->setAuthor($a);

$p=new Publisher();
$p->setName("Belin");
$p->save();
$b->setPublisher($p);
$b->save();
?>
```

### Include path

Le module php d'Apache doit connaître l'emplacement des classes PHP générées. Ceci peut être réalisé de 2 façons :

- soit par modification du fichier `C:\wamp\Apache2\bin\php.ini` avec une ligne `include_path=". ; c:\wamp\php\pear"`
- soit par la fonction PHP `set_include_path("build/classes" . PATH_SEPARATOR . get_include_path());`

On utilisera l'une et/ou l'autre des méthodes selon qu'on a accès ou non à `php.ini` !

## 7.4.5 Erreurs fréquentes

### Nom du projet, de la bd, du DSN

En cas de changement de BD et/ou de SGBD et/ou de projet, il faut bien comprendre quelles modifications effectuer. Les noms suivants doivent être **absolument identiques** :

- dans le fichier `schema.xml`, la racine `<database name="toto">`;
- dans `build.properties`, `propel.project = toto`;
- dans `runtime-conf.xml`, `<datasources default="toto">` et `<datasource id="toto">`

Avec MySQL, créer un couple (utilisateur, lieu) par type d'utilisation : par exemple (meynard, localhost) et (meynard, %).

### Partage de projet via SVN

En cas de partage du code PHP, on peut très bien attaquer 2 bd différentes à condition :

- qu'elles aient le même nom (toto) et le même type;
- de changer directement dans `.../build/conf/toto-conf.php` les paramètres d'accès à la bd : `hostspect`, `username`, `password`;
- ignorer ce fichier ainsi que `runtime-conf.xml` dans les commit SVN;
- à chaque `propel-gen`, il faudra rétablir le fichier local (on l'aura sauvé préalablement)

### makefile

```
PROPEL-GEN=propel-gen.bat
```

```
all:
```

```
# schema.xml puis modele objet php
```

```
make schema
```

```
make om
```

```
schema:
```

```
#genere le schema.xml à partir de la bd définie dans build.properties
```

```
$(PROPEL-GEN) . reverse
```

```
om:
```

```
#genere classes PHP dans répertoire build/classes/meynardprojetinfo13
```

```
$(PROPEL-GEN) . om
```

```
sql:
```

```
#genere script build/sql/schema.sql pour créer la bd
```

```
$(PROPEL-GEN) . sql
```

```
bd:
```

```
#execute le script build/sql/schema.sql pour créer la bd
```

```
$(PROPEL-GEN) . insert_sql
```

```
doc:
```

```
cd Doxygen; make
```





# Chapitre 8

## Doctrine

### 8.1 Introduction

Doctrine 2 est un ORM pour PHP 5.4+. Il est possible de l'utiliser avec les frameworks Symfony 2, Zend Framework, ... Une correspondance objet-relationnel (en anglais object-relational mapper ou ORM) est un patron de conception (design pattern) qui permet de manipuler une base de données relationnelle en utilisant des objets PHP persistants. Ainsi le code PHP est plus homogène, ce qui facilite sa programmation et son débogage. Doctrine 2 est une refondation complète de Doctrine 1 et utilise un grand nombre de patron de conception (Design Pattern). La manipulation des objets PHP (persistance, consultation) est réalisée grâce à un Gestionnaire d'Entités (EntityManager) qui implémente le "data mapper pattern".

Une **entité** est un objet PHP identifié par un attribut unique qui est lié à la clé primaire de la ligne qui lui correspond. Une classe d'entités possède des entités, chaque entité correspondant à une ligne d'une table relationnelle. Quelques caractéristiques :

- génération de la BD à partir de classes d'entités métiers ou construction du modèle objet à partir d'une BD existante (rétro-ingénierie) ;
- pas de schéma XML intermédiaire entre modèle objet et BD ;
- DQL (Doctrine Query Language) inspiré par Hibernate Query Language (ORM Java) ;
- les entités sont liées entre elles par des associations : une association matérialise sur les objets PHP une contrainte d'intégrité référentielle de la BD ;
- des transactions ACID sont présentes ;
- l'actualisation des associations (références à d'autre(s) entité(s)) dans les entités est réalisée de manière **faïnéante** (lazy) ; c'est-à-dire au fur et à mesure des accès ;
- Plusieurs démarches de conception de projet existent :
  - "code first" consiste à suivre la procédure suivante consistant à écrire le code PHP des entités et leurs métadonnées puis à créer la BD ;
  - "database first" permet de construire automatiquement le modèle objet PHP à partir d'une BD existante (rétro-ingénierie) ;
  - "model first" devrait permettre de tout construire (entités et BD) à partir d'un modèle UML ...

Actuellement, seule la démarche "code first" est totalement fonctionnelle. La création programmée d'entités à partir d'une BD existante nécessite de rajouter par la suite certaines associations entre entités oubliées par l'outil.

#### 8.1.1 Installation

Doctrine s'installe, comme Propel, en utilisant **Composer**. On crée le fichier `composer.json` suivant :

```
{
    "require": {
        "doctrine/orm": "*"
    }
}
```

Puis on lance la ligne de commande suivante :

```
php composer.phar install
```

Le paquetage Doctrine installé, on peut avoir besoin d'outils en ligne de commande de la façon suivante :

```
$ php vendor/bin/doctrine
```

## 8.2 Le gestionnaire d'entité et la persistance

La classe `EntityManager` fournit le point d'accès au cycle de vie complet des entités. C'est une classe singleton utilisant le patron de conception `factory` :

```
// obtaining the entity manager
$entityManager = EntityManager::create($conn, $config);
```

Cet objet PHP nous permettra par la suite d'enregistrer les entités devant persister en deux phases :

- `$entityManager->persist($product)`; permet de tamponner l'entité; cette méthode sera utilisée plusieurs fois avant de lancer la transaction de BD qui effectuera l'ensemble des requêtes (insert ou update ou delete).
- `$entityManager->flush()`; lance la transaction sur la BD. Remarquons que toute entité enregistrée (tamponnée) par `persist()` peut être modifiée par la suite, l'état final de l'entité sera enregistré lors du `flush()`.

### 8.2.1 Classe d'entité et métadonnées

Cependant, avant de manipuler les entités, il faut définir :

- la classe d'entité en PHP : ensemble d'attributs protégés ayant une paire d'accessor/mutateur (`get/set`) et un attribut `id` qui est la clé primaire et qui lui ne possède qu'un accessor; (dans le répertoire `src`)
- les métadonnées de la classe qui représentent la liaison entre classe et table et entre attribut et colonne. Ces métadonnées peuvent être fournies en XML, en YAML (XML indenté) ou en PHP grâce à des annotations (`/** @Entity ...`) dans le fichier définissant la classe d'entité.

#### Exemple 33

*Code PHP de la classe d'entité `Product` ayant un `id` et un `nom`*

```
<?php
// src/Product.php
class Product{
    /**
     * @var int
     */
    protected $id;
    /**
     * @var string
     */
    protected $name;

    public function getId(){
        return $this->id;
    }
    public function getName(){
        return $this->name;
    }
    public function setName($name){
        $this->name = $name;
    }
}
```

*Code des métadonnées en YAML*

```
# config/yaml/Product.dcm.yml (dans le répertoire config/yaml)
Product:
  type: entity
  table: products
  id:
    id:
      type: integer
      generator:
```

```

    strategy: AUTO
fields:
  name:
    type: string

```

## 8.2.2 Génération du schéma de BD : la ligne de commande doctrine

Afin de générer le schéma de BD en SQL (CREATE TABLE ...) et de l'exécuter, il faut utiliser la commande *doctrine* suivante depuis un terminal :

```

$ vendor/bin/doctrine orm:schema-tool:create --dump-sql
CREATE TABLE products (id INTEGER NOT NULL, name VARCHAR(255) NOT NULL, PRIMARY KEY(id));

```

On peut vérifier grâce à PhpMyAdmin que la table `products` a bien été créée. En cas de modification du schéma de BD, ajout de classe d'entités ou modifications, il faut penser à chaque fois à mettre à jour le schéma.

```

$ vendor/bin/doctrine orm:schema-tool:update --force

```

L'option *force* permet d'éviter de changer de schéma de BD par mégarde alors qu'on est en environnement de production. En effet, Cela risque de modifier complètement le comportement des scripts.

## 8.2.3 Utilisation de la BD

Afin de tester, nous allons créer un script interactif php ajoutant des produits ayant comme nom les arguments passés à la ligne de commande :

```

<?php
// creerProd.php
require_once "bootstrap.php"; // fichier de configuration et entityManager
$p=array();
for ($i=1; $i<count($argv);$i++){
    $p[$i] = new Product();
    $p[$i]->setName($argv[$i]);
    $entityManager->persist($p[$i]);
    echo "produit enregistré (id,name) = (".$p[$i]->getId().",".$p[$i]->getName().")\n";
}
$entityManager->flush();
echo "Produits insérés en BD !\n";
for ($i=1; $i<count($argv);$i++){
    echo "produit (id,name) = (" . $p[$i]->getId() . ",".$p[$i]->getName() .")\n";
}

```

Examinons une exécution de ce script :

```

$ php creerProd.php toto abc foo
produit enregistré (id,name) = (,toto)
produit enregistré (id,name) = (,abc)
produit enregistré (id,name) = (,foo)
Produits insérés en BD !
produit (id,name) = (9,toto)
produit (id,name) = (10,abc)
produit (id,name) = (11,foo)

```

Remarquons qu'avant le vidage (*flush*) les entités n'ont pas d'identifiant car celui-ci est ensuite fourni par MySQL puisque c'est une clé primaire entière auto-incrémentée.

## 8.3 Consultation des entités (query)

### 8.3.1 EntityRepository (dépôt d'entités)

Afin de manipuler l'ensemble des lignes de la table `products`, on manipule un dépôt (*repository*) depuis l'*entity-Manager*. Puis on lui applique une requête (*findAll*). `listeProduits.php`

```

<?php
// listeProduits.php
require_once "bootstrap.php";

$productRepository = $entityManager->getRepository('Product');
$products = $productRepository->findAll();

foreach ($products as $p) {
    echo sprintf("%d %s<br>\n", $p->getId(), $p->getName(), $p->getId());
}

```

### 8.3.2 Par clé primaire

La méthode `find($entityName, $id)` de l'`entityManager` permet de récupérer une entité par son identifiant. Le formulaire web suivant permet d'afficher un produit :

```

<form method="get">
Id<input type="number" name="id">
<input type="submit" Value="Chercher !">
</form>
<?php
// chercheProduit.php
if(isset($_GET["id"])){
    require_once "bootstrap.php";
    $p = $entityManager->find('Product', $_GET["id"]);
    echo sprintf("%d %s<br>\n", $p->getId(), $p->getName());
}

```

### 8.3.3 Par des conjonctions d'égalité (where name='x' and id=5)

On peut utiliser `findBy()` ou `findOneBy()` sur un dépôt en fournissant un tableau de (clé, valeur) ou chaque clé est un nom d'attribut et chaque valeur l'unique valeur d'attribut accepté. L'exemple du script suivant recherche dans la table produit s'il y a un produit dont la valeur d'id est fourni en premier argument ET dont le name est fourni en second argument.

```

<?php
// prodIdName.php
require_once "bootstrap.php"; // fichier de configuration et entityManager
$p=$entityManager->getRepository('Product')->findBy(array(
'id' => $argv[1],
'name' => $argv[2]
));
if (count($p)){ // au moins un produit (ici 0 ou 1 car id)
    //print_r($p);
    echo "Trouvé ! (" . $p[0]->getId() . ", " . $p[0]->getName() . ")\n";
}else{
    echo "Aucun Produit !\n";
}

```

La méthode `findBy()` permet également :

- de fournir un ensemble de valeurs possibles pour un attribut (id IN (1,2,3)) en indiquant un tableau de valeur (`array()`);
- de trier selon une colonne (ORDER BY) (2nd paramètre);
- de limiter le nb de résultats (LIMIT x) (3eme paramètre);
- de définir un offset (résultats sauf les n premiers) (4eme paramètre);

L'exemple de script suivant affiche tous les produits dont le nom est 'un' ou 'deux' ou 'trois' en ordre décroissant sur les noms, croissant sur les id, en en prenant que 10 au maximum et en ne prenant pas les 2 premiers !

```

<?php
// produndeuxtrois.php
require_once "bootstrap.php"; // fichier de configuration et entityManager

```

```

$ps=$entityManager->getRepository('Product')->findBy(
array('name' => array("un","deux", "trois")), // IN
array('name' => 'DESC', 'id' => 'ASC'),      // ORDER BY
10,          // LIMIT
2           // OFFSET
);
if (count($ps)){ // plusieurs produits
    foreach ($ps as $p) {
        echo sprintf("%d %s\n",$p->getId(), $p->getName());
    } //print_r($ps);
} else {
    echo "Aucun Produit !\n";
}

```

### 8.3.4 Doctrine Query Language (DQL)

Le langage DQL ressemble au SQL mais n'est pas du SQL ! Il permet de créer une requête complexe (`createQuery()`) puis de l'exécuter (`getResult()`) en récupérant un tableau d'entités.

```

<?php
// dql.php
require_once "bootstrap.php"; // fichier de configuration et entityManager
$q=$entityManager->createQuery("select p from Product p where p.id>8 or p.name
    in ('un', 'deux', 'trois') order by p.name asc");
$ps=$q->getResult();
if (count($ps)){ // plusieurs produits
    foreach ($ps as $p) {
        echo sprintf("%d %s\n",$p->getId(), $p->getName());
    } //print_r($ps);
} else {
    echo "Aucun Produit !\n";
}

```

On remarquera que l'on récupère une collection (sorte de tableau) d'entités (`ps`) selon des conditions diverses puis on la parcourt.

## 8.4 Associations entre entités

Les associations entre entités permettent de modéliser les relations existant entre les objets métiers. Elles sont, pour certaines, liées à des contraintes d'intégrité référentielles dans la BD mais **pas toujours**. Par exemple, l'héritage entre classe d'entités ne donnera pas forcément lieu à une relation dans la BD. N'oublions pas que certaines BD n'implémentent pas certaines contraintes d'intégrité.

### 8.4.1 Association unidirectionnelle "one to one"

Cas d'utilisation : un étudiant est un utilisateur, certains utilisateurs ne sont pas étudiant. On indique ci-dessous le code PHP annoté (méta-données) ainsi que le code SQL associé :

```

class Etudiant{
    ...
    /**
     * @OneToOne(targetEntity="User")
     * @JoinColumn(name="user_id", referencedColumnName="id")
     */
    private $user;
    ...
}
CREATE TABLE Etudiant (
    ...,
    user_id INT DEFAULT NULL,

```

```

    UNIQUE INDEX UNIQ_6FBC94267FE4B2B (user_id),
    PRIMARY KEY(id)
) ENGINE = InnoDB;
CREATE TABLE User (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE Etudiant ADD FOREIGN KEY (user_id) REFERENCES User(id);

```

Remarquons l'index **unique** créé sur la colonne `user_id`

### 8.4.2 Association bidirectionnelle "one to one"

Cas d'utilisation : un caddy appartient à un unique client qui ne peut en avoir qu'un au maximum. On veut conserver du côté de l'entité client une référence au caddy (bidirectionnel). On indique ci-dessous le code PHP annoté (méta-données) ainsi que le code SQL associé :

```

class Customer{
    // ...
    /**
     * @OneToOne(targetEntity="Cart", mappedBy="customer")
     */
    private $cart;
}
/** @Entity */
class Cart{
    // ...
    /**
     * @OneToOne(targetEntity="Customer", inversedBy="cart")
     * @JoinColumn(name="customer_id", referencedColumnName="id")
     */
    private $customer;
}
...
CREATE TABLE Cart (
    id INT AUTO_INCREMENT NOT NULL,
    customer_id INT DEFAULT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;
CREATE TABLE Customer (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE Cart ADD FOREIGN KEY (customer_id) REFERENCES Customer(id);

```

Ici, la table propriétaire de clé étrangère (`customer_id`) est le caddy. Au niveau des entités, elles sont symétriques en possédant chacune une référence à l'autre. Au niveau des méta-données, c'est l'attribut `inversedBy` qui désigne le propriétaire tandis que sa propriété le désigne par `"mappedBy"`. L'entité propriétaire est importante car au moment du **flush**, la sauvegarde en BD se basera sur les attributs des propriétaires.

### 8.4.3 Association unidirectionnelle many-to-one

Plusieurs utilisateurs (many) peuvent partager la même adresse. On indique ci-dessous le code PHP annoté (méta-données) ainsi que le code SQL associé :

```

class User{
    ...

```

```

/**
 * @ManyToOne(targetEntity="Address")
 * @JoinColumn(name="address_id", referencedColumnName="id")
 **/
private $address;
}
...
CREATE TABLE User (
    ...,
    address_id INT DEFAULT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;
CREATE TABLE Address (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE User ADD FOREIGN KEY (address_id) REFERENCES Address(id);

```

Remarquons que cette cas étant extrêmement fréquent, la ligne `@JoinColumn` n'est pas nécessaire si le nommage des attributs et des colonnes suit la convention. Remarquons encore que l'adresse est optionnelle (NULL).

#### 8.4.4 Association bidirectionnelle one-to-many

Un produit possédant plusieurs caractéristiques aura une propriété `oneToMany` "features" qui sera forcément en correspondance (`mappedBy`) par une propriété `product` `manyToOne` dans l'entité `Feature`. Remarquons que la contrainte d'intégrité référentielle sera dirigée depuis `feature(product_id)` vers `product(id)`. A l'inverse, la propriété "features" de l'entité `Product` sera une collection de caractéristiques (`Doctrine\Common\Collections\Collection`).

```

class Product{
    // ...
    /**
     * @OneToMany(targetEntity="Feature", mappedBy="product")
     **/
    private $features;
    // ...

    public function __construct() {
        $this->features = new ArrayCollection();
    }
    ...
class Feature{
    // ...
    /**
     * @ManyToOne(targetEntity="Product", inversedBy="features")
     * @JoinColumn(name="product_id", referencedColumnName="id")
     **/
    private $product;
    // ...
}

```

#### 8.4.5 Association unidirectionnelle many-to-many

Sans être exhaustif, le cas suivant illustre des utilisateurs affiliés à différents groupes d'utilisateurs. Dans la BD, une table de jointure (non modélisée par une entité!) permettra de représenter les associations multiples de groupes et d'utilisateurs.

```

class User{
    // ...
    /**
     * @ManyToMany(targetEntity="Group")
     * @JoinTable(name="users_groups",
     *           joinColumns={@JoinColumn(name="user_id", referencedColumnName="id")},

```

```

*      inverseJoinColumn=@JoinColumn(name="group_id", referencedColumnName="id")}
*      )
**/
private $groups;
// ...
public function __construct() {
    $this->groups = new \Doctrine\Common\Collections\ArrayCollection();
}
...
class Group{
    // ...
}

```

SQL :

```

CREATE TABLE User (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;
CREATE TABLE users_groups (
    user_id INT NOT NULL,
    group_id INT NOT NULL,
    PRIMARY KEY(user_id, group_id)
) ENGINE = InnoDB;
CREATE TABLE Group (
    id INT AUTO_INCREMENT NOT NULL,
    PRIMARY KEY(id)
) ENGINE = InnoDB;
ALTER TABLE users_groups ADD FOREIGN KEY (user_id) REFERENCES User(id);
ALTER TABLE users_groups ADD FOREIGN KEY (group_id) REFERENCES Group(id);

```

### 8.4.6 Héritages

Plusieurs héritages sont techniquement possibles grâce aux associations Doctrine :

- Mapped Superclass : permet de définir les attributs et associations communs dans une superclasse qui ne donnera pas lieu à création de table mais chaque entité fille donnera une table contenant les champs de sa superclasse ;
- Single Table Inheritance : dans ce patron, toute la hiérarchie est représentée dans une unique table qui contient un champ discriminant la classe. Les champs spécifiques à chaque sous-classe doivent absolument être NULL puisqu'ils existeront pour tout le monde !
- Class Table Inheritance : chaque classe fille correspondra à une table la représentant et autant de tables que nécessaire pour représenter ses ancêtres avec des clés étrangères les reliant ;
- Overrides : permet d'indiquer qu'un attribut ou une association de la classe fille prend le pas sur un attribut ou une association de la super-classe (valable dans le cadre de Mapped Superclass).

Remarquons encore que l'association OneToOne uni ou bi-directionnelle permet de modéliser une sorte d'héritage : cependant les attributs communs seront dupliqués dans chaque table ...

## 8.5 Rétro-ingénierie

### 8.5.1 Métadonnées

Afin de fabriquer les métadonnées associées à une BD préexistantes, on peut utiliser la commande doctrine afin de générer un fichier de métadonnées yml, xml ou même php (à éviter) :

```
$ vendor/bin/doctrine orm:convert-mapping --from-database yml .
```

Cela produira un fichier `Product.dcm.yml` dans le répertoire courant (.) dont le contenu suit :

```

Products:
  type: entity
  table: products

```



```

id:
  id:
    type: integer
    nullable: false
    unsigned: false
    id: true
    generator:
      strategy: IDENTITY
fields:
  name:
    type: string
    nullable: false
    length: 255
    fixed: false
  lifecycleCallbacks: { }

```

Attention, cette technique ne permet pas de récupérer :

- les associations inverses (mapped by) ;
- les héritages ;
- les clés étrangères qui sont aussi clés primaires ;
- les cascades ;

De plus, cette technique créera des métadonnées d’entités à partir des tables de jointures (users-groups) et ne définira que des associations ManyToOne (clé étrangères). Par conséquent, il faudra ensuite reprendre les entités créées afin qu’elles correspondent aux fonctionnalités souhaitées par l’application !

## 8.5.2 Retour d’expériences

Dans un schéma de BD dense (beaucoup de clés étrangères), doctrine va tenter de créer toutes les associations bi-directionnelles possibles ce qui crée une erreur dans le cas où le même nom de propriété existe dans 2 tables connexes : “Property "tournoi" in "Equipe" was already declared, but it must be declared only once”. En effet, Equipe est reliée à tournoi via la table inscription (en début de tournoi) mais aussi via la table classement (en fin de tournoi) ! Il faut donc renommer certaines colonnes : tournoi\_id devient tournoi2\_id ! De plus, chaque table doit avoir une clé primaire. Les tables de jointures ne donnent pas lieu à création d’entité (pas d’entité Inscription!). La création des méta-données en php ne permet pas de créer les entités (préférer yml). Des tables de jointures possédant des colonnes ne donnent pas lieu à création d’entités donc on perd ces propriétés !

## 8.5.3 Génération des entités

Attention, le “convert-mapping” ne permet que de créer les métadonnées : afin de créer les fichiers php définissant les classes d’entités, il faut utiliser la commande “generate-entities” qui utilise les méta-données (quelque soit leur format (xml, yml, php)) et quelle que soit la façon dont elles ont été produites, manuellement ou automatiquement (convert-mapping) :

```
$ vendor/bin/doctrine orm:generate-entities src/
```

Cette commande va créer le fichier Product.php ou le modifier afin d’ajouter attributs **privés** pour les références aux objets liés, accesseurs et mutateurs. Encore une fois, il est préférable de créer les entités à la main si l’on veut spécifier de l’héritage entre classes ou définir des méthodes spécifiques métier.

## 8.5.4 Comparaison Propel/Doctrine

Alors que le fichier xml de Propel décrit précisément la BD relationnelle et que les classes créées dans le modèle objet sont très proches de celles-ci, Doctrine invite le développeur à créer ses classes métiers sans penser forcément à l’implémentation en terme de BD. Ensuite, ce sont les méta-données qui permettront de relier les entités à la BD.

# 8.6 Les fichiers de configuration

## 8.6.1 Installation

L’installation est réalisée via composer :

```

cd public_html/
mkdir DctExemple
cd DctExemple
curl -sS https://getcomposer.org/installer | php
xemacs composer.json
{
    "require": {
        "doctrine/orm": "2.4.*",
        "symfony/yaml": "2.*"
    },
    "autoload": {
        "psr-0": {"": "src/"}
    }
}
php composer.phar install

```

Un répertoire `DctExemple/vendor` est créé qui contient :

```

autoload.php  qui permet le chargement automatique des classes d'entités
bin           qui contient des liens vers des exécutable php
composer      qui contient les fonctionnalités de composer
doctrine     qui contient les fonctionnalités de doctrine
symfony      qui contient les fonctionnalités de Yaml

```

Le répertoire `bin/vendor` contient notamment la commande `doctrine`. Il ne reste plus qu'à créer les fichiers `bootstrap.php` et `cli-config.php`.

## 8.6.2 bootstrap.php

Le fichier `bootstrap.php` suivant contient 3 sections après l'utilisation des espaces de noms et le lancement du mécanisme d'autoload :

- configuration des métadonnées liant entités et tables soit en annotations PHP, soit en XML, soit en YAML (notre choix);
- définition de la configuration d'accès à la BD qui sera utilisée par DBAL (DataBase Abstraction Layer) puis par PDO puis par MySQL (dans notre cas);
- création de l'`entityManager`;

```

<?php
// bootstrap.php
use Doctrine\ORM\Tools\Setup;
use Doctrine\ORM\EntityManager;

require_once "vendor/autoload.php";

// Create a simple "default" Doctrine ORM configuration for Annotations
$isDevMode = true;
// $config = Setup::createAnnotationMetadataConfiguration(array(__DIR__."/src"), $isDevMode);
// or if you prefer yaml or XML
// $config = Setup::createXMLMetadataConfiguration(array(__DIR__."/config/xml"), $isDevMode);
// METADATA YAML
$config = Setup::createYAMLMetadataConfiguration(array(__DIR__."/config/yaml"), $isDevMode);

// database configuration parameters
// $dsn = array(
//     'driver' => 'pdo_sqlite',
//     'path' => __DIR__ . '/db.sqlite',
// );
$dsn = array(
    'dbname' => 'tempo',
    'user' => 'tempo',
    'password' => 'tempo',

```

```

    'host' => 'localhost',    // '127.0.0.1' ne fonctionne pas !
    'driver' => 'pdo_mysql',
    'charset' => 'utf8' ,    // sinon les char utf-8 ne passent pas en BD
);
// obtaining the entity manager
$entityManager = EntityManager::create($dsn, $config);

```

### 8.6.3 Ligne de commande doctrine

Pour exécuter les commandes `doctrine`, il est nécessaire de définir un fichier de configuration de l'interpréteur en ligne de commande désigné par `cli-config.php` :

```

<?php
// cli-config.php
require_once "bootstrap.php";

return \Doctrine\ORM\Tools\Console\ConsoleRunner::createHelperSet($entityManager);

```

## 8.7 Bugs et astuces

### 8.7.1 MAMP spécifique

Il faut créer un lien symbolique vers la socket mysql de MAMP depuis le répertoire où l'interprète en ligne de commande l'attend (`php -i`). `$ sudo ln -s /Applications/MAMP/tmp/mysql/mysql.sock /var/mysql/mysql.sock` On pourrait également modifier le `php.ini` !

### 8.7.2 Ligne de commande

Attention à distinguer l'interprète php de la ligne de commande (celui qui exécute les commandes doctrine) et celui du serveur Apache : ils n'ont pas forcément la même version ni les mêmes modules installés et pas le même `php.ini` !

- `which php` renverra `/usr/bin/php`
- `php -i` affichera les infos (`phpinfo()`)
- `php -ini` indique la localisation du `php.ini` (`/etc/`)
- `php -interactive` permet de tester après avoir tapé `<?` et terminé par `Ctrl-D`;

## 8.8 DBAL DataBase Access Layer

Doctrine 2 fournit DBAL qui est la couche entre l'ORM et PDO. DBAL permet notamment de contrôler les limites d'une transaction si l'on veut être plus précis qu'avec l'utilisation de la méthode `flush()` pour valider ses modifications.

```

<?php
// $em instanceof EntityManager
$em->getConnection()->beginTransaction(); // suspend auto-commit
try {
    //... do some work
    $user = new User;
    $user->setName('George');
    $em->persist($user);
    $em->flush();
    $em->getConnection()->commit();
} catch (Exception $e) {
    $em->getConnection()->rollback();
    throw $e;
}

```

Remarquons que dans cet exemple, c'est le `beginTransaction()` qui supprime le mode auto-commit par défaut de PDO. Par conséquent, ensuite le `flush` ne suffira pas pour valider et il faudra le `commit`.

Enfin, DBAL permet également de manipuler des verrous (lock) sur les tables.

## 8.9 Conclusion

L'utilisation d'un ORM tel que Doctrine ne peut être envisagé que dans le cadre d'un développement à plein temps d'un projet. En effet, le nombre de concepts employés ainsi que le vocabulaire utilisé nécessite un long temps d'apprentissage. Intégré dans Symfony 2, un framework PHP complet, Doctrine 2 semble avoir pris le pas sur les autres ORMs PHP. De plus, ses concepts proviennent d'ORM leaders dans d'autres langages (Hibernate en Java) et son apprentissage restera utile.

# Bibliographie

- [1] *Programmation HTML et Javascript*, de Chaléat, Charnay, Eyrolles (2000), “simple et rapide pour les bases HTML, CSS, JavaScript, 450 pages”
- [2] *Webmaster in a nutshell*, de S. Spainhour, R. Eckstein, O’Reilly (2003), “Manuel de référence rapide pour HTML, CSS, JavaScript, PHP, XML, 550 pages, en anglais”
- [3] *Spécification HTML*, <http://www.w3.org/TR/REC-html40/>, W3C (1999), “La spécification complète en anglais”
- [4] *Un site de documentation sur les technologies du web*, <http://www.w3schools.com/>, “Très complet et en anglais (XHTML, CSS, JavaScript) ”
- [5] *Le manuel PHP en français*, <http://fr3.php.net/manual/fr/>, “La référence ”
- [6] *Site PEAR*, <http://pear.php.net/>, “Le site Web de PEAR”
- [7] *Site WAMP*, <http://www.wampserver.com/>, “Le site Web de WAMP (Apache, MySQL, PHP pour Windows)”
- [8] *Site Creole*, <http://creole.phpdb.org/trac/>, “Le site Web de Creole, une couche d’abstraction de différentes bases de données relationnelles”
- [9] *Site Propel*, <http://propel.phpdb.org/trac/>, “Le site Web de Propel, une couche d’abstraction du relationnel par l’objet”



# Solutions des exercices

## Solution 1 — multiplication.html

```
<html><head><title>Multiplication</title></head><body>
<h1>Multiplication</h1>
<form action="multiplication.php" method="get">
X<input type="number" name="x" size="10"><br>
Y<input type="number" name="y" size="10"><br>
<input type="submit" value="Multiplier !" name="mult">
</form>
</body></html>
```

## — multiplication.php

```
<html><head><title>Multiplication</title></head><body>
<h1>Multiplication</h1>
<?php
    echo "Résultat {$_GET['x']} * {$_GET['y']} = " . $_GET['x'] * $_GET['y'] . " !";
?>
</body></html>
```

## Solution 2 mult.php

```
<html><head><title>Multiplication</title></head><body>
<h1>Multiplication</h1>
<?php
if (isset($_GET['mult'])) {
    echo "Résultat {$_GET['x']} * {$_GET['y']} = " . $_GET['x'] * $_GET['y'] . " !<br/>";
    echo "<hr/> Nouvelle Multiplication :<br/>";
}
?>
<form action="" method="get">
<label for="x">X</label><input type="number" name="x" id="x" size="10"><br>
<label for="y">Y</label><input type="number" name="y" id="y" size="10"><br>
<input type="submit" value="Multiplier !" name="mult">
</form>
</body></html>
```