Le langage PHP4

Ce tutoriel complet sur PHP scindé en deux parties présente tout les aspects de PHP avec Mysql. La première partie introduit PHP4 avec l'affichage classique et la deuxième partie aborde l'aspect Objet de PHP avec les classes, la composition, l'héritage, la sérialisation…. etc.

Il n'est pas inutile de rappeler que les scripts PHP s'exécutent coté serveur. Pour cela il suffit de se connecter avec un serveur par exemple un serveur Apache intégré à un environnement Wamp, Xamp ou EasyPHP (ces 3 produits sont libres et gratuits en téléchargement sur internet)

D'autre part il faut se connecter à un serveur de base de données, je vous propose Mysql 5.0 en mode console également libre.

Pour un intérêt pédagogique, il est préférable de travailler avec Mysql pour la création et l'administration de la base de données, évidemment vous pouvez administrer avec Wamp, dont la simplicité est déconcertante mais où est l'intérêt didactique ? En plus Mysql existera dans 10 ans, qu'en sera t-il de Wamp ?
MySql.5.0 permet de tester ses tables, ce qui est beaucoup plus difficile à faire avec Wamp.

<u>En résumé : 3 entités pour la réalisation d'un projet</u>
1)1 Serveur Apache avec Wamp (gratuit)
2)1 Serveur Mysql avec Mysql 5.0 pour la base de données (gratuit)
3) Un simple éditeur de texte comme le notepad++ ou le bloc notes suffit.

<u>Installation de MySQL 5.0</u>
L'installation ne présente aucune difficulté. Vous allez créer un mot de passe à l'installation, ne pas l'oublier vous en aurez besoin pour vous connecter.
Pour vous connecter Démarrer /tous les programmes/MySql/Mysql server 5.0 / Mysql/Mysql Command line Client.

L'installation de Wamp ne présente aucune difficulté. Lancer l'exécutable faites « suivant » à chaque fois. Une fois installé, dans la barre des taches vous avez une icône à coté de l'horloge que vous double cliquez pour lancer différents services.
Si cette icône apparaît et est jaune avec un clignotant rouge, ce sont les différents services qui ont démarré.Si cette icône n'apparaît pas dans la barre des tâches, la lancer avec Démarrer /tous les programmes/Wampserver/démarrer Wampserver.
Si les services ont démarré vous avez une connexion avec le serveur qui sera effective avec la création de 2 méthodes de connexion de votre code.(Voir tutoriel)
Sous Wamp ou EasyPHP mettre ses fichiers de code dans www (c:\Program Files\Wamp\www) lui indiquer cette adresse sous Dreamweaver. Egalement la machine

est locale, vous êtes en localhost. Pour lancer le navigateur http://localhost/site1/ (site1 est par exemple le nom de votre dossier de travail qui est dans www ou htdocs) Voir la doc de dreamweaver et celle de Wamp qui sont en français.

Il n'est pas nécessaire d'avoir une connexion internet pour effectuer toutes ces opérations que ce soit pour le serveur Apache PHP ou pour le serveur Base de données Mysql 5.0. On travaille en local.

Partie 1
Afficher une phrase ou une image
Afficher la date et l'heure
PHP dans du code HTML
La concaténation
Récupérer les valeurs d'un formulaire
Les structures de contrôle
Ecrire et lire dans un fichier texte
Les fonctions utilisateurs
Les variables d'environnement
Quelques fonctions utiles
SQL/MySQL (Create, Alter & Drop)
SQL/MySQL (Insert et Select)
SQL/MySQL (Delete et Update)
SQL/MySQL (Where)
Fonctions PHP pour MySQL
Interroger une table MySQL
Alimenter une ou plusieurs tables MySQL
Les pseudos-frames
Les sessions PHP4
Affichage page par page
Images dynamiques
Les messages d'erreur
Variables globales à OFF
Les variables dynamiques
Partie 2
Les array/templates
Les cookies
Les fonctions
Classes et Objets I
Classes et Objets II
Classes et Objets III
Les mises au point

1)Définition de PHP

PHP est un langage de programmation qui s'intègre dans vos pages **html**. Il permet entre autres de rendre automatiques des tâches répétitives, notamment grâce à la communication avec une base de données (utilisation la plus courante de **PHP**). Mais, avant de continuer, il est bon d'expliquer comment se déroule **une requête http** : en clair que se passe-t-il lorsque vous tapez une adresse dans votre navigateur, où interviennent **PHP et MySQL** (**la base de données**), et les conséquences que cela implique.

Voici, en simplifiant, ce qu'il se passe, lorsque vous consultez une **page html** dite statique :

- Votre navigateur envoie l'adresse **URL (Uniform Ressource Locator)** que vous avez tapée
- Le **serveur web** est un ordinateur présent sur l'Internet et qui héberge la page que vous demandez. Sur ce serveur on trouve **Apache**, logiciel apte à traiter les requêtes **HTTP** que vous envoyez lorsque vous demandez une page web. **Apache** va donc chercher le fichier demandé dans son arborescence et renvoie à votre navigateur la **page html.**
- Votre navigateur interprète les différents langages se trouvant dans ce fichier (**HTML, JavaScript, CSS, etc**.) et affiche la page.

Maintenant, voyons ce qui se passe lorsque **votre page HTML** contient du code PHP :

- Votre navigateur envoie l'adresse que vous avez tapée.
- Le serveur web cherche dans son arborescence si le fichier existe, et si celui-ci porte une extension reconnue comme une **application PHP (.PHP, .PHP3, .PHP4** par exemple). Si c'est le cas, le serveur web transmet ce fichier à **PHP**.
- **PHP** parse le fichier, c'est-à-dire qu'il va analyser et exécuter le code **PHP** qui se trouve entre les balises **<?php et ?>.** Si ce code contient des requêtes vers une base de données **MySQL, PHP** envoie la requête **SQL**. La **base de données** renvoie les informations voulues au script qui peut les exploiter (pour les afficher par exemple).
- **PHP** continue de parser la page, puis retourne le fichier dépourvu du **code PHP** au serveur web.
- Le **serveur web** renvoie donc un fichier ne contenant plus de PHP, donc seulement du **html** au navigateur qui l'interprète et l'affiche.

Vous remarquez donc que **PHP** s'exécute côté serveur. Il n'y a plus aucune trace du **code PHP** lorsque vous regardez le code source de la page dans votre **navigateur PHP**.

La base de données la plus couramment utilisée avec **PHP** est sans aucun doute **MySQL**. A quoi sert une base de données ? Lorsque vous allez produire des informations dans votre **script PHP**, vous devez les stocker quelque part. Si ce n'est pas le cas, elles seront alors perdues lorsque le serveur renverra la **page html au client (votre navigateur)**. Pour les stocker, il existe deux solutions: la première consiste à

les enregistrer dans un fichier texte sur le serveur (quelque part dans l'arborescence de votre hébergement), la seconde à les enregistrer dans une base de données. La sauvegarde dans un fichier texte n'est pas l'idéal, notamment lorsque vous souhaitez chercher, modifier ou supprimer une partie de l'information que vous stockez. Les bases de données ont été conçues dans cette optique-là. Vous verrez dans quelques exercices comment SQL permet de traiter l'information.

Utiliser PHP sur son ordinateur :

Pourquoi **installer PHP sur son ordinateur** ? Pour tester vos **scripts PHP**, vous allez être amené à les envoyer sur votre hebergeur, sur Internet. Cependant il devient vite très lourd de sans cesse renvoyer ces fichiers par **FTP**. C'est pourquoi installer un serveur web sur son ordinateur est utile, et permet de tester ses scripts plus souplement. Concrètement, votre ordinateur sera à la fois client et serveur. Ainsi vous pourrez programmer en **PHP** sans avoir besoin d'être connecté à Internet, ce qui peut être utile pour les personnes ne disposant pas de connexions illimitées. Alors que pour tester des pages web html en local, il suffit d'ouvrir le fichier dans un navigateur, il faut un serveur web sur votre PC local pour tester **une page PHP**. Pour cela, il existe plusieurs utilitaires très pratiques qui installeront Apache, le serveur web le plus utilisé, **PHP, MySQL**. Leur installation et son utilisation sont très simples et détaillées. Les plus connus sont :

- **EasyPHP** : vous le trouverez sur **www.easyPHP.org**
- **WAMP**
- et le plus léger, mais très fonctionnel **MOV'AMP**

Si vous êtes sous **linux, la plupart des distributions (RedHat, Mandrake, etc ...)** installent par défaut **Apache, PHP et MySQL**. Un article sur Lea-linux indique comment installer **Apache, PHP et Mysql** si cela n'a pas été fait. Il se trouve ici. De plus, une application similaire à **EasyPHP** existe pour Linux, il s'agit de **Linux Easy Installer.**

Les outils pour faire du PHP

Pour faire du PHP, il ne vous faut rien d'autre qu'un simple éditeur de texte. Vous en trouverez une liste ici ainsi qu'un comparatif plutôt complet ici.

Mais l'outil le plus indispensable est certainement la documentation officielle. Celle-ci est disponible ici. Elle contient la description complète des fonctions, la syntaxe de **PHP**, comment l'installer. Enfin sachez utiliser ces merveilleux outils que sont les moteurs de recherche, et plus particulièrement Google. En cherchant bien vous trouverez facilement des ressources (documentation, tutorial, article, ...) dont vous aurez besoin. Le but pour vous est de pouvoir être autonome lorsque vous developperez et de savoir se débrouiller seul face à un problème.

Notions essentielles

Il est très important de comprendre ce qui suit :

- Il faut bien distinguer le **client et le serveur** (imaginez tout bêtement la scène dans un bar). **Votre navigateur est le client**. C'est lui qui demande la page web que vous avez entrée. Le serveur est l'ordinateur sur l'Internet qui héberge cette page web. **PHP** s'exécute donc côté serveur. Cependant, quand **PHP** envoie une **requête SQL** au serveur **MySQL**, il est alors client)
  En voici quelques conséquences :
  - Tout ce qui a trait à la présentation de la page (couleur du texte, etc..) est à faire en **HTML et CSS**, exécutés côté client. PHP n'a rien à voir avec le design de votre page
  - Tout ce qui touche au comportement du navigateur est du domaine du **JavaScript**, lui aussi exécuté par le client
  - L'interêt de PHP est de générer du HTML ou du Javascript dynamiquement. Le travail effectué avec PHP sur votre page est totalement invisible pour le visiteur.
- Le **SQL est un langage a part entière de PHP**, il ne faut surtout pas confondre les deux. C'est **MySQL** qui parse (c'est à dire analyse et exécute) votre **code SQL**, PHP ne fait qu'envoyer une requête au **serveur MySQL.**
- **PHPMyAdmin** n'est pas une base de données. Il s'agit simplement d'un script PHP qui permet d'administrer vos bases de données **MySQL.**

Exemples et contre-exemples

Voyons maintenant des contre-exemples :

- Je souhaite qu'un **pop-up s'ouvre** quand la page se charge. Ici, **PHP** n'est pas la solution. Vous devez utiliser un script qui soit exécuté par le navigateur car c'est lui qui gère l'ouverture des **pop-up**. Le **langage JavaScript** est fait pour cela.
- Je souhaite connaître la résolution de l'écran du visiteur afin de faire des statistiques.**PHP** ne peut pas gérer celà car il s'exécute côté serveur.
- Je souhaite faire un chat en **PHP**. En effet il n'est pas impossible de réaliser un script de chat, il en existe d'ailleurs plusieurs. Cependant, les limitations du **protocole HTTP** font que ce n'est vraiment pas souple et agréable à utiliser. Le mieux reste d'utiliser IRC.

En revanche, voici une liste non-exhaustive des cas où PHP pourra résoudre vos problèmes :

- Automatiser la **gestion de news**, d'articles ou autres éléments de votre site qui ont un caractère répétitif.
- Réaliser des **webmails** afin de gérer ses mails partout.
- Mettre à disposition des autres sites automatiquement des informations de son site (via le **format RSS** par exemple)
- **Gérer des galeries photos, des annuaires de liens, des sondages, des forums, des moteurs de recherche internes à votre site**, etc. **PHP** est l'idéal dans ces domaines là.

Le carnet d'adresse du débutant

- Site officiel **d'Apache**
- Site officiel de **PHP**
- Site officiel de **MySQL**
- Site officiel d'**EasyPHP et Wamp**
- **WAMP**
- **MOV'AMP**
- Le repertoire francais de **dmoz.org sur PHP**
- **phpscripts-fr.net**, **php-help.org**, **hotscripts.com** pour trouver des applications

1)Afficher une phrase ou une image

Créer le fichier

Ouvrez votre éditeur préféré, et créez un nouveau **fichier PHP**. Le **code PHP** est toujours encadré par des balises le signalant. Les balises sont :

- **<?php ?>**

La première chose à savoir c'est qu'une syntaxe se termine **toujours** (sauf quelques exceptions) par un point-virgule, si vous l'oubliez vous verrez apparaître **une PARSE ERROR** lors de l'exécution de votre fichier.

Code PHP	**Donne comme résultat à l'écran**
<?php **echo 'Bonjour à tout le monde ' ;** **?>**	**Bonjour à tout le monde**

C'est la **construction echo** que nous utiliserons pour afficher du texte à l'écran. Ici on voit bien que la phrase n'est pas du tout formatée, voici donc comment l'on peut utiliser les balises **HTML dans PHP** (ci-dessous).

Code PHP	**Donne comme résultat à l'écran**
<?php **echo ' Bonjour à tout le monde ';** **?>**	**Bonjour à tout le monde**

Voilà nous avons ajouté la balise font en **html** pour formater le texte. En fait **PHP** ne fait pas le formatage, il faut utiliser **html** pour cela**. PHP** ne génére que du texte, et en général on veut que ce texte soit une source **html**, mais aussi quelque fois du **JavaScript**, de la feuille de style, voir du **xml**, etc. bref, tout ce qui est du texte.

Affichons maintenant **une image** en plus du texte.

Code PHP	Donne comme résultat à l'écran
<?php echo ' Bonjour à tout le monde '; echo ' '; ?>	Bonjour à tout le monde

On utilise simplement une **balise image (<img ...>)** du **html**. On voit donc clairement qu'il est indispensable de maîtriser le **html** avant de vouloir coder en **PHP**.

Différences entre les navigateurs

Il est important de ne pas oublier qu'**Internet Explorer** n'est pas le seul navigateur, il existe aussi notamment **Mozilla, Opera, Netscape, Galeon, Phoenix, etc.** Vous devrez donc faire particulièrement attention à bien **refermer les balises**, dans le bon ordre, ainsi que respecter la norme. Il est fortement conseillé que vous testiez vos scripts sous netscape plutôt que sous **IE**, même si l'idéal est de le faire en même temps sous les deux.

Différentes fonctions

Il existe deux fonctions pour l'affichage : **echo et print ().** La première est en fait une construction du **langage PHP**. Elle n'a pas besoin de parenthèses, contrairement à print, et peut prendre plusieurs paramètres, séparés par des ",". **echo** est donc légérement plus rapide que **print**, c'est d'ailleurs pour ça que nous la privilégions. Voici un exemple des différentes formulations possibles :

- **echo ' le texte ';**
- **echo ' le texte ',' le texte ';**
- **print (' le texte ');**

Chaîne de caractères

Une chaîne de caractères est un ensemble de caractères délimités par des signes. Les signes permettant de délimiter une chaîne de caractères en **PHP** sont ' ou ". La différence entre les deux réside dans le fait que **PHP** examinera ce que contient une chaîne entre ", mais pas une chaîne qui est entre ' qu'il affichera directement. Il est donc préférable d'utiliser les chaînes délimitées par ' qui sont plus rapides.

Caractères spéciaux

Il existe un problème avec les **chaînes de caractères**, quand on veut afficher une chaîne contenant un ' et que celle-ci est délimitée par des '. En effet, cela donne une ligne comme :

Code PHP	Donne comme résultat à l'écran
echo ' j'utilise PHP ';	Parse error: parse error, unexpected T_STRING, expecting ',' or ';' in votrefichier.php on line 2

Là forcément **PHP** croit qu'il faut s'arreter au deuxième ', et ne comprend donc pas la suite, ce qui se traduit par un parse error. La solution, **c'est l'antislash (\)** qui permet de faire comprendre à **PHP** qu'il ne faut pas s'arrêter sur ce caractère-là. Ce qui donne :

Code PHP	Donne comme résultat à l'écran
`echo ' j\'utilise PHP ';`	j'utilise PHP

Il faut utiliser la même chose avec les " dans des chaînes délimitées par des ". Mais du coup, pour mettre un \ dans une chaîne (ce qui est déjà plus rare), comment faire ? Eh bien c'est simple, il suffit de le faire précéder par un autre \, ce qui donne un \\.

Il existe aussi d'autres caractères spéciaux :

- **\t : tabulation**
- **\r : retour chariot**
- **\n : nouvelle ligne**

Attention, ces caractères spéciaux ne fonctionnent que dans une chaîne délimitée par des ", si vous les utilisez dans des chaînes délimitées par des ' vous verrez apparaître à l'écran \n par exemple ! Voici des exemples de codes utilisant ces caractères spéciaux :

- `echo ' un texte ',' \n ';`
- `echo " un texte\n ";`

Notez bien que ces caractères s'appliquent aux sources html qui sont générées. Et comme vous le savez, un retour à la ligne dans un fichier html ne fait pas d'effet, il faut mettre un **
**. Néanmoins ils peuvent servir à clarifier le code source, ou pour d'autres utilisations que nous verrons plus loin.

2)Afficher la date et l'heure

Avec **PHP** il est fort simple d'afficher la date du jour mais aussi de savoir quel jour nous serons dans 432 jours et réciproquement dans le passé. Voyons tout d'abord une date simple, nous allons en profiter pour utiliser notre première variable **(les variables commencent toujours par le signe dollar $).**

Code PHP	Ce qui donne à l'écran

`<?php`
`$date = date("d-m-Y");`
`$heure = date("H:i");`
`Print("Nous sommes le $date et il est $heure");`
`?>`
	Nous sommes le 14-09-2000 et il est 15:10

C'est donc la fonction **date()** qui permet d'obtenir l'heure locale du serveur, mais attention l'heure locale est fonction de la situation géographique du serveur en lui-même. En effet un serveur situé au canada vous donnera l'heure du Canada. Dans le code ci-dessus nous

générons la variable **$date** en lui donnant la valeur de ce que retourne la fonction **date("d-m-Y")** en l'occurrence : **14-09-2000**. Les paramètres contenus entre les parenthèses **d-m-Y** peuvent être placés dans l'ordre que vous désirez, ainsi la date au format US sera écrite ainsi : **date("Y-m-d"),** il existe beaucoup de paramètres (extrait de la doc. en français de **Nexen.net**), je vous conseille de les tester pour vous rendre compte de ce que chaque paramètre retourne comme résultat :

- a - "am" (matin) ou "pm" (après-midi)
- A - "AM" (matin) ou "PM" (après-midi)
- d - Jour du mois, sur deux chiffres (éventuellement avec un zéros) : "01" à "31"
- D - Jour de la semaine, en trois lettres (et en anglais) : par exemple "Fri" (pour Vendredi)
- F - Mois, textuel, version longue; en anglais, i.e. "January" (pour Janvier)
- h - Heure, au format 12h, "01" à "12"
- H - heure, au format 24h, "00" à "23"
- g - Heure, au format 12h sans les zéros initiaux, "1" à "12"
- G - Heure, au format 24h sans les zéros initiaux, "0" à "23"
- i - Minutes; "00" à "59"
- j - Jour du mois sans les zéros initiaux: "1" à "31"
- l - ('L' minuscule) - Jour de la semaine, textuel, version longue; en anglais, i.e. "Friday" (pour Vendredi)
- L - Booléen pour savoir si l'année est bissextile ("1") ou pas ("0")
- m - - Mois; i.e. "01" à "12"
- n - Mois sans les zéros initiaux; i.e. "1" à "12"
- M - Mois, en trois lettres (et en anglais) : par exemple "Jan" (pour Janvier)
- s - Secondes; i.e. "00" à "59"
- S - Suffixe ordinal d'un nom

3)PHP dans du code HTML

Attention : A partir du moment où vous placez du code PHP dans un fichier *.htm ou *.html, vous devrez renommer ce fichier en *.php ou encore *.phtml, bien que le plus utilisé soit *.php. Si vous ne faites pas cette manipulation, le code apparaîtra en toutes lettres dans le navigateur sans être exécuté par le serveur (n'ayant pas reconnu l'extension associée à PHP).

Comme je vous le disais en introduction, l'un des avantages du PHP c'est qu'il s'intègre facilement dans du code HTML classique. C'est d'ailleurs pour cela (en partie) qu'il connaît un fort succès sur les **homepages persos**. En effet chacun peut à sa guise inclure quelques parties en **PHP** sans avoir à casser le site entièrement.

Le code PHP/HTML	**Donne comme résultat à l'écran**
<html> <body> Le texte en HTML <?php // le code PHP --------- $heure = date("H\hi");	**Le texte en HTML et celui en PHP. Il est 18h16.**

print(" et celui en PHP."); ?> <!-- retour au code HTML --> Il est <?php echo $heure; ?>. </body> </html>	

L'exemple ci-dessus démontre bien cette facilité à mélanger les deux langages. Notez que la seconde fois j'ai utilisé la fonction echo et non pas print(), c'est le plus souvent dans ce cas que je l'utilise pour une lecture plus simple du code dans les autres cas ce sera print().

En **PHP** si vous souhaitez ajouter des commentaires il suffit de faire suivre deux slashes // puis le commentaire de votre choix. Je mets l'accent sur les commentaires surtout lorsque que vous aurez des dizaines de lignes de code, ils vous seront utiles pour vous y retrouver 6 mois plus tard, donc n'hésitez pas à en mettre, même sur des choses qui vous paraissent logiques sur l'instant. Si vous souhaitez mettre plusieurs lignes en commentaire, vous pouvez également utiliser le "slash étoile" puis "étoile slash" à la fin comme ceci : /* le commentaire */.

La fonction **include()** : si le code de votre page **HTML** est long, rajouter des lignes de codes **PHP** en plus n'est pas pertinent, la méthode utilisée pour bien séparer le **code HTML** du code **PHP** est la fonction **include().**

Le code HTML/PHP	Donne comme résultat à l'écran

<html>
<body>
Le texte en html
<?php
include("toto.inc.php"); // on appelle le fichier
?>
</body>
</html>
	Le texte en html et celui en PHP. Il est 18h16.

Le code PHP de toto.inc.php	
<?php $heure = date("H\hi"); print(" et celui en PHP. Il est $heure. "); ?>	

Et voilà le **code PHP** est maintenant dans un fichier bien séparé mais est exécuté à l'appel du fichier principal. Vous aurez noté que les fichiers qui sont inclus portent l'extension *.inc.php, ceci pour une meilleure lisibilité. Ainsi, en effet vous savez tout de suite si le fichier est exécuté directement ou bien s'il est uniquement appelé dans un ou plusieurs autres fichiers.

4) La concaténation

Le **point** est utilisé pour concaténer des chaînes, variables etc. Prenons l'exemple d'une phrase où un texte doit être collé au bout d'une variable, (voyez ci-dessous), pour que **PHP** sache que le nom de la variable s'arrête à un endroit précis, nous utiliserons le point.

Le code PHP	Donne comme résultat à l'écran
<?php $date = gmdate("H\hi"); print("Il est $date"."gmt."); ?>	Il est 19h05gmt.

Vous le voyez pour éviter que **PHP** pense que la variable porte le nom **$dategmt**, il faut refermer la double quote, mettre un point puis la rouvrir pour mettre le restant du texte (gmt). Notez également que le second point est lui placé entre les doubles quotes, donc, sera interprété comme du texte simple et non pas comme une demande de concaténation.

Nous allons maintenant voir la difference entre du texte entre ' ' dites **simples quotes** et du texte entre **double quotes** " "

Le code PHP	Donne comme résultat à l'écran
<?php $nom = "Martin"; echo "Mon nom est $nom"; ?>	Mon nom est Martin
<?php $nom = "Martin"; // affichage avec des simple quote echo 'Mon nom est $nom'; ?>	Mon nom est $nom
<?php $nom = "Martin"; // affichage avec des simple quote echo 'Mon nom est '.$nom; ?>	Mon nom est Martin

Vous l'aurez compris, **PHP** n'interprète pas ce qui se trouve entre simple quotes, ainsi, ce n'est pas la valeur de $nom qui est affiché, mais **$nom**. Il faut donc utiliser un opérateur de concaténation (le .) pour avoir l'affichage voulu. Pensez aussi que si vous voulez afficher un ' dans un texte entre deux ' ', alors il faudra faire:

```
echo 'Aujourd\'hui';
```

Ainsi le \ (**backslash ou antislash**) indique à **PHP** qu'il ne faut pas considérer le ' du mileu comme celui qui délimite la fin de la chaîne de caractères, mais juste un caractère comme un autre. Il en va de même pour afficher un " entre deux " ".

Ci-dessous vous allez voir qu'il est possible de concaténer directement une fonction et une chaîne de caractères.

Le code PHP	Donne comme résultat à l'écran

<?php print('Nous sommes le '.gmdate('d-m-Y').'...'); ?>	Nous sommes le 15-09-2000...

Nous avons réduit le code d'une ligne, ce qui n'est pas négligeable pour les développements importants, par contre j'admets que ceci est moins lisible pour quelqu'un qui débute totalement.

Dans certains exercices futurs, nous verrons comment appeler une page en passant quelques variables, dans ce cas la concaténation nous servira je propose donc de regarder le tableau ci-dessous :

Ce qu'il faut éviter de faire :	Ce qu'il est conseillé de faire :
<?php $fichier = "fichier.php3?var=$var&data=$data"; ?>	<?php $fichier = 'fichier.php3?var='.$var.'&data='.$data; ?>

En d'autres termes, chaque fois que vous collez du texte et une variable (ou fonction), n'oubliez pas de mettre le **point**. Je ne dis pas que la première méthode ne fonctionne pas, mais elle n'est pas orthodoxe, et autant prendre les bonnes habitudes tout de suite.

5)Récupérer les valeurs d'un formulaire

Quand l'un de vos visiteurs entre les informations dans un formulaire, celle-ci sont récupérées sous forme de variables. Le nom de ces variables dépend de la méthode d'envoi du formulaire. Comme dans notre exemple suivant la méthode d'envoi est POST, il faut mettre comme nom **$_POST['nom_du_champ'].**
Pour les anciens qui exploitaient les variables de facon **$nom_du_champ** au lieu de **$_POST['nom_du_champ'],** je conseille de lire de toute urgence le tutoriel de flyingcow sur les variables globales à **OFF** et surtout d'arrêter de coder ainsi. Cette variable contient ce qu'a entré le visiteur dans le champ. Allez, un exemple me paraît plus simple, ci-dessous le **name="nom"** devient **$_POST['nom']** et **name="prenom"** devient $_POST['prenom'], il ne reste plus qu'à faire un **print()** des variables. Pour simplifier le nom des variables, dans notre exemple, on fait **$nom = $_POST['nom']** et **$prenom = $_POST['prenom']** pour assigner la valeur de la variable **$_POST['prenom']** a **$prenom** et idem pour **$_POST['nom']** **(attention un nom de variable ne doit pas contenir d'espace et ne doit pas commencer par un chiffre alors n'en mettez pas dans vos nom de champ).**

Le code HTML du formulaire	Donne comme résultat à l'écran
<html><body> <form method="post" action="verif.php"> Nom : <input type="text" name="nom" size="12"> Prénom : <input type="text" name="prenom" size="12"> <input type="submit" value="OK">	Nom: Prénom: OK

</form></body></html>	
Le code PHP de verif.php	**Donne comme résultat à l'écran après envoi "OK"**
<?php $prenom = $_POST['prenom']; $nom = $_POST['nom']; print(" Bonjour $prenom $nom "); ?>	**Bonjour Thaal Rasha**

Il va bien sûr maintenant falloir contrôler les informations que rentre le visiteur pour éviter au maximum les erreurs. La première fonction que nous utiliserons est **empty(),** qui permet de contrôler si un champs est vide. Ensuite nous allons contrôler que **$_POST['url']** commence bien par **http://** à l'aide des deux fonctions **strtolower()** et **substr().**

Le code HTML du formulaire	**Donne comme résultat à l'écran**
<html><body> <form method="post" action="verif.php"> Titre : <input type="text" name="titre" size="12"> URL : <input type="text" name="url" size="12" value="http://"> <input type="submit" value="OK"> </form></body></html>	Titre: URL: OK
Le code PHP de verif.php	**Donne comme résultat à l'écran après envoi "OK"**

<?php
$titre = $_POST['titre'];
$url = $_POST['url'];
if(empty($titre))
{
print(" Le 'Titre' est vide ! ");
exit();
}
// vérification du début de l'url
$verif_url = strtolower($url);
$verif_url = substr("$verif_url", 0, 7);
// on vérifie les 7 premiers caractères
if ($verif_url!="http://")
{
print("L'URL doit commencer par http://");
exit();
}
else
{
print("$titre :
$url");
	Erreur n°1 :
	Le 'Titre' est vide !
	
	Erreur n°2 : L'URL doit commencer par http://

```
}
?>
```

Avec cet exemple nous commençons à attaquer les conditions, c'est un aspect primordial dans tous les langages. La première vérification porte sur le champ titre, la fonction empty() permet de contrôler si celui-ci est vide ou non. Ce qui nous donne :

- **if(empty($titre)){ print(" Le 'Titre' est vide ! "); exit(); } :** Si la variable **$titre** est vide alors j'affiche le message : 'Le titre est vide' (placé entre accolades) et j'arrête l'exécution du reste du code avec la commande exit().
- Par contre si la variable n'est pas vide, l'exécution ne prend pas en compte ce qui se trouve entre accolades et continue.

La seconde vérification est plus fine puisqu'il s'agit de vérifier que les sept premiers caractères qui ont été entrés par le visiteur sont bien **http://**. Pour commencer nous utilisons la fonction **strtolower()** qui permet de transformer tous les caractères en minuscules (ex. **HTTP://www.MONsite.CoM** devient **http://www.monsite.com**). Puis à l'aide de la fonction substr(), nous sélectionnons les 7 premiers caractères **(0 est toujours le premier caractère d'une chaine - le second chiffre ' 7 ' étant le nombre de caractères à sélectionner),** puis nous les comparons à ce que nous avons dans notre condition if :

- **if ($verif_url!="http://"){** print("L'url doit commencer par http://"); exit(); **} :** Si les 7 premiers caractères sont différents (signe: !=) de http://, alors on exécute ce qui ce trouve entre accolades (en l'occurrence on affiche un message d'erreur), puis nous arrêtons le reste du code avec la commande exit().
- Par contre si le résultat est correct, PHP ignore ce qui ce trouve entre accolades et exécute le reste du code.

Vous pourrez faire autant de tests que vous voudrez sur les champs, mais ne soyez pas trop draconien car les visiteurs n'aiment pas trop que l'on empiète sur leur liberté. Les contrôles les plus fréquents s'effectuent sur les **url** et **email** pour savoir si l'email comporte bien un **@** et un **point**.

Le code HTML du formulaire	**Donne comme résultat à l'écran**
<html><body> <form method="post" action="verif.php"> Votre email : <input type="text" name="email" size="20"> <input type="submit" value="OK"> </form></body></html>	Votre email: [ ] OK
Le code PHP de verif.php	**Donne comme résultat à l'écran après envoi "OK"**
<?php **$email = $_POST['email'];** **$point = strpos($email,".");** **$aroba = strpos($email,"@");** **if($point=='')** **{**	**Erreur n°1 :** Votre email doit comporter un **point** ! **Erreur n°2 :** Votre email doit comporter un '**@**' !

echo "Votre email doit comporter un point";
}
elseif($aroba=='')
{
echo "Votre email doit comporter un '@'";
}
else
{
echo "Votre email est: '$email'";
}
?>
	Si pas d'erreur : Votre email est : email@email.com

Comme son nom l'indique, la fonction **strpos()** retourne la position d'un caractère dans une chaîne si celui-ci existe, autrement **strpos()** retourne **rien**. C'est ce que nous utilisons pour savoir si les **points** et **@** sont bien présents dans l'email.

Exemple : Si **strpos()** retourne **10** cela veut dire que le premier caractère recherché est placé juste après les 10 premiers caractères donc en **11ème position** dans la chaîne, puisque vous devez toujours vous rappeler que **PHP** commence à compter à **0** et non pas **1**.

6) Les structures de contrôle

Pour commencer voici un petit tableau bien utile sur les instructions les plus utilisées

if	Si
else	Autrement
elseif	Autrement Si
switch	selon
while	Chaque fois que (boucle)
for	Tant que (boucle)
==	Strictement égal à
!=	Différent de
<	Plus petit que
>	Plus grand que
<=	Plus petit ou égal
>=	Plus grand ou égal
and ou &&	Et
or ou ||	Ou

Pour illustrer les **if**, **else** et **elseif**, voici un exemple très simple à lire, nous définissons une variable à la valeur **512** puis nous allons tester si celle-ci est comprise entre 0 et 499 puis entre 500 et 999 et enfin supérieure à 999, ce qui nous donne :

Le code PHP	Ce qui donne à l'écran

<?php
$toto = 512;
// on enchaîne les contrôles ci-dessous ----
if($toto>=0 && $toto<500) //1er
{
echo $toto.' est compris entre 0 et 499';
}
elseif($toto>=500 && $toto<1000) //2eme
{
echo $toto.' est compris entre 500 et 999';
}
else //3eme
{
echo $toto.' est plus grand que 999';
}
?>
	512 est compris entre 500 est 999

1er contrôle : Si (512 est plus grand ou égal à 0 et que 512 est plus petit que 500**) {** on affiche le 1er message **; }**
2e contrôle : Si (512 est plus grand ou égal à 500 et que 512 est plus petit que 1000**) {** on affiche le 2e message **; }**
3e contrôle : Dans tous les autres cas { on affiche le 3e message **; }**

Je vous invite à faire plusieurs tests en changeant à chaque fois la valeur de **$toto** pour vérifier que les messages respectifs s'affichent bien quand les conditions sont remplies.

Voici un autre exemple pour mieux comprendre, et qui nous permettra de voir la structure **switch** par la suite.

Le code PHP	Ce qui donne à l'écran

<?php
$medor = 'chien';
// on enchaîne les contrôles ci-dessous ----
if($medor == 'girafe')
{
echo 'Medor est une girafe ';
}
elseif($medor == 'elephant')
{
echo 'Medor est un éléphant';
}
elseif($medor == 'souris')
{
echo 'Medor est une souris';
}
elseif($medor == 'chien')
{
echo 'Medor est un chien';
}
	Medor est un chien

```
elseif($medor == 'chat')
   {
   echo 'Medor est un chat';
   }
else
   {
   echo 'Peut être un hippopotame ? Qui sait ...';
   }
?>
```

Cependant vous vous apercevez que cette structure est peu souple, et pas forcement facile à lire. Utiliser un **switch** permet de résoudre ce problème. Le code suivant est exactement le même que le précédent, mais ecrit avec un **switch**.

Le code PHP	Ce qui donne à l'écran

`<?php`
`$medor = 'chien';`
`switch($medor)`
`{`
`case 'girafe':`
`echo 'Medor est une girafe ';`
`break;`
`case 'elephant':`
`echo 'Medor est un éléphant';`
`break;`
`case 'souris':`
`echo 'Medor est une souris';`
`break;`
`case 'chien':`
`echo 'Medor est un chien';`
`break;`
`case 'chat':`
`echo 'Medor est un chat';`
`break;`
`default:`
`echo 'Peut être un hippopotame ? Qui sait ...';`
`}`
`?>`
	Medor est un chien

Notez bien l'utilisation de break;. Cela permet de sortir de la boucle et donc de gagner en efficacité.

Passons maintenant à la **fameuse boucle while**. Nous allons reprendre notre variable **$toto** à laquelle nous allons donner la valeur **6**, puis à l'aide de la boucle nous allons nous mettre dans le cas où nous ne connaissons pas la valeur de $toto et allons la rechercher. Ce qui donne :

Le code PHP	Ce qui donne à l'écran

`<?php`
`$toto = 6;`
`$i = 0;`
	Toto est different de 0
	Toto est different de 1
	Toto est different de 2

//----------[DEBUT BOUCLE]----------
while($i != $toto)
{
echo 'Toto est different de '.$i.'
';
$i++; // $i++ est équivalent à ($i+1)
}
//----------[FIN BOUCLE]----------
echo 'Toto est égal à '.$i;
?>
	Toto est different de 3
	Toto est different de 4
	Toto est different de 5
	Toto est égal à 6

Par convention, la variable $i fait toujours office de compteur dans une boucle elle a toujours la valeur 0 au début, vous notez que cette valeur prend **+1** à la fin de la boucle **($i++)** et chaque fois que la condition a été respectée **(en l'occurrence que $i est différent de $toto)** on retourne à while et l'on fait un nouveau test, etc., ce qui donne en français :

1. **$i = 0**, on teste si **0 est différent de toto = Oui**, on affiche le message (echo) puis on ajoute 1 à $i ($i++) et on retourne à **while**.
2. **$i = 1**, on teste si **1 est différent de toto = Oui**, on affiche le message (echo) puis on ajoute 1 à $i ($i++) et on retourne à while.
3. **$i = 2**, on teste si **2 est différent de toto = Oui**, on affiche le message (echo) puis on ajoute 1 à $i ($i++) et on retourne à **while**.
4. etc...
5. **$i = 6**, on teste si **6 est différent de toto = Non**, on sort de la boucle (accolades), et on poursuit le code. En l'occurrence on affiche le message (Toto est égal à 6).

Nous allons maintenant voir les **boucles for**. Elles permettent de realiser la même chose que les boucles while, encore une fois c'est la syntaxe qui change. Au lieu de déclarer le compteur avant le debut de la boucle ($i=0;) et a chaque fin de tour d'incrementer le **compteur ($i++),** on le fait directement dans la déclaration de la boucle. Voici le meme code que précédemment avec une **boucle for** :

Le code PHP	Ce qui donne à l'écran

<?php
$toto = 6;
//----------[DEBUT BOUCLE]----------
for($i=0; $i != $toto ; $i++)
{
echo 'Toto est different de '.$i.'
';
}
//----------[FIN BOUCLE]----------
echo 'Toto est égal à '.$i;
?>
	Toto est different de 0
	Toto est different de 1
	Toto est different de 2
	Toto est different de 3
	Toto est different de 4
	Toto est different de 5
	Toto est égal à 6

Il est vraiment très important de maîtriser les boucles car c'est là un élément systématiquement utilisé pour afficher des résultats venant d'une base de données et nous l'utiliserons beaucoup.

7)Ecrire et lire dans un fichier texte

Nous allons voir ici comment l'on peut ouvrir et lire depuis un fichier se trouvant sur un serveur **FTP**, le vôtre en l'occurrence. Pour commencer vous créez un fichier de type *.txt

(vous pouvez mettre l'extension que vous voulez, voire pas du tout) et nous placerons le fichier dans le même répertoire que le script **PHP**.

Contenu du fichier data.txt	**Donne comme résultat l'écran**
1523	**Le fichier contient : 1523**
Le code PHP data.php	

```
<?php
$fp = fopen("data.txt","r");   //(1)
$donnees = fgets($fp,255);     //(2)
fclose($fp);                   //(3)
//Affichage du résultat ----------
echo'Le fichier contient : '.$donnees;
?>
```

Vous le voyez il est relativement simple de lire ce qui se trouve dans un fichier :

1. On ouvre le fichier data.txt en lecture seule r avec la fonction **fopen().**
2. La lecture s'effectue avec la fonction **fgets()** et on spécifie le nombre de caractères (ici 255 soit la première ligne).
3. Ensuite il ne reste plus qu'à refermer le fichier texte c'est la fonction **fclose**().
4. Enfin on affiche le résultat, c'est la variable **$donnees** qui contient **1523**.

Revenons à la première ligne. La commande **r** indique que l'on ouvre le fichier uniquement en lecture seule. Nous allons voir ci-dessous que pour l'ouvrir en **lecture/écriture**, il suffit de mettre **r+.** Concernant la seconde fonction **fgets (),** on spécifie le nombre de caractères que l'on veut lire dans le fichier (255). Dans notre cas nous aurions très bien pu mettre **($fp,4)**; puisque **1523** ne comporte que 4 caractères = logique. Ceci dit, le fait de mettre systématiquement 255 n'engendre pas de problème dans notre cas, sachez tout de même que **255** est le nombre maximum de caractères par ligne, le **256e** passera à la ligne automatiquement à l'affichage.

Voici le **code PHP** qui va vous permettre de réaliser un compteur de page. Notez qu'ici le fichier texte s'appelle **compteur.txt**.

Le code PHP de compteur.php

```
<?php
$fp = fopen("compteur.txt","r+"); // 1.On ouvre le fichier en lecture/écriture
$nbvisites = fgets($fp,11);       // 2.On récupère le nombre dans le fichier
$nbvisites++;                     // 3.On incrémente le nombre de visites(+1)
fseek($fp,0);                     // 4.On se place en début de fichier
fputs($fp,$nbvisites);            // 5.On écrit dans le fichier le nouveau nb
fclose($fp);                      // 6.On ferme le fichier
print("$nbvisites visiteurs");    // 7.On affiche le compteur à l'écran
?>
```

Il vous suffit de placer ce code dans la page un **index.php** de votre site. Ici la fonction **fseek()**

permet de se replacer où l'on veut en l'occurrence 0, donc au début, ensuite avec **fputs()** on écrit dans le fichier à l'endroit spécifié.

Pour finir avec les fichiers, n'oubliez pas que votre fichier texte qui se trouve sur votre **FTP** doit avoir avoir tous les droits (**chmod 777**) pour que le script puisse y écrire.

PHP permet également de créer et effacer des fichiers sur un serveur distant (FTP).

8)Les fonctions utilisateurs

PHP propose de nombreuses fonctions, mais un autre avantage est de pouvoir créer les siennes à l'aide de **function().** Ceci est vraiment très utile pour ne pas avoir à retaper des parties de code en entier.

Le code PHP de fonction.php

```
<?php
function Arial($size,$color,$texte)
{
  print("<font face=Arial size=".$size." color=".$color.">".$texte."</font>");
}
?>
```

Le code PHP de index.php

```
<?php
Require("fonction.php");    // on appelle la page contenant la fonction
// affichage ---------------------------------------
Arial("2","red","Ici le texte ...");
Arial("3","#0F74A3","Le second texte ...");
?>
```

On appelle index.php dans le navigateur, ce qui donne à l'écran

Ici le texte ...Le second texte ...

Reprenons depuis le début. On distingue donc deux étapes lors de l'utilisation de fonctions. La première consiste à créer votre fonction en lui donnant un nom, en choisissant les paramètres dont elle aura besoin et déterminer ce qu'elle doit faire. Une fois cela fait, vous pouvez appeler votre fonction. Plus précisément : entre parenthèses ce sont les arguments que vous entrerez lors de l'utilisation de cette fonction. Ces arguments correspondent bien sûr à ceux qui se trouvent entre parenthèses à la création de la fonction. Ici, il s'agit juste de donner la taille **$size**, la couleur **$color** et le contenu du texte **$texte**. L'utilisation de vos fonctions se fait de la même manière que les fonctions intégrées de PHP **(print et echo en l'occurrence).** Dans **index.php**, vous voyez que l'on appelle la fonction **Arial()** et qu'on lui donne les valeurs que l'on veut.

Pratique : Je vous conseille de créer un fichier **fonction.php** qui va contenir toutes vos fonctions de texte, tableau, etc. Il suffit juste ensuite de mettre un **require()** en entête de chacun de vos autres **fichiers PHP** pour pouvoir utiliser toutes les fonctions. Je pense que vous voyez de suite l'intérêt de créer ses propres fonctions. Cela permet une bien meilleure lecture du code et un gain de temps à la programmation.

9)Les variables d'environnement

PHP propose toute une **série de variables** qui sont déjà implantées dans le langage sans que vous ayez à les créer, on les appelle les **variables d'environnement**. Ces variables appartiennent à la famille des **variables globales $_SERVER.**

Code PHP	Ce qui donne à l'écran
<?php print("Votre adresse IP est : $_SERVER['REMOTE_ADDR']"); ?>	Votre adresse IP est : 201.65.8.56

Voilà, rien de plus simple pour connaître **l'adresse IP** d'un visiteur, il s'agit de la variable d'environnement **$_SERVER['REMOTE_ADDR'].**

Voici ci-dessous la liste exhaustive des variables d'environnement existantes :

Variables	Description	Résultat à l'écran (Free.fr)
$_SERVER['DOCUMENT_ROOT']	Racine du serveur	/var/www/php.proxad.net
$_SERVER['HTTP_ACCEPT_LANGUAGE']	Langage accepté par le navigateur	fr
$_SERVER['HTTP_HOST']	Nom de domaine du serveur	proxyphp3.free.fr
$_SERVER['HTTP_USER_AGENT']	Type de navigateur	Mozilla/4.0 (compatible; MSIE 5.0; W DigExt)
$_SERVER['PATH_INFO']	Chemin web du script	/d2expert.free.fr/debut/fichier.php
$_SERVER['PATH_TRANSLATED']	Chemin complet du script	/var/www/free.fr/3/d/2/e/x/d2expert/de
$_SERVER['REQUEST_URI']	Chemin du script	/d2expert.free.fr/debutant/fichier.php
$_SERVER['REMOTE_ADDR']	Adresse IP du client	195.132.7.201
$_SERVER['REMOTE_PORT']	Port de la requête HTTP	45039
$_SERVER['QUERY_STRING']	Liste des paramètres passés au script	var=23&data=ok
$_SERVER['SERVER_ADDR']	Adresse IP du serveur	212.27.32.44
$_SERVER['SERVER_ADMIN']	Adresse de l'administrateur du serveur	email@email.com

$_SERVER['SERVER_NAME']	Nom local du ser`_veur	php.proxad.net
$_SERVER['SERVER_SIGNATURE']	Type de serveur	?
$_SERVER['REQUEST_METHOD']	Méthode d'appel du script	GET

Je voulais m'attarder quelques instants sur ce qui est une fonction et non pas une variable d'environnement. Je veux parler de **phpinfo().** A l'aide de celle-ci et en 10 secondes vous aller pouvoir connaître la configuration et la version exacte de **PHP** qu'utilise le serveur où vous êtes hébergé.

Code PHP	Ce qui donne à l'écran
<?php phpinfo(); ?>	Cela serait trop long

10)Quelques fonctions utiles

Fonction	Description	Code PHP	Rés.
addslashes()	Ajoute des anti-slashes devant les caractères spéciaux	$res = addslashes("L'a");	L\'a
stripslashes()	Retire les anti-slashes devant les caractères spéciaux.	$res = stripslashes("L\'a");	L'a
dechex()	Retourne la valeur hexadécimale d'un nombre (ici 2548).	$res = dechex("2548");	9f4
ceil()	Retourne le nombre entier supérieur (utiliser floor() pour le nombre entier inférieur et round() pour le nombre entier le plus proche).	$res = ceil("12.1"); *	13
chunk_split()	Permet de scinder une chaîne en plusieurs morceaux.	$res = chunk_split("DGDFEF","2","-");	DG-DF-EF-
htmlentities()	Remplace les caractères par leur équivalent HTML (si ils existent).	$res = htmlentities("&");	&
strstr()	Recherche le premier caractère 'p' dans la chaîne et affiche le reste de la chaîne v	$res = strstr ("webmaster@debut.org", "p");	debut.org

	compris le 'p'.		
strlen()	Retourne la longueur de la chaîne	$res = strlen("lachainedecaracteres");	20
strtolower()	Passe tous les caractères en minuscules.	$res = strtolower("LA CHAINE dE caRActERes");	la chaine de caracteres
strtoupper()	Passe tous les caractères en MAJUSCULES.	$res = strtoupper("LA CHAINE dE caRActERes");	LA CHAINE DE CARACTERES
str_replace()	Remplace un caractère par un autre dans une chaîne. Tiens compte de la casse.	$res = str_replace("a","o","Lalala");	Lololo
trim()	Efface les espaces blancs (\n, \r, etc) au début et à la fin d'une chaîne (pas au milieu).	$res = trim(" Salut le monde ");	Salut le monde
ucfirst()	Met la première lettre de chaque chaîne en Majuscule.	$res = ucfirst("salut le monde. ca va ?");	Salut le monde. ca va ?
ucwords()	Met la première lettre de chaque mot d'une chaîne en Majuscule.	$res = ucwords("salut le monde");	Salut Le Monde
strpos()	Recherche la position du premier caractères trouvé. Retourne le nombre de caractères placés avant lui (ici 4).	$res = strpos("abcdef","e");	4
ereg()	Recherche si une chaîne de caractère est contenue dans une autre (ex. recherche si "ABCDE" contient "BCD").	if(ereg("BCD","ABCDEF")) {echo "oui";} else {echo "non";}	oui

*** La virgule sous PHP est représentée par un point ".", ainsi 12,1 s'écrit : 12.1**

Dans la colonne Code PHP du tableau ci-dessus, j'ai volontairement omis de mettre un : echo $res; pour afficher le résultat, mais bien sûr vous devrez le rajouter à chaque fois pour que le résultat apparaisse à l'écran.

11) SQL/MySQL (Create, Alter & Drop)

1) CREATE TABLE : Le **langage SQL (Structured Query Langage)** permet d'interroger une base de données qui supporte ce langage, les plus connues sont **MS Access, mSQL, MySQL et PostgreSQL**. Comme vous le savez nous utiliserons **MySQL.**

La première chose que nous allons devoir faire c'est de créer une table, c'est la commande **CREATE TABLE**, voyez la syntaxe ci-dessous qui permet de créer **clients_tbl**. Il est important de savoir comment l'on crée une table en **SQL** avant de passer par l'interface **phpMyAdmin** pour le faire :

Syntaxe SQL pour créer la table : clients_tbl

```
CREATE TABLE clients_tbl (id INT not null AUTO_INCREMENT, prenom VARCHAR (50) not nu
, nom VARCHAR (50) not null , ne_le DATE not null , ville VARCHAR (90) not null , enfants INT no
null , PRIMARY KEY (id))
```

Commentons la table en la visualisant sous phpMyAdmin pour MySQL

Champ	Type	Null	Defaut	Extra	Commentaire
id	int	Non		auto_increment	int pour Integer (nombre entier) : C'est l'id qui va nous permettre de classer nos enregistrements, l'auto-increment se charge d'affecter un nouveau numéro aux nouveaux enregistrements qui s'ajoutent dans la table.
prenom	varchar(50)	Non			varchar pour Chaîne de caractères : Nous l'utilisons quand nous souhaitons que le champ puisse recevoir des chaînes mélangeant texte, nombre, etc.
nom	varchar(50)	Non			Idem que prenom, la valeur placée entre parenthèses définit le nombre maximum de caractères que le champ accepte, ici 50. Si vous essayez d'insérer une chaîne de 55 caractères, les 5 derniers seront coupés.
ne_le	date	Non	0000-00-00		date : Permet de stocker des dates, mais attention toujours au format US soit : Année/mois/jour, si vous envoyez un format français J/M/A vous obtiendrez une date fausse dans la base.
ville	varchar(90)	Non			Idem que PRENOM, ici nous avons spécifié 90 caractères car certaines villes comportent beaucoup de caractères (avec 90 nous sommes sûr)
enfants	int	Non	0		int (nombre entier) : Dans ce champ n'arrivera que des nombres entiers

donc le INT voir SMALL INT et de rigeur.

Ci-dessous la table : clients_tbl une fois créée

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ id + prenom + nom + ne_le + ville + enfants +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Bien sûr pour le moment cette table ne comporte aucun enregistrement ...

Dès cet instant vous devez consulter la documentation officielle de MySQL pour connaître tous les types de champs que vous pouvez utiliser dans une table, voici quelques exemples :

- **TINYINT : Entier de 0 à 255 (unsigned)**
- **SMALLINT : Entier de 0 à 65535 (unsigned)**
- **MEDIUMINT : Entier de 0 à 16777215 (unsigned)**
- **INT : Entier de 0 à 4294967295 (unsigned)**
- **BIGINT : Entier de 0 à 18446744073709551615 (unsigned)**
- **DECIMAL : Un nombre à virgule flottante**
- **DATE : Une date, va de '1000-01-01' à '9999-12-31'**
- **DATETIME : Date et Heure, va de '1000-01-01 00:00:00' à '9999-12-31 23:59:59'**
- **TIMESTAMP : Date et Heure exprimée en secondes depuis le 1er janviers 1970. Va de '1970-01 00:00:00' à quelque part, durant l'année 2037**
- **TIME : Une mesure de l'heure, va de '-838:59:59' à '838:59:59'**
- **YEAR : Une annnée, va de 1901 à 2155**
- **CHAR : Chaîne de caractère de taille fixe, va de 1 à 255 caractères**
- **VARCHAR : Chaîne de caractère de taille variable, va de 1 à 255 caractères**
- **TINYTEXT ou TINYBLOB : Un objet BLOB ou TEXT, longueur maximale de 255**
- **TEXT ou BLOB : Un objet BLOB ou TEXT, longueur maximale de 65535**
- **MEDIUMTEXT ou MEDIUMBLOB : Un objet BLOB ou TEXT, longueur maximale de 167772**
- **LONGTEXT ou LONGBLOB : Un objet BLOB ou TEXT, longueur maximale de 4294967295**

Le TINYINT couvre de 0 à 255 quand il est UNSIGNED, mais il couvrira de - 128 à 127 (soit toujours 255) si il ne l'est pas. Ceci est valable pour tous les Integers (nombres entiers).

2. ALTER TABLE : Une fois que votre table est créée vous pourrez bien sûr la modifier en utilisant **ALTER TABLE,** voyez l'exemple ci-dessous pour ajouter un champ :

Syntaxe SQL pour ajouter le champ 'tel' à la table : clients_tbl

```
ALTER TABLE clients_tbl ADD tel INT not null
```

Ci-dessous la table clients_tbl une fois modifiée

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ id + prenom + nom + ne_le + ville + enfants + tel +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Voilà ci-dessus notre table a été modifiée grâce à **ALTER TABLE** puis **ADD** (comme ajouter).

Maintenant supprimons le champ tel de la table :

Syntaxe SQL pour supprimer le champ 'tel' de la table : clients_tbl

```
ALTER TABLE clients_tbl DROP tel
```

Ci-dessous la table clients_tbl une fois modifiée

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
+ id + prenom + nom + ne_le + ville + enfants +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Voilà ci-dessus notre table a été modifiée grâce à **ALTER TABLE** puis **DROP**. Cette **commande DROP** va également nous être utile pour supprimer une table complète.

3. DROP TABLE : Il s'agit de la commande qui permet de supprimer une table complète, attention en supprimant une table vous perdez tout ce qu'elle contenait, donc à utiliser avec prudence.

Syntaxe SQL pour supprimer la table : clients_tbl

```
DROP TABLE clients_tbl
```

Notre table **clients_tbl** a été complètement effacée avec la commande **DROP TABLE**, dans notre cas elle ne contenait aucune information mais dans le cas contraire, cela signifie la perte de toutes les données de la table.

Nous avons vu ici la majeure partie des choses importantes à connaître en ce qui concerne la création et la modification des tables. Dans les prochains exercices nous verrons comment traiter les données d'une table avec les commandes **SELECT, INSERT, UPDATE et DELETE.**

12)SQL/MySQL (Insert et Select)

Pour commencer nous allons re-créer la **table client_tbl** de l'exercice précédent.

Syntaxe SQL pour créer la table : clients_tbl

```
CREATE TABLE clients_tbl (id INT not null AUTO_INCREMENT, prenom
VARCHAR (50) not null , nom VARCHAR (50) not null , ne_le DATE not null , ville
VARCHAR (90) not null , enfants INT not null , PRIMARY KEY (id))
```

Ci-dessous la table : clients_tbl une fois créée

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ id + prenom + nom + ne_le + ville + enfants +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
```

Bien sûr pour le moment cette table ne comporte aucun enregistrement ...

La commande INSERT INTO

Cette commande permet d'insérer des enregistrements dans une table en l'occurrence **clients_tbl**.

- **INSERT INTO clients_tbl(id,prenom,nom,ne_le,ville,enfants) VALUES('','Patrick','Martin','1965-10-08','Bordeaux','2')**

Ou bien cette autre requête qui aura le même résultat, mais que nous vous déconseillons fortement, car elle ne marchera plus si vous modifiez votre table, en particulier si vous ajoutez un champ :

- **INSERT INTO clients_tbl VALUES('','Patrick','Martin','1965-10-08','Bordeaux','2')**

Ci-dessous la table : clients_tbl avec le nouvel enregistrement

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ id + prenom + nom + ne_le + ville + enfants +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 1 + Patrick + Martin + 1965-10-08 + Bordeaux + 2 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
```

Dans le premier exemple nous avons spécifié les noms des champs (entre parenthèses) juste après le nom de la table. Dans ce cas il n'est pas obligatoire de le faire si dans les values vous spécifiez une valeur par champs, **mysql** affectera les valeurs dans l'ordre donné.

Par contre si vous ne donnez que deux valeurs, il sera important de toujours spécifier les champs dans lesquels elles doivent être insérées, exemple :

- **INSERT INTO** clients_tbl(id,nom) Values('','Dupond')

  Ou encore :

- **INSERT INTO** clients_tbl(nom,enfants) Values('Dupond','2')

La valeur du champ **id** est vide, je vous rappelle que ce champ est en **auto-increment** dans notre base, un nombre automatique sera donc attribué à chaque nouvel enregistrement, dans ce cas la valeur dans la requête peut rester vide.

Important : Si vous voulez n'insérer que quelques-unes des valeurs d'un enregistrement, vous devrez avoir spécifié lors de la création de la table que les autres champs peuvent rester vides **(NULL).**

La commande SELECT

Nous allons travailler à partir de la table ci-dessous qui comporte cinq enregistrements :

Ci-dessous le contenu final de la table : clients_tbl

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ id + prenom + nom + ne_le + ville + enfants +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 1 + Patrick + Martin + 1965/10/08 + Bordeaux + 2 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 2 + Julien + Lebreton + 1964/02/21 + Paris + 2 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 3 + Marc + Richard + 1958/04/15 + Lille + 4 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 4 + Francis + Perrin + 1982/12/05 + Paris + 0 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 5 + Daniel + Bacon + 1974/07/13 + Reims + 1 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
```

Ceci est une représentation en mode console et non pas sous phpMyAdmin

Admettons que nous voulions afficher uniquement les personnes qui n'ont que deux enfants, la **requête SQL** sera :

```
SELECT * FROM clients_tbl WHERE enfants='2'
```

Soit en français :

SELECT	**Je sélectionne**
*****	**Tous les champs**
FROM clients_tbl	**Depuis la table client_tbl**
WHERE enfants='2'	**Quand le champ enfants est égal à 2**

Ce qui donne comme résultat :

```
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 1 + Patrick + Martin + 1965/10/08 + Bordeaux + 2 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 2 + Julien + Lebreton + 1964/02/21 + Paris + 2 +
+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
```

Voici la même requête, mais cette fois nous n'allons demander l'affichage que des noms et prénoms :

SELECT nom, prenom FROM clients_tbl WHERE enfants='2'

Soit en français :	
SELECT	**Je sélectionne**
nom, prenom	**Les champs nom et prenom**
FROM clients_tbl	**Depuis la table client_tbl**
WHERE enfants='2'	**Quand le champ enfants est égal à 2**

Ce qui donne comme résultat :

```
++++++++++++++++++++++++++
+  Patrick  +  Martin   +
++++++++++++++++++++++++++
+  Julien   +  Lebreton +
++++++++++++++++++++++++++
```

Reprenons la première requête et ajoutons d'autres conditions (where), à savoir que nous allons demander l'affichage des personnes qui ont 1 ou 2 enfants et qui habitent la ville de Paris :

SELECT * FROM clients_tbl WHERE enfants='1' OR enfants='2' AND ville='Paris' ;

- Vous devez faire la différence entre OR et AND, nous pouvons l'analyser ainsi :

Si le nombre d'enfant est 1 et que la ville est Paris, c'est OK.
Et si le nombre d'enfants est 2 et que la ville est Paris, c'est OK.

Il faut que les deux conditions soient respectées pour que l'enregistrement soit affiché.

Voici les opérateurs possibles :	
+	**Addition**
-	**Soustraction**
*****	**Multiplication**
/	**Division**
<	**Plus petit que**
<=	**Plus petit ou égal à**
=	**Égal à**
!= ou <>	**N'est pas égal à**
>=	**Plus grand ou égal à**
and	**ET**
or	**OU**
not	**Négation**

Pour finir voici quelques autres exemples de SELECT :

SELECT * FROM clients_tbl WHERE ne_le < "1978-01-01"

Sélection des personnes ayant une date de naissance plus petite que 1978/01/01.

SELECT * FROM clients_tbl WHERE enfants != '0'

Sélection des personnes dont le nombre des enfants est différent de " 0 ".

SELECT * FROM clients_tbl WHERE nom LIKE 'le%'

Sélection des personnes dont le nom commence par " le ".

SELECT * FROM clients_tbl WHERE nom LIKE '%ri%'

Sélection des personnes qui ont dans leur nom la syllabe " ri " .

Il serait bien trop long d'énumérer tous les exemples possibles, mais n'hésitez pas à consulter **la documentation en Français de Nexen.net sur MySQL**.

13)SQL/MySQL (Delete et Update)

Si nous continuons de travailler la table client_tbl de l'exercice n°11.

La commande UPDATE

Cette commande permet de modifier les valeurs d'un enregistrement déjà présent dans la table :

- **UPDATE clients_tbl SET prenom='Jacques' WHERE id=1**

Cette commande ne pose vraiment pas de problème particulier, décortiquons la syntaxe :

UPDATE clients_tbl	**Mise à jour de la table Clients_tbl**
SET prenom='Jacques'	**Modifier le champ prenom pour la valeur Jacques**
WHERE id=1	**Quand le champ id est égal à 1**

Ci-dessous l'enregistrement de la table : clients_tbl une fois modifié.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ id + prenom + nom + ne_le + ville + enfants +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 1 + Jacques + Martin + 1965/10/08 + Bordeaux + 2 +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
```

Bien sûr nous pouvons changer plusieurs valeurs d'un même enregistrement dans la même requête :

- **UPDATE clients_tbl SET prenom='Jean-Pierre', nom='Papin', ville='Marseille', enfants=3 WHERE id=1**

Vous le voyez, il suffit de séparer les **champs/valeurs** par une virgule, ce qui donne comme résultat dans la table :

Ci-dessous l'enregistrement de la table : clients_tbl une fois modifié.

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ id + prenom + nom + ne_le + ville + enfants +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
+ 1 + Jean-Pierre+ Papin + 1965/10/08 + Marseille + 3 +
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
```

La commande DELETE

Bon vous l'aurez sans doute compris cette commande sert à supprimer un ou plusieurs enregistrements d'une table ainsi :

- **DELETE FROM clients_tbl WHERE id=1**

Là non plus la commande ne pose vraiment pas de problème particulier, décortiquons la syntaxe :

DELETE FROM clients_tbl	**Effacer de la table Clients_tbl**
WHERE id=1	**Quand l'id de l'enregistrement est égal à 1**

14)SQL/MySQL (Where)

Nous allons ici comprendre les mécanismes du **where** et ses possibilités grâce à un exemple comprenant deux tables, une table **Livres** et une table **Genres**.Tout d'abord il faut savoir que le mot clé **where** peut être utilisé dans les requêtes **SELECT**, **DELETE** et **UPDATE** (son utilisation est cumulative grâce au **AND** ou **OR**) mais pas dans le **INSERT** car on ne peut pas insérer à un endroit précis de la table, l'insertion se fait toujours à la fin.

Livres

IDLivre	Prix	Titre	CodeGenre
1	40	Le glaive magique	BD
3	40	Gaffes en gros	BD
4	40	Lagaffe nous gâte	BD
5	45	QRN sur Bretzelburg	BD

Genres

CodeGenre	LibelleGenre
BD	Bande Dessinée
Po	Poésie
RG	Roman de gare
Ro	Roman
SF	Science Fiction

6	80	Tour de manège	RG
7	45	Le spectre aux balles d'or	BD
8	30	La bonne chanson	Po
9	50	La jeune Parque	Po
10	50	Michel Strogoff	Ro
11	50	La Serpe d'or	BD
12	70	Toujours aimer	RG
13	70	Toujours aimante	RG
14	72	Toujours aimé	RG

1 - Sélection directe par l'ID ou le code

Ce cas vous permet de faire une sélection par exemple sur un livre dont vous connaissez le numéro, imaginons si vous faites un répertoire de tous les livres donc avec une instruction de sélection, lorsque le visiteur clique sur le titre de l'un des livres par exemple **Le spectre aux balles d'or**, le détail de celui-ci apparaît.

SELECT * FROM Livres WHERE **IDLivre = 7;**

Cette sélection va vous permettre d'avoir les informations concernant le livre :

Livres

IDLivre	Prix	Titre	CodeGenre
7	45	Le spectre aux balles d'or	BD

SELECT * FROM Livres WHERE **IDLivre = 7** OR **IDLivre = 8;**

Bien sûr ce genre de sélection ne s'applique pas seulement sur les identifiants des tables.

Attention : il faut bien choisir entre le **AND** et le **OR** lorsque vous voulez faire des sélections multiples, par exemple ici le **OR** permet de sélectionner deux fois sur le même champ, le **AND** est utilisé lorsque vous faites une sélection sur différent champ de la table.

SELECT * FROM Livres WHERE IDLivre = 7 AND Prix > 50;

La sélection dans cette requête prend le livre avec l'ID égal à 7 et d'un prix supérieur à 50. Donc aucun enregistrement ne sera retourné par la requête puisque le livre 7 coûte 45F.

SELECT * FROM Livres WHERE IDLivre = 7 OR Prix > 50;

Ici le livre 7 et tous les autres livres d'un prix supérieur à 50F sont sélectionnés.

2 - Le mot clé LIKE

Le mot clé **LIKE** avec le **WHERE** va vous permettre de faire une sélection "approximative" dans vos tables, par exemple ici vous voulez tous les livres dont le titre commence par 'tou'.

SELECT * FROM Livres WHERE **Titre** LIKE **'tou%';**

Livres

IDLivre	Prix	Titre	CodeGenre
6	80	Tour de manège	RG
12	70	Toujours aimer	RG

13	70	Toujours aimante	RG
14	72	Toujours aimé	RG

SELECT * FROM Livres **WHERE** CodeGenre **LIKE** 'r_';

Livres

IDLivre	Prix	Titre	CodeGenre
6	80	Tour de manège	RG
10	50	Michel Strogoff	Ro
12	70	Toujours aimer	RG
13	70	Toujours aimante	RG
14	72	Toujours aimé	RG

Pour remplacer un seul caractère dans un **LIKE** il faut utiliser **_**, pour remplacer une chaîne de caractère il faut utiliser **%**. Le **LIKE** ne respecte pas la casse (minuscule - majuscule) contrairement au égal.

3 - Sélection selon une liste de valeur

Ici nous allons voir un mot clé permettant de faire la sélection à partir d'une liste de valeur. Le **WHERE** est ici utilisé avec un **IN**.

select * from Livres where **Prix** in**(40, 50, 72);**

Le mot clé **IN** permet de sélectionner les enregistrements dans la table **Livres** où le prix est de 40, 50 ou 72.

Livres

IDLivre	Prix	Titre	CodeGenre
1	40	Le glaive magique	BD
3	40	Gaffes en gros	BD
4	40	Lagaffe nous gâte	BD
9	50	La jeune Parque	Po
10	50	Michel Strogoff	Ro
11	50	La Serpe d'or	BD
14	72	Toujours aimé	RG

De la même façon il y a le **mot clé contraire** au **in** qui est le **not in** qui va en fait sélectionner les enregistrements qui n'ont pas l'une des valeurs indiquées sur le champ demandé. **SELECT * FROM Livres** WHERE **Prix** NOT IN **(40, 50, 72);**

Cette requête sélectionne donc les livres qui ne valent pas 40, 50 ou 72F.

4 - Sélection selon une plage de valeurs

Nous allons lier dans cette partie le **WHERE** au **BETWEEN...AND** qui permet de faire une sélection en précisant une plage de données, ce mot clé n'est utilisé qu'avec des valeurs numériques (integer, date,etc.).

SELECT * FROM Livres WHERE **Prix** BETWEEN **40** AND **50;**

Livres

IDLivre	Prix	Titre	CodeGenre
1	40	Le glaive magique	BD
3	40	Gaffe en gros	BD
4	40	Lagaffe nous gâte	BD
5	45	QRN sur Bretzelburg	BD
7	45	Le spectre aux balles d'or	BD
9	50	La jeune Parque	Po

10	50	Michel Strogoff	Ro
11	50	La Serpe d'or	BD

S'il on veut par exemple avoir toutes les BD qui ont un prix entre 40 et 50F la requête est la suivante : **SELECT * FROM Livres** WHERE **(Prix** BETWEEN **40** AND **50)** AND **(CodeGenre = 'BD');**
Les parenthèses ne sont bien sûr pas obligatoires mais c'est pour une question de clarté.

5 - Jointures

Ce système permet de joindre deux tables différentes qui ont un champ en relation, ici la table **Livres** et la table **Genres** ont le champ CodeGenre en commun, ces deux tables peuvent donc être jointes grâce au **where**.
select * from Livres, Genres where **Livres.CodeGenre = Genres.CodeGenre;**

IDLivre	Prix	Titre	Livres.CodeGenre	Genres.CodeGenre	LibelleGenre
1	40	Le glaive magique	BD	BD	Bande Dessinée
3	40	Gaffes en gros	BD	BD	Bande Dessinée
4	40	Lagaffe nous gâte	BD	BD	Bande Dessinée
5	45	QRN sur Bretzelburg	BD	BD	Bande Dessinée
6	80	Tour de manège	RG	RG	Roman de gare
7	45	Le spectre aux balles d'or	BD	BD	Bande Dessinée
8	30	La bonne chanson	Po	Po	Poésie
9	50	La jeune Parque	Po	Po	Poésie
10	50	Michel Strogoff	Ro	Ro	Roman
11	50	La Serpe d'or	BD	BD	Bande Dessinée
12	70	Toujours aimer	RG	RG	Roman de gare
13	70	Toujours aimante	RG	RG	Roman de gare
14	72	Toujours aimé	RG	RG	Roman de gare

Pour joindre deux tables il faut indiquer leur nom dans le **FROM** et utiliser la syntaxe avec le nom, le point et le champ de jointure. Le problème avec **cette jointure** c'est que l'on se retrouve avec deux fois la même colonne, car elle vient une première fois de la table **Livres** et l'autre de la table **Genres**. Pour éviter cela et avoir quelque chose de plus sympa, la requête peut être modifiée comme ceci :
SELECT Titre, Livres.CodeGenre, LibelleGenre, Prix FROM **Livres, Genres** WHERE **Livres.CodeGenre = Genres.CodeGenre;**

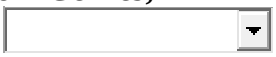
Titre	Livres.CodeGenre	LibelleGenre	Prix
Le glaive magique	BD	Bande Dessinée	40
Gaffes en gros	BD	Bande Dessinée	40
Lagaffe nous gâte	BD	Bande Dessinée	40
QRN sur Bretzelburg	BD	Bande Dessinée	45
Tour de manège	RG	Roman de gare	80
Le spectre aux balles d'or	BD	Bande Dessinée	45
La bonne chanson	Po	Poésie	30
La jeune Parque	Po	Poésie	50
Michel Strogoff	Ro	Roman	50
La Serpe d'or	BD	Bande Dessinée	50
Toujours aimer	RG	Roman de gare	70
Toujours aimante	RG	Roman de gare	70
Toujours aimé	RG	Roman de gare	72

Pour différencier les deux tables dans le **SELECT** il faut indiquer le nom de la table sur les champs à afficher qui sont éventuellement en double à cause de la jointure.

6 - Application

Une fois n'est pas coutume, une petite application sur le même exemple ne fait pas de mal pour comprendre encore mieux les tutoriaux.
Nous allons faire une sélection de tous les LibelleGenre et avec cette sélection, en utilisant le **where** on affichera les livres correspondant.

La requête de sélection des libellés est la suivante : **select * from Genres;**

Le **script** permettant de créer la liste dans une **boîte <select>** : 

```
<select name="genre">
<?php
mysql_connect($server, $user, $pass) or die('Erreur de connexion');
mysql_select_db($db) or die('Base inexistante');
$sql = 'SELECT * FROM Genres;';
$query = mysql_query($sql) or die( 'Erreur' );
$nb = mysql_num_rows($query);
if ( !$nb[0] ) {
echo '<option>Aucun genre</option>';
```

```
} else {
while ( $list = mysql_fetch_array( $query ) ) {
echo '<option value="'.$list['CodeGenre'].'">'.$list['LibelleGenre'].'</option>';
}
}
mysql_close();
?>
</select>
```

Lorsque le visiteur valide le formulaire on reçoit donc une variable passé en **POST** sur la page de résultat où l'on va pouvoir faire notre requête de sélection avec le **where**. Pour l'exemple on va choisir les BD.
select * from Livres **where** CodeGenre = 'BD';
Dans le script voici ce qui peut se passer :

```
<?php
mysql_connect($server, $user, $pass) or die('Erreur de connexion');
mysql_select_db($db) or die('Base inexistante');
$sql = "select * from Livres where CodeGenre = '".$_POST['genre']."';";
$query = mysql_query($sql) or die( 'Erreur' );
$nb = mysql_num_rows($query);
if ( $nb>0 ) {
echo 'Aucun livre dans ce genre';
} else {
while ( $list = mysql_fetch_array( $query ) ) {
//traitement du résultat
}
}
 mysql_close();
?>
```

Et voilà le résultat :

Livres

IDLivre	Prix	Titre	CodeGenre
1	40	Le glaive magique	BD
3	40	Gaffes en gros	BD
4	40	Lagaffe nous gâte	BD
5	45	QRN sur Bretzelburg	BD
7	45	Le spectre aux balles d'or	BD
11	50	La Serpe d'or	BD

15) Fonctions PHP pour mySQL

Les fonctions PHP pour MySQL commencent toujours par **mysql_,** en voici ci-dessous la liste exhaustive. En règle générale je n'utilise pas plus de 6 ou 7 fonctions, c'est d'ailleurs sur celles-ci que je mettrai l'accent plus bas dans cet exercice, ceci dit, je vous conseille tout de même les tester toutes.

Fonctions	Descriptions
mysql_affected_rows()	Retourne le nombre de rangées affectées par la dernière requête faite sur la base de données.
mysql_close()	Ferme la connexion à une base de données.
mysql_connect()	Établit une connexion vers la base de données spécifiée dans les arguments. Si cette base se trouve sur un port différent, faites suivre le nom de la machine par (**:**) puis le numéro du port (ex. **:8080**). Cette connexion est automatiquement fermée à la fin de l'exécution du script sauf si une fonction **mysql_close()** intervient auparavant.
mysql_create_db()	Permet de créer une nouvelle base de données.
mysql_data_seek()	Déplace le pointeur interne de la rangée du résultat vers la rangée spécifiée. Utilisez cette fonction avec **mysql_fetch_row()** pour passer à la rangée spécifiée et récupérer les données.
mysql_db_query()	Permet d'exécuter une requête **SQL** sur une ou plusieurs tables d'une base de données. **Attention cette fonction est obsolète, ne l'utilisez plus, utilisez mysql_query !**
mysql_drop_db()	Permet de supprimer une base de données. Dans ce cas toutes les données sont perdues.
mysql_errno()	Retourne le numéro de l'erreur générée lors de la dernière action sur la base de données.
mysql_error()	Retourne la description textuelle d'une erreur générée par la dernière action sur une base de données.
mysql_fetch_array()	Retourne un tableau qui représente tous les champs d'une rangée dans le résultat. Chaque appel récupère la prochaine rangée et ce jusqu'à ce qu'il n'y en ait plus. Chaque valeur de champ est stockée de deux façons: elle est indexée par un offset qui commence par '**0**', et indexée par le nom du champ.
mysql_fetch_assoc()	Comme **mysql_fetch_array()** mais indexée uniquement par le nom du champ. Dans la plupart des cas, utiliser cette fonction ou mysql_fetch_row() de préférence à mysql_fetch_array().
mysql_fetch_field()	Récupère l'information attachée à un champs du résultat. Ces champs sont numérotés à partir de zéro.
mysql_fetch_lengths()	Retourne un tableau d'une longueur spécifiée pour chaque champ de résultat.
mysql_fetch_object()	Cette fonction est semblable à **mysql_fetch_array()** et à **mysql_fetch_row()**. Mais à la place, elle retourne un objet. Chaque champ du résultat correspond à une propriété dans l'objet renvoyé. Chaque appel à **mysql_fetch_object()** retourne la rangée suivante, ou **False** s'il ne reste plus de rangée.
mysql_fetch_row()	Comme **mysql_fetch_array()** mais indexée uniquement par le numéro d'ordre du champ. C'est la méthode la plus rapide pour obtenir des résultats à partir d'une requête. Dans la plupart des cas, utiliser cette fonction ou **mysql_fetch_assoc()** de préférence à mysql_fetch_array().

mysql_field_flags()	Permet d'obtenir une description des options rattachées aux champs spécifiés.
mysql_field_len()	Retourne la longueur maximale du champ spécifié.
mysql_field_name()	Retourne le nom d'une colonne. L'argument champ correspond à un offset numéroté à partir de zéro.
mysql_field_seek()	Déplace le pointeur interne du champ vers le champ spécifié. Le prochain appel vers **mysql_field_seel()** retournera l'information de ce champs.
mysql_field_table()	Retourne le nom de la table pour le champ spécifié.
mysql_field_type()	Retourne le type d'un champ particulier dans le résultat obtenu.
mysql_free_result()	Libère la mémoire associée au résultat spécifié. Elle n'est toutefois pas strictement nécessaire, car cette mémoire est automatiquement vidée lorsqu'un script termine son exécution.
mysql_insert_id()	Après l'insertion d'un nouvel enregistrement avec un champ **auto_increment**, la fonction **mysql_insert_id()** retourne l'**ID** qui vient d'être affecté au nouvel enregistrement.
mysql_list_dbs()	Interroge le serveur pour obtenir une liste de bases de données. Elle retourne un pointeur de résultat qui pourra être exploité avec **mysql_fetch_row()** et d'autres fonctions similaires.
mysql_list_fields()	Retourne un pointeur de résultat correspondant à une requête sur une liste de champs pour la table spécifiée. Ce pointeur pourra être exploité par toutes les fonctions qui recherchent des rangées à partir d'un résultat. Notez que l'argument lien reste optionnel.
mysql_list_tables()	Retourne le pointeur de résultat d'une liste de tables pour la base de données spécifiée. Ce pointeur pourra être exploité par toutes les fonctions qui recherchent des rangées à partir d'un résultat. Notez que l'argument lien reste optionnel.
mysql_num_fields()	Retourne le nombre de champs dans un résultat.
mysql_num_rows()	Retourne le nombre de rangées dans un résultat. Anciennement **mysql_numrows()**
mysql_pconnect()	Cette fonction opère de la même manière que **mysql_connect()**, sauf que la connexion ne se referme pas à la fin de l'exécution du script sauf si un **mysql_close()** se trouve en fin de script.
mysql_query()	Permet d'exécuter une requête **SQL** sur une ou plusieurs tables d'une base de données. Si la requête exécute une instruction: **INSERT**, **DELETE** ou **UPDATE**, une valeur booléenne sera retournée (**0** ou **1**). Dans le cas d'une requête de type **SELECT**, vous obtiendrez un identifiant de résultat.
mysql_result()	Retourne la valeur du champ spécifié dans la rangée concernée. L'argument champ peut être un numéro et, dans ce cas, il sera considéré comme un champ offset. Il peut également désigner le nom de la colonne, avec éventuellement celui de la table. Enfin, il peut également renvoyer à un alias.
mysql_select_db()	Sélectionne la base de données par défaut.

Fonctions: **mysql_connect(), _select_db() , _query(), _num_rows(), _close()**

Ces fonctions sont le minimum que vous utiliserez à chaque fois que vous interrogerez une base de données **MySQL**, voyez le code ci-dessous (nous avons repris notre table **clients_tbl**).

Code PHP

```
<?php
$db = mysql_connect('sql.free.fr', 'login', 'password');  // 1
mysql_select_db('nom_de_la_base',$db);                    // 2
$req = mysql_query('SELECT * FROM clients_tbl');          // 3
$res = mysql_num_rows($req);                              // 4
echo 'Il y a '.$res.' enregistrement(s) dans la table Clients.';  // 5
mysql_close($db);  // 6
?>
```

Donne à l'écran

Il y a 5 enregistrement(s) dans la table Clients.

Explication du code ci-dessus :

1. On se connecte à la base de données :
   Host : Par exemple **sql.free.fr**, vous pouvez également utiliser **localhost** par défaut.
   Login : Ensuite vous devez mettre le **login** pour accéder à la base (chez les hébergeurs gratuits c'est souvent le même login que l'accès **FTP**).
   Password : Et pour finir le **mot de passe,** là aussi il s'agit très souvent du même password que l'accès **FTP.**
2. On sélectionne la base de données, en effet je vous rappelle que **MySQL** est un **serveur de bases de données**, donc il peut contenir plusieurs bases. Bien sûr dans votre cas si vous êtes chez un hébergeur gratuit, vous n'avez en général droit qu'à une seule base, mais **MySQL** ne le sait pas, il faut donc lui spécifier sur quelle base vous souhaitez vous connecter.
   Chez Free.fr et Nexen le nom de la base est souvent le même que le login de connection.
3. La fonction **mysql_query()** permet de passer une requête SQL vers la base de données, c'est évidement l'un des attraits intéressants de php (nous initialisons au passage la variable **$req** qui contient la requête).
4. La fonction **mysql_num_rows()** permet de compter le nombre d'enregistrements que retourne la requête **$req** dans notre cas : 5 puisqu'il s'agit d'un simple **select** sur la table sans aucune condition, nous initialisons donc une variable **$res** qui contient : 5.
5. Il ne reste plus qu'à afficher le nombre de résultats avec un **echo()** de **$res**.
6. Et pour finir on referme la connexion ouverte avec **mysql_connect** - avec la fonction **mysql_close().** Cette fonction n'est pas vraiment obligatoire avec un **mysql_connect(),** car par défaut la connexion sera coupée automatiquement à la fin de l'exécution du script.

16)Interroger une table MySQL

Maintenant que nous nous sommes connectés à la base de données, nous allons interroger une table pour en extraire les résultats puis les ranger dans un ordre précis. Créons d'abord une

table comme ci-dessous.

```
Requête SQL : CREATE TABLE famille_tbl ( id int(11) NOT NULL auto_increment, nom varchar(255) NOT NULL, prenom varchar(255) NOT NULL, statut varchar(255) NOT NULL, date date DEFAULT '0000-00-00' NOT NULL, PRIMARY KEY (id), KEY id (id), UNIQUE id_2 (id) );

INSERT INTO famille_tbl VALUES( '', 'Dupond', 'Grégoire', 'Grand-père', '1932-05-17');
INSERT INTO famille_tbl VALUES( '', 'Dupond', 'Germaine', 'Grand-mère', '1939-02-15');
INSERT INTO famille_tbl VALUES( '', 'Dupond', 'Gérard', 'Père', '1959-12-22');
INSERT INTO famille_tbl VALUES( '', 'Dupond', 'Marie', 'Mère', '1961-03-02');
INSERT INTO famille_tbl VALUES( '', 'Dupond', 'Julien', 'Fils', '1985-05-17');
INSERT INTO famille_tbl VALUES( '', 'Dupond', 'Manon', 'Fille', '1990-11-29');
```

Structure de la table (PhpMyAdmin) :

Champ	Type	Null	Defaut	Extra
id	int(11)	Non	0	auto_increment
nom	varchar(255)	Non		
prenom	varchar(255)	Non		
statut	tinyint(255)	Non	0	
date	date	Non	0000-00-00	

Contenu de la table "famille_tbl"

id	nom	prenom	statut	date
1	Dupond	Grégoire	Grand-père	1932-05-17
2	Dupond	Germaine	Grand-mère	1939-02-15
3	Dupond	Gérard	Père	1959-12-22
4	Dupond	Marie	Mère	1961-03-02
5	Dupond	Julien	Fils	1985-05-17
6	Dupond	Manon	Fille	1990-11-29

Affichage des résultats tels qu'ils sont dans la table sans condition.

Code PHP

```php
<?php
// on se connecte à MySQL
$db = mysql_connect('localhost', 'login', 'password');
// on sélectionne la base
mysql_select_db('nom_de_la_base',$db);
// on crée la requête SQL
$sql = 'SELECT nom,prenom,statut,date FROM famille_tbl';
```

```php
// on envoie la requête
$req = mysql_query($sql) or die('Erreur SQL !<br>'.$sql.'<br>'.mysql_error());
// on fait une boucle qui va faire un tour pour chaque enregistrement
while($data = mysql_fetch_assoc($req))
    {
    // on affiche les informations de l'enregistrement en cours
    echo '<b>'.$data['nom'].' '.$data['prenom'].'</b> ('.$data['statut'].')';
    echo ' <i>date de naissance : '.$data['date'].'</i><br>';
    }
// on ferme la connexion à mysql
mysql_close();
?>
```

Donne à l'écran

Dupond Grégoire (Grand-père), date de naissance : 1932-05-17
Dupond Germaine (Grand-mère), date de naissance : 1939-02-15
Dupond Gérard (Père), date de naissance : 1959-12-22
Dupond Marie (Mère), date de naissance : 1961-03-02
Dupond Julien (Fils), date de naissance : 1985-05-17
Dupond Manon (Fille), date de naissance : 1990-11-29

Explication :

Voici donc notre première boucle, vous vous apercevez que les résultats qui s'affichent sont exactement dans le même ordre que la table et pour cause, nous n'avons pas spécifié de condition dans notre requête (2), donc dans ce cas, la requête scanne la table de haut en bas.

Remarquez que la fonction **mysql_fetch_assoc()** renvoie un tableau dont les clés sont les noms des champs sélectionnés. On a aussi rajouté après **mysql_query()** ceci: or **die('Erreur SQL !
' .$sql. '<br>'. mysql_error());.** Cela veut dire qu'en cas d'erreur dans la requete vers mysql, ce qui arrive fréquemment, **PHP** va afficher un message indiquant l'erreur renvoyée par mysql (grâce à **mysql_error())** ce qui fournit une aide précieuse pour comprendre le problème.

Vous notez également que les dates de naissance sont au **format US**, ceci est normal puisque nous avons défini un type **DATE** dans notre table, nous verrons plus bas comment convertir les **dates US au format FR**.

Nous allons maintenant faire plusieurs tests en ne changeant uniquement que la requête SQL (2). **Le reste du code ne change pas.**

Affichage des résultats par ordre alphabétique de prénom.

Le code PHP de la requête

```php
<?php
// Gardez le code ci-dessus, changez juste la requête SQL
$sql = 'SELECT nom,prenom,statut,date FROM famille_tbl ORDER BY prenom';
// L'opérateur ORDER BY permet de classer soit alphabétiquement
```

```
// soit numériquement suivant le type du champ.
// Si l'on souhaite classer en décroissant (ex. de Z à A), nous
// y ajouterons DESC soit : ORDER BY prenom DESC
?>
```

Donne à l'écran

Dupond Gérard (Père), date de naissance : 1959-12-22
Dupond Germaine (Grand-mère), date de naissance : 1939-02-15
Dupond Grégoire (Grand-père), date de naissance : 1932-05-17
Dupond Julien (Fils), date de naissance : 1985-05-17
Dupond Manon (Fille), date de naissance : 1990-11-29
Dupond Marie (Mère), date de naissance : 1961-03-02

Affichage des résultats par comparaison de date.

Le code PHP de la requête

```
<?php
// Gardez le code ci-dessus, changez juste la requête
$sql = "SELECT nom,prenom, statut FROM famille_tbl WHERE date<'1960-01-01'";
// L'avantage d'avoir un type DATE dans notre base de données, c'est que
// nous pouvons comparer des dates dans la requête SQL.
// Ici nous ne souhaitons afficher que les membres de la famille qui sont
// nés avant le 1er janvier 1960, soit : WHERE date<'1960-01-01'
?>
```

Donne à l'écran

Dupond Grégoire (Grand-père), date de naissance : 1932-05-17
Dupond Germaine (Grand-mère), date de naissance : 1939-02-15
Dupond Gérard (Père), date de naissance : 1959-12-22

Affichage des résultats avec le commande LIKE.

La commande **LIKE en SQL** permet de fouiller le contenu de chaque champ. Recherchons tous les enregistrements dont le champ **prenom** commence par la lettre **G**.

Le code PHP de la requête

```
<?php
// Gardez le code ci-dessus, changez juste la requête
$sql = "select  nom,prenom,statut,date from famille_tbl where prenom like 'G%'";
// Le signe pourcentage "%" placé après le G, indique que la lettre G peut
// être suivie, mais pas précédée, d'autres caractères
// Notez aussi que LIKE n'est pas sensible à la casse, cela veut dire que
// la requête cherchera aussi bien des G majuscules que minuscules.
?>
```

Donne à l'écran

Dupond Grégoire (Grand-père), date de naissance : 1932-05-17
Dupond Germaine (Grand-mère), date de naissance : 1939-02-15
Dupond Gérard (Père), date de naissance : 1959-12-22

Maintenant voyons la même chose mais cette fois nous allons chercher la syllabe **ma** dans le champ **prenom**, qu'elle soit placée au début ou au milieu d'autres caractères.

Le code PHP de la requête

```
<?php
// Gardez le code ci-dessus, changez juste la requête
$sql = "select * from famille_tbl where prenom like '%MA%'";
// Le signe pourcentage "%" placé avant et après MA indique que la syllabe
// peut-être précédée ou suivie de caractères.
// Une fois de plus notez que LIKE n'est pas sensible à la casse, la requête
// cherchera aussi bien des MA majuscules que des ma en minuscules.
?>
```

Donne à l'écran

Dupond Marie (Mère), date de naissance : 1961-03-02
Dupond Manon (Fille), date de naissance : 1990-11-29

Bien sûr, il est possible de mettre plusieurs conditions dans la même requête, exemple :

- **SELECT * FROM famille_tbl WHERE prenom LIKE '%MA%' AND date<'1960-01-01' ORDER BY prenom**

Pour terminer voici comment vous allez pouvoir convertir la date du format US au format FR, une fois que vous avez récupéré l'information depuis la table. Nous vous donnons deux façons de procéder, la deuxième est bien meilleure.

Le code PHP (déconseillé pour ce cas précis)

```
<?php
// on se connecte à MySQL
$db = mysql_connect('localhost', 'login', 'password');
// on sélectionne la base
mysql_select_db('nom_de_la_base',$db);
// on crée la requete SQL
$sql = 'SELECT nom,prenom,statut,date FROM famille_tbl';
// on envoie la requête
$req = mysql_query($sql) or die('Erreur SQL !<br>'.$sql.'<br>'.mysql_error());

// on fait une boucle qui va faire un tour pour chaque enregistrement
while($data = mysql_fetch_array($req))
    {
    $a = substr($data['date'], 0, 4);     // conversion
    $m = substr($data['date'], 5, 2);     // de la date
    $j = substr($data['date'], 8, 2);     // au format
    $date = $j.'-'.$m.'-'.$a;             // Français
 // on affiche les informations de l'enregistrement en cours
    echo '<b>'.$data['nom'].' '.$data['prenom'].'</b> ('.$data['statut'].')';
    echo ' <i>date de naissance : '.$date.'</i><br>';
    }
// on ferme la connexion à mysql
mysql_close();
?>
```

Le code PHP (bien meilleur, utilisant les ressources de MySQL)

```
<?php
// on se connecte à MySQL
$db = mysql_connect('localhost', 'login', 'password');
// on seléctionne la base
mysql_select_db('nom_de_la_base',$db);
// on crée la requete SQL
$sql = "select nom,prenom,statut,DATE_FORMAT(date, '%d-%m-%Y') as datefr
from famille_tbl";
// on envoie la requête
$req = mysql_query($sql) or die('Erreur SQL !<br>'.$sql.'<br>'.mysql_error());
// on fait une boucle qui va faire un tour pour chaque enregistrement
while($data = mysql_fetch_array($req))
   {
   // on affiche les informations de l'enregistrement en cours
   echo '<b>'.$data['nom'].' '.$data['prenom'].'</b> ('.$data['statut'].')';
   echo ' <i>date de naissance : '.$data['datefr'].'</i><br>';
   }
// on ferme la connexion à mysql
mysql_close();
?>
```

Donne à l'écran, dans les deux cas

Dupond Grégoire (Grand-père), date de naissance : 17-05-1932
Dupond Germaine (Grand-mère), date de naissance : 15-02-1939
Dupond Gérard (Père), date de naissance : 22-12-1959
Dupond Marie (Mère), date de naissance : 02-03-1961
Dupond Julien (Fils), date de naissance : 17-05-1985
Dupond Manon (Fille), date de naissance : 29-11-1990

17)Alimenter une ou plusieurs tables mySQL

Dans cet exercice nous allons voir comment alimenter une ou plusieurs tables avec les données qui proviennent d'un même formulaire.

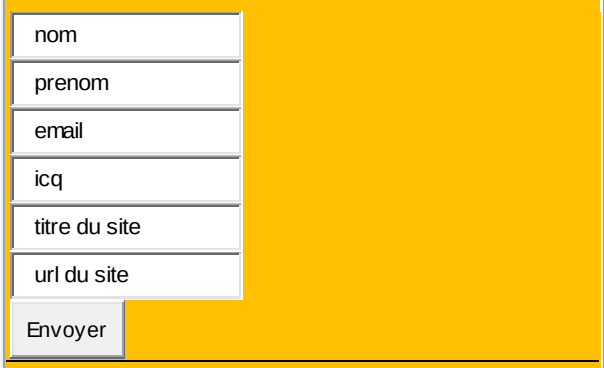
Alimenter une table

Pour commencer vous allez créer la table **infos_tbl dans phpMyAdmin** comme suit :

Requête SQL:

```
CREATE TABLE infos_tbl (id INT (11) not null AUTO_INCREMENT, nom
VARCHAR (35) not null, prenom VARCHAR (35) not null , email VARCHAR
(70) not null , icq INT (11) null , titre VARCHAR (70) not null , url VARCHAR
(255) not null , PRIMARY KEY (id), INDEX (id), UNIQUE (id));
```

Ensuite nous allons utiliser le formulaire ci dessous qui va alimenter la table :

Code HTML	Donne à l'écran
<html> <form method="POST" action="add.php"> <input type="text" name="nom" size="20" value="nom" maxlength="35"> <input type="text" name="prenom" size="20" value="prenom" maxlength="35"> <input type="text" name="email" size="20" value="email" maxlength="70"> <input type="text" name="icq" size="20" value="icq" maxlength="11"> <input type="text" name="titre" size="20" value="titre du site" maxlength="70"> <input type="text" name="url" size="20" value="url du site" maxlength="255"> <input type="submit" value="Envoyer" name="envoyer"> </form> </html>	

Vous noterez les **maxlength** dans chacun des champs, ceci permet de brider le nombre de caractères maximum que le visiteur peut entrer dans le champ, bien sûr ici le **maxlength** correspond au nombre de caractères spécifié dans la création de la table **infos_tbl**. Cela a un intérêt, celui d'être sûr que le visiteur ne tapera pas plus de caractères que prévu.

Voyons maintenant le **script php** en lui-même, celui-ci sera contenu dans le fichier **add.php** auquel fait référence le **POST** du formulaire :

Code PHP de "add.php"

```
<?php
// On commence par récupérer les champs
if(isset($_POST['nom']))      $nom=$_POST['nom'];
else      $nom="";
if(isset($_POST['prenom']))      $prenom=$_POST['prenom'];
else      $prenom="";
if(isset($_POST['email']))      $email=$_POST['email'];
else      $email="";
if(isset($_POST['icq']))      $icq=$_POST['icq'];
else      $icq="";
if(isset($_POST['titre']))      $titre=$_POST['titre'];
else      $titre="";
if(isset($_POST['url']))      $url=$_POST['url'];
```

```
else     $url="";
// On vérifie si les champs sont vides
if(empty($nom) OR empty($prenom) OR empty($email) OR empty($titre) OR empty($u
rl))
   {
   echo '<font color="red">Attention, seul le champs <b>ICQ</b> peut rester vide
!</font>';
   }
// Aucun champ n'est vide, on peut enregistrer dans la table
else
   {
 // connexion à la base
$db = mysql_connect('localhost', 'login', 'password')  or die('Erreur de connexion
'.mysql_error());
// sélection de la base
mysql_select_db('nom_de_la_base',$db)  or die('Erreur de selection '.mysql_error());
 // on écrit la requête sql
   $sql = "INSERT INTO infos_tbl(id, nom, prenom,email, icq, titre,
url) VALUES('','$nom','$prenom','$email','$icq','$titre','$url')";
   // on insère les informations du formulaire dans la table
   mysql_query($sql) or die('Erreur SQL !'.$sql.'<br>'.mysql_error());
   // on affiche le résultat pour le visiteur
   echo 'Vos infos on été ajoutées.';
   mysql_close();  // on ferme la connexion
   }
?>
```

Explication :

La **requête INSERT INTO** permet donc l'insertion des champs du formulaire dans la table. Dans notre cas le premier champ reste vide car il s'agit de l'id (identifiant) qui s'incrémente automatiquement à chaque nouvelle requête **INSERT**.Notez que dans le cas ou vous ne voudriez insérer que les champs Nom et Prénom dans la table vous pouvez très bien spécifier dans la requête le nom de chaque champ, comme suit :

```
$sql = "INSERT INTO infos_tbl(nom,prenom) VALUES('$nom','$prenom')";
```

Mais attention dans ce cas les autres champs de la table devront avoir l'attribut **NULL** et non pas **NOT NULL**, Null indique au champ qu'il pourra rester vide. Et si vous essayez d'insérer un nombre de valeurs différent du nombre des champs, mysql vous fera un joli message d'errreur. Bien sûr, l'idéal et de tester si l'url existe dans la table pour éviter les doublons, si celle-ci existe déjà on indique au visiteur qu'il ne peut pas valider son formulaire. J'ai pris l'exemple ici de l'url mais il va sans dire que vous pouvez vérifier les champs de votre choix, cela dépend des doublons que vous ne souhaitez pas avoir dans votre table. Voici le code :

Code PHP de "add.php" avec la vérification de doublon sur l'URL

```
<?php
// On récupère les champs (voir plus haut)
// On vérifie si les champs sont vides
```

```
if(empty($nom) OR empty($prenom) OR empty($email) OR empty($titre) OR empty($u
rl))
   {
   echo '<font color="red">Attention, seul le champs <b>ICQ</b> peut rester vide
!</font>';
   }
// Aucun champ n'est vide, on peut enregistrer dans la table
else
   {
   $db = mysql_connect('localhost', 'login', 'password');  // connexion à la base
   mysql_select_db('nom_de_la_base',$db);                  // sélection de la base

   // on regarde si l'url existe déjà
   $sql = "SELECT id FROM infos_tbl WHERE url='$url'";
   $req = mysql_query($sql) or die('Erreur SQL !'.$sql.'<br>'.mysql_error());

   // on compte le nombre de résultats
   $res = mysql_num_rows($req);
   if($res!=0)  // l'url existe déjà, on affiche un message d'erreur
      {
      echo '<font color="red">Désolé, mais cette URL existe déjà dans notre
base.</font>';
      }
   else  // L'url n'existe pas, on insère les informations du formulaire dans la table
      {
      $sql = "INSERT INTO infos_tbl(id, nom, prenom,email, icq, titre,
url) VALUES('','$nom','$prenom','$email','$icq','$titre','$url')";
      mysql_query($sql) or die('Erreur SQL !'.$sql.'<br>'.mysql_error());

      // on affiche le résultat pour le visiteur
      echo 'Vos infos on été ajoutées.';
      }
   mysql_close();  // on ferme la connexion
   }
?>
```

Alimenter deux tables et créer une liaison

Voyons maintenant comment ces mêmes informations peuvent être enregistrées dans deux tables différentes en gardant une liaison entre les deux. Le principe est en fait simple, admettons que nous ayons une première table qui va nous servir à stocker les coordonnées du visiteur (**$nom**, **$prenom**, **$email**, **$icq**) et une seconde dans laquelle ne seront sauvegardées que les informations du site (**$titre** et **$url**).

Examinons la méthode à employer :

1. On commence par insérer **$titre** et **$url** dans la table 2 (**sites_tbl**).
2. Une fois l'insertion effectuée, on utilise la **fonction PHP mysql_insert_id()** pour connaître l'**id_site** qui a été affecté à notre nouvel enregistrement.
3. On insère le reste du formulaire dans la table 1 (**infos_tbl**), soit : **$nom**, **$prenom**, **$email**, **$icq** et **$id_site**.

Nous avons maintenant une liaison entre les tables 1 et 2 via l'id_site. Ci-dessous le **code PHP** de cette manipulation :

Code PHP avec création de la liaison entre les deux tables

```
<?php
// On récupère les champs (voir plus haut)
$db = mysql_connect('localhost', 'login', 'password');  // connexion à la base
mysql_select_db('nom_de_la_base',$db);                  // sélection de la base
// on regarde dans la table SITES_TBL si l'url existe déjà
$sql = "SELECT id FROM sites_tbl WHERE url='$url'";
$req = mysql_query($sql) or die('Erreur SQL !'.$sql.'<br>'.mysql_error());
$res= mysql_num_rows($req);

if($res!=0)  // l'url existe déjà, on affiche un message d'erreur
   {
   echo '<font color="red">Désolé, mais cette URL existe déjà dans notre
base.</font>';
   }
else  // L'url n'existe pas, on insère d'abord les infos dans SITES_TBL
   {
   $sql = "INSERT INTO sites_tbl(id_site, titre, url) VALUES('','$titre','$url')";
   mysql_query($sql) or die('Erreur SQL !'.$sql.'<br>'.mysql_error());
   // on récupère l'id_site qui vient d'être généré
   $id_site = mysql_insert_id();
   // ci-dessous on insère les infos dans INFOS_TBL
   $sql = "INSERT INTO infos_tbl(id, nom, prenom,email, icq,
id_site) VALUES('','$nom','$prenom','$email','$icq','$id_site')";
   mysql_query($sql) or die('Erreur SQL !'.$sql.'<br>'.mysql_error());
   }
mysql_close($db);  // on ferme la connexion
?>
```

Important : L'id_site de la table 2 (**sites_tbl**) doit être en **auto_incrément automatique** au même titre que l'id de la table 1 (**infos_tbl**). Je ne vous donne volontairement pas la requête SQL pour créer les deux tables, et ce pour vous faire travailler un peu tout de même.

Conclusion :

Vous pouvez créer autant de liaisons que vous le voulez entre vos tables, cela a pour but de les alléger ce qui permet une plus grande rapidité lorsque vous souhaitez les interroger.

18)Les pseudos-frames

Cet tutorial a pour but de répondre aux questions fréquentes posées à savoir : Comment faire un site avec des url du type **domain.com?page=news**. Ce type d'architecture s'appelle les **pseudo-frames**, car elles remplacent plus qu'avantageusement l'archaïque système de **frame html**. A savoir que les **url** de ce type ne servent pas qu'à la création de **pseudo-frames**, mais c'est le rôle qu'elles auront ici.

L'inclusion de fichier

Quelques explications

Les **pseudo-frames** permettent de découper votre site en plusieurs parties, chacune dans un fichier différent, puis les regrouper dans une page. Et bien afin de faciliter les mises à jour, tout comme c'était le cas avec les **frames originaux**. En effet imaginons que vous avez un menu sur votre site. Ce menu est affiché sur toutes les pages de votre site. Si vous deviez le recopier dans chaque page, cela voudrait dire qu'à chaque mise à jour du menu, il faudrait aller le corriger dans chaque page. Avec les **pseudo-frames**, ce problème ne se pose plus.Vous créez un fichier **menu.html**, et vous appeler ce fichier dans chaque page. Lorsque vous mettez à jour le fichier **menu.html**, votre menu est mis à jour sur chaque page automatiquement. Cela n'explique pas encore la raison du **page=news** dans l'**url**, mais cela va nous servir de point de départ.

Un peu de pratique

Nous allons donc créer notre menu dans le **fichier menu.html**. Il s'agit ici simplement d'un peu de **html**, notez qu'il n'y a pas de balise **<head>, <html>** ou **<body>**, car comme nous l'avons vu, cette page sera insérée dans une autre qui, elle, contiendra ces balises. On n'y met donc que le strict nécessaire. Notez également que ce code sera modifé par la suite, pour le moment on se contente d'un menu simple, sans **pseudo-frames**.

Le code de la page menu.html

```
<ul>
  <li>
    <a href="accueil.php" title="L'accueil">Accueil</a>
  </li>
  <li>
    <a href="news.php" title="Les News ">News</a>
  </li>
</ul>
```

Puis nous allons créer deux autres pages pour notre site, la page **accueil.php** et la page **news.php** (qui sont les pages sur lesquelles pointe notre menu).

Le code de la page accueil.php

```
<div class="colonneGauche">
<?php
  include('menu.html');  // Nous appelons notre menu
?>
</div>
<div class="colonneDroite">
  <p>Ici nous aurions le contenu de notre page d'accueil.</p>
</div>
```

Le code de la page news.php

```
<div class="colonneGauche" >
<?php
  include('menu.html');  // Nous appelons notre menu
?>
</div>
<div class="colonneDroite">
  <p>Ici nous aurions les news de notre site.</p>
</div>
```

Explication : comme vous pouvez le voir, le processus est le même sur ces deux pages. Nous utilisons la structure de langage **include()** qui permet d'inclure un ficher et d'exécuter son contenu. Là en l'occurence il s'agit simplement de **html**, donc il n'a rien à exécuter mais cela aurait très bien pu être du **PHP** comme nous allons le voir plus loin. Je vous invite à vous documenter sur cette structure de langage dans la **doc PHP** afin d'en connaître le fonctionnement.

Ce que nous avons fait aura pour effet d'afficher le menu sur ces deux pages. Les avantages sont que le rendu final se fera sur une page unique, et, comme nous l'avons vu plus haut, en cas de modification du menu, vous ne modifiez que **menu.html**. Comme c'est le même fichier qui est inclus à chaque fois, les changements se répercuteront automatiquement sur tout le site (du moins partout où vous avez inclus ce fichier).

Tout ceci est bien beau mais cela ne nous explique pas la raison du paramètre d'**url page=news**, vu plus haut.

Les pseudos-frames

Encore quelques explications

Nous avons donc vu qu'il était possible et très facile d'inclure un fichier dans un autre fichier en **PHP.** Cela va nous ouvrir la voie des **pseudos-frames**. En effet imaginons que nous soyons capable de prévoir quelle page doit être incluse et quand, nous pourrions découper tout notre site en plusieurs morceaux tout gardant une page centrale qui se chargerait d'appeler le bon morceau au bon moment. C'est ici que va nous servir le paramètre d'**url**. En effet c'est lui va nous permettre de savoir que quand l'utilisateur clique sur tel lien, c'est tel page qu'il faut appeler et afficher.

La preuve par l'exemple

Veuillez noter que le code qui va suivre contient un énorme trou de sécurité. Il n'est qu'à titre d'exemple et d'apprentissage par pallier. Le code sécurisé suit.

Nous allons commencer par créer les morceaux de pages. Ici nous avons besoin de l'accueil et des news. Le menu lui aussi va être légèrement modifié, vous comprendrez plus tard pourquoi.

Le code de la page accueil.php

```
<p>Mettez ici simplement le contenu que vous voulez voir afficher en accueil.</p>
```

Le code de la page news.php

```
<p>Mettez ici simplement le contenu que vous voulez voir afficher dans les news. Il y a de forte chance que cela soit du code PHP qui vous génère ces news. Cela ne pose aucun problème, procédez comme à votre habitude: requête, boucle et echo.</p>
```

Le code de la page menu.htm

```
<ul>
  <li>
    <a href="index.php?page=accueil" title="L'accueil">Accueil</a>
  </li>
  <li>
    <a href="index.php?page=news" title="Les News ">News</a>
  </li>
</ul>
```

Nous allons maintenant créer une page **index.php** qui va être le noyau de notre site. Elle va se charger d'inclure tout les éléments découpés afin de construire une page complète.

Le code de la page index.php

```
<div class="entete" >
<?php
  include('entete.html');  // Nous appelons l'entete du site
?>
</div>
<div class="colonneGauche" >
<?php
  include('menu.html');   // Nous appelons notre menu
?>
</div>
<div class="colonneDroite">
<?php
  include($_GET['page'].'.php');   // Nous appelons le contenu central de la page
?>
</div> <div class="pied">
<?php
  include('pied.html');   // Nous appelons le pied de page
?>
</div>
```

Comme vous l'avez sûrement compris, tout le problème consiste à savoir quand il faut inclure les **news ou l'accueil**. Notez les modifications de la page menu : les liens pointent dorénavant tous sur la page **index.php**, mais il y a plus. Ces derniers ont été complétés avec un paramètre **page** auquel nous affectons une valeur en fonction de ce que nous voulons afficher.

Sur la page **index.php** nous récupérons ce paramètre ainsi que la valeur qu'il contient à l'aide de **$_GET['page'].** Cette dernière est ce l'on appelle une variable globale. Tout comme **$_POST** permet de récupérer des valeurs passées par formulaires, **$_GET** permet de récupérer des valeurs passées dans l'**url**.

Conclusion

Donc lorsque le visiteur clique sur le lien news, le paramètre **page** avec la valeur **news** est transmis à la page **index.php**. Sur cette dernière on récupère cette valeur. Et donc l'include qui dans le code est **include($_GET['page'].'.php')** va devenir dynamiquement **include('news.php')** et c'est donc bien la page contenant le news qui va être incluse dans la page noyau. De même pour l'accueil. Cette fois le paramètre **page** sera égal à accueil et donc l'**include** sera modifié en fonction.

Problème de sécurité

Le problème

Si vous avez bien lu, vous savez que le paramètre page est affiché en clair dans la barre d'adresse du navigateur. Il est donc facilement modifiable par l'utilisateur. Ce dernier peut dès lors inclure n'importe quoi dans votre page **index.php**. Il convient donc de tester ce que vaut ce paramètre avant d'inclure la page demandée et surtout vérifier que cela correspond bien à ce que vous permettez.

Nous allons pour cela créer un tableau contenant un listing de toutes les pages que vous autorisez dans cet include.

Le code de la page index.php

```
<div class="entete" >
<?php
  include('entete.html');  // Nous appelons l'entete du site
?>
</div>
<div class="colonneGauche" >
<?php
  include('menu.html');   // Nous appelons notre menu
?>
</div>
<div class="colonneDroite">
<?php
// On définit le tableau contenant les pages autorisées
  // ------------------------------------------------
  $pageOK = array('news' => 'news.php',
                  'accueil' => 'accueil.php');
// On teste que le paramètre d'url existe et qu'il est bien autorisé
  // ------------------------------------------------------------
  if ( (isset($_GET['page'])) && (isset($pageOK[$_GET['page']])) ) {
    include($pageOK[$_GET['page']]);   // Nous appelons le contenu central de la page
  } else {
```

```
   include('accueil.php');   // Page par défaut quant elle n'existe pas dans le tableau
  }
?>
</div> <div class="pied">
<?php
  include('pied.html');   // Nous appelons le pied de page
?>
</div>
```

Explication

Comme vous pouvez le voir, nous avons créé un **tableau associatif**. Ce tableau est construit ainsi : son index représente les **paramètres d'inclusion** autorisés dans l'**url**, les valeurs représentent le chemin réel du fichier. En l'occurence les fichiers sont dans le même répertoire que la page **index.php**, mais vous auriez pu vouloir inclure un fichier dans le répertoire **colGauche** contenu dans le répertoire courant. Dans ce cas vous auriez dû rajouter une ligne au tableau de cette forme : **'page' => 'colGauche/fichier.php'**.

Ensuite nous testons l'existence de cette ligne du tableau avec comme **index du tableau**, le paramètre passé en url. Si la ligne existe c'est que le paramètre est autorisé, on inclut donc le fichier en utilisant la valeur de la ligne concernée du tableau.

Comme vous l'aviez remarqué, j'avais spécifié des classes dans mes **divs**. Elles sont là à titre purement indicatif, vous pouvez les modifier à loisir.

Conclusion

Si vous avez bien suivi vous pouvez maintenant créer des architectures de sites plus ou moins complexes qui vous faciliteront les mises à jour et le graphisme tout en restant parfaitement accessibles (si vous respectez les standards au niveau du **code HTML** bien sûr). N'oubliez jamais le problème de sécurité et le moyen d'y remédier sans quoi vous pourriez rencontrer quelques surprises.

Sachez également que vous pouvez utiliser plus d'un paramètre dans vos url, il suffit pour cela de les séparer par des **"&"** (**domain.com?paramOne=1&paramTwo=2**).

19)Les sessions PHP4

Beaucoup d'entre-vous se posent la question de savoir comment passer leur variables de pages en pages, ou encore comment conserver certaines informations pendant la durée d'une visite sur leur site. Les sessions devraient répondre à leurs attentes.

1 .Quelques petites choses à savoir sur les sessions

Une session est en fait un fichier conservé sur le serveur et accessible à vos scripts en fonction d'un identifiant généré à la création. Chaque fois qu'un de vos visiteurs génère une session, un identifiant lui est attribué. Tout ce qui est dans cette session est accessible de partout à vos scripts. On comprend dès lors très vite tout l'intérêt de la chose. En effet, si chaque session est propre à un visiteur, on peut personnaliser nos scripts en fonction du visiteur, ou encore alléger nos requêtes : plutôt que d'aller chercher un pseudonyme dans la base de données sur chaque page, vous le rapatriez à l'arrivée du visiteur sur le site, vous le stockez en session et c'est de là que vous y accéderez par la suite.

La **session** est finalement l'équivalent d'un **cookie** en plus sécurisé. En effet, étant stocké sur le serveur et non chez le client, elle est plus difficile d'accès aux éventuels pirates. Mais un risque demeure, prenez donc l'habitude de ne pas y stocker des informations trop sensibles. N'oubliez pas également que la session, à la différence du cookie, n'est valable qu'un temps limité (aux alentours des 30 minutes, mais cela dépend de la configuration de votre serveur), et est automatiquement détruite à la fermeture du navigateur du visiteur.

Vous pouvez enregistrer en session tout type de variable : du simple numérique au tableau en passant par la chaîne de caractères. Il existe plusieurs fonctions liées aux sessions, mais deux seulement sont essentielles à leur fonctionnement, ainsi qu'un tableau :

session_start();
Cette fonction sert à démarrer une session ou appeler la session existante. Elle doit donc être présente sur toutes les pages de votre site. A savoir que cette fonction ne tolère pas d'envoi au navigateur avant elle. Plus simplement il ne faut aucune sortie avant elle : pas de html, pas de echo. Prenez l'habitude de la placer au début du fichier, avant quoi que ce soit d'autre et tout se passera bien.

session_destroy();
Cette fonction détruit la session en cours. Mais elle ne détruit pas les **variables de sessions** associées à la session courante. Nous verrons plus loin comment détruire une session complètement et proprement.

$_SESSION
Il s'agit du tableau global contenant toutes les **variables de sessions** pour la session courante. Son utilisation est exactement identique aux tableaux habituels, à savoir :

Utilisation du tableau global $_SESSION

```
// Nous allons créer une variable de session appellée "nombre" et qui contient "1234" :
$_SESSION['nombre'] = 1234;// Nous pouvons maintenant afficher la variable "nombre"
echo $_SESSION['nombre'];  // Ceci va afficher 1234
```

2. Etude de cas concret

Nous allons maintenant étudier un cas concret à savoir une procédure de login. En effet c'est un cas répandu d'utilisation des sessions. Nous allons donc créer un formulaire qui permet au visiteur de se loguer, puis une page de vérification qui va consulter la **base de données** et vérifier que les informations de connexion sont correctes. Et enfin mettre les données utiles sur l'utilisateur en session afin de pouvoir les réutiliser sur d'autres pages du site.

Le code HTML de la page de connexion	Le rendu

`<html>`
`<head>`
`<title>Connexion au site</title>`
`</head>`
`<body>`
	Votre login [ ]
	Votre mot de passe [ ]
	login

```
 <form method="post"
action="verifLogin.php">
  <table border="0" width="400"
align="center">
   <tr>
    <td width="200"><b>Vôtre
login</b></td>
    <td width="200">
     <input type="text"
name="login">
    </td>
   </tr>
   <tr>
    <td width="200"><b>Vôtre mot de
passe<b></td>
    <td width="200">
     <input type="password"
name="password">
    </td>
   </tr>
   <tr>
    <td colspan="2">
     <input type="submit"
name="submit" value="login">
    </td>
   </tr>
  </table>
 </form>
</body>
</html>
```

Nous allons maintenant passer à la récupération des données et au test de celles-ci.

Code de la page verifLogin.php

```
<?php
// On démarre la session
session_start();
$loginOK = false;  // cf Astuce
// On n'effectue les traitements qu'à la condition que
// les informations aient été effectivement postées
if ( isset($_POST) && (!empty($_POST['login'])) && (!empty($_POST['password']))
) {
extract($_POST);  // je vous renvoie à la doc de cette fonction
 // On va chercher le mot de passe afférent à ce login
  $sql = "SELECT pseudo, age, sexe, ville, mdp FROM user WHERE login =
'".addslashes($login)."'";
  $req = mysql_query($sql) or die('Erreur SQL : <br />'.$sql);
  // On vérifie que l'utilisateur existe bien
  if (mysql_num_rows($req) > 0) {
    $data = mysql_fetch_assoc($req);
```

```
    // On vérifie que son mot de passe est correct
    if ($password == $data['mdp']) {
      $loginOK = true;
    }
  }
}
// Si le login a été validé on met les données en sessions
if ($loginOK) {
  $_SESSION['pseudo'] = $data['pseudo'];
  $_SESSION['age'] = $data['age'];
  $_SESSION['sexe'] = $data['sexe'];
  $_SESSION['ville'] = $data['ville'];
}
else {
  echo 'Une erreur est survenue, veuillez réessayer !';
}
?>
```

Si l'utilisateur est loggué, nous avons ses informations à disposition partout sur le site. On part de l'idée que le login ne fonctionne pas en initialisant une variable à **false**. Et c'est seulement lorsque le login est confirmé qu'on modifie cette variable à **true**. Il suffit ensuite de tester cette variable pour logguer l'utilisateur ou pas. Cela nous économise des **else** inutiles à chaque étape de test. Remarquez que je n'ai effectué que les tests minimums sur le login avant de lancer la requête. Il conviendrait d'effectuer plus de tests afin d'éviter d'éventuelles tentatives d'intrusions dans votre application. Imaginons maintenant que nous voulions afficher ces informations plus tard, sur une autre page. Sans les sessions nous aurions dû faire une requête qui retournerait chercher le tout, tandis que là, il nous suffit de faire cela :

Code de la page affichInfo.php

```
<?php
// On appelle la session
session_start();
// On affiche une phrase résumant les infos sur l'utilisateur courant
echo 'Pseudo : ',$_SESSION['pseudo'],'<br />
    Age : ',$_SESSION['age'],'<br />
    Sexe : ',$_SESSION['sexe'],'<br />
    Ville : ',$_SESSION['ville'],'<br />';
?>
```

Notez que le **session_start()** est toujours présent. En effet il faut préciser au script que nous allons utiliser une session. C'est l'interpréteur qui va déterminer s'il faut en démarrer une ou sélectionner celle existante. Bien entendu ce script n'a qu'une utilté réduite, à vous de voir quelles pourraient être les multiples façons de profiter des sessions.

3. Déconnexion

Il s'agit là d'un point important. En effet une **session reste ouverte tant que le visiteur ne ferme pas son navigateur**. Vous devez lui offrir la possibilité de se délogguer d'une autre manière. Il vous suffit en fait de faire un lien appelé par exemple **déconnexion** qui pointe sur un script contenant ceci :

Code de la page logout.php

```
<?php
// On appelle la session
session_start();
// On écrase le tableau de session
$_SESSION = array();
// On détruit la session
session_destroy();
?>
```

20) Affichage page par page

Je vais vous apprendre à créer simplement un affichage page par page à partir du résultat d'une **requête SQL**.

La table SQL.

Pour notre exercice, nous allons utiliser une table assez simple.

Requête de création et d'insertion des données

```
CREATE TABLE vaches (
 id int(11) NOT NULL auto_increment,
 surnom varchar(50) NOT NULL default '',
 prenom varchar(20) NOT NULL default '',
 PRIMARY KEY  (id)
) TYPE=MyISAM;

INSERT INTO vaches VALUES (1, 'Qui aime les frites', 'Marguerite');
INSERT INTO vaches VALUES (2, 'Belle clochette', 'Roussette');
INSERT INTO vaches VALUES (3, 'La starlette', 'Blanchette');
INSERT INTO vaches VALUES (4, 'Petite Pataude', 'Noireaude');
INSERT INTO vaches VALUES (5, 'Toujours chouette', 'Paquerette');
INSERT INTO vaches VALUES (6, 'Tête de linotte', 'Cowpilot');
INSERT INTO vaches VALUES (7, 'Qui saute partout', 'Kancowroo');
INSERT INTO vaches VALUES (8, 'La gentille', 'Cowchenille');
INSERT INTO vaches VALUES (9, 'Le plus bizarre des cows', 'Haricow');
```

Première étape : afficher tous les enregistrements dans une table HTML

Je vais vous expliquer comment faire pour afficher des résultats dans une table html.

Affichage des enregistrements dans une table HTML.

```
<html>
<body>
<?php
// information pour la connection à le DB
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'test';
// connection à la DB
$link = mysql_connect ($host,$user,$pass) or die ('Erreur : '.mysql_error() );
mysql_select_db($db) or die ('Erreur :'.mysql_error());
// requête SQL qui compte le nombre total d'enregistrements dans la table et qui
//récupère tous les enregistrements
$select = 'select prenom,surnom from vaches';
$result = mysql_query($select,$link) or die ('Erreur : '.mysql_error() );
$total = mysql_num_rows($result);
// si on a récupéré un résultat on l'affiche.
if($total) {
   // début du tableau
   echo '<table bgcolor="#FFFFFF">'."\n";
      // première ligne on affiche les titres prénom et surnom dans 2 colonnes
      echo '<tr>';
      echo '<td bgcolor="#669999"><b><u>Prénom</u></b></td>';
      echo '<td bgcolor="#669999"><b><u>Surnom</u></b></td>';
      echo '</tr>'."\n";
   // lecture et affichage des résultats sur 2 colonnes, 1 résultat par ligne.
   while($row = mysql_fetch_array($result)) {
      echo '<tr>';
      echo '<td bgcolor="#CCCCCC">'.$row['prenom'].'</td>';
      echo '<td bgcolor="#CCCCCC">'.$row['surnom'].'</td>';
      echo '</tr>'."\n";
   }
   echo '</table>'."\n";
   // fin du tableau.
}
else echo 'Pas d\'enregistrements dans cette table...';
// on libère le résultat
mysql_free_result($result);
?>
</body>
</html>
```

Le résultat de ce code donnera ceci à l'écran :

Prénom	**Surnom**
Marguerite	Qui aime les frites
Roussette	Belle clochette
Blanchette	La starlette
Noireaude	Petite Pataude

Paquerette	Toujours chouette
Cowpilot	Tête de linotte
Kancowroo	Qui saute partout
Cowchenille	La gentille
Haricow	Le plus bizarre des cows

Explications :

Juste avant notre boucle, nous affichons une ligne de la **table HTML**. Elle est constituée de deux colonnes dans lesquelles nous mettons le titre de chacune d'elles sous cette forme :

Affichage des titres dans des colonnes d'une table HTML

```
<tr>
<td bgcolor="#669999"><b><u>Prénom</u></b></td>
  <td bgcolor="#669999"><b><u>Surnom</u></b></td>
  </tr>;
```

Dans la boucle, nous allons afficher chaque enregistrement dans une ligne du tableau.
Le prénom dans une cellule et le surnom dans une autre, sous cette forme :

Affichage des données

```
  <tr>
  <td bgcolor="#CCCCCC"><b><u>Roussette</u></b></td>
  <td bgcolor="#CCCCCC"><b><u>Belle clochette</u></b></td>
  </tr>;
```

Explication d'une option d'un select SQL : Limit

Bien pratique, cette option nous permet de sélectionner une partie des enregistrements de la table en lui donnant le nombre d'enregistrements à récupérer à partir d'une position donnée. Voici quelques exemples :

Syntaxe d'un select avec une limite - Exemple 1

```
Select  prenom,surnom from vache limit 0,4;
```

Cette requête retournera 4 enregistrements à partir du premier de la liste (0 + 1).
Dans le cas de notre table :

Prénom	Surnom

Margueritte	Qui aime les frites
Roussette	Belle clochette
Blanchette	La starlette
Noireaude	Petite Pataude

Syntaxe d'un select avec une limite - Exemple 2

```
select prenom,surnom from vache order by prenom asc limit 0,4;
```

Cette requête retournera quatre enregistrements à partir du premier de la liste dans l'ordre alphabétique. Donc, la requête trie les enregistrements et ensuite en prend 4 à partir du premier (0 + 1). Dans le cas de notre table :

Prénom	**Surnom**
Blanchette	La starlette
Cowchenille	La gentille
Cowpilot	Tête de linotte
Haricow	Le plus bizarre des cows

Syntaxe d'un select avec une limite - Exemple 2

```
Select prenom,surnom from vache order by prenom asc limit 5,4;
```

Cette requête retournera quatre enregistrements à partir du 6ème (on en saute 5, 5 +1) dans l'ordre alphabétique.Voilà ce que donne le résultat :

Prénom	**Surnom**
Margueritte	Qui aime les frites
Noireaude	Petite Pataude
Paquerette	Toujours chouette
Roussette	Belle clochette

Si vous voulez comparer avec ce que donnerait un select sans limit avec un tri alphabétique :

Comparaison de l'exemple 2 sans limit

```
Select  prenom,surnom from vache order by prenom asc;
```

Voilà le résultat :

Prénom	**Surnom**
Blanchette	La starlette
Cowchenille	La gentille
Cowpilot	Tête de linotte
Haricow	Le plus bizarre des cows
Kancowroo	Qui saute partout

Margueritte	Qui aime les frites
Noireaude	Petite Pataude
Paquerette	Toujours chouette
Roussette	Belle clochette

Notre affichage page par page.

Je vais reprendre chaque partie du script de base pour le modifier afin de créer un affichage page par page.

Première partie du script

```
// information pour la connection à le DB
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'test';
 // connection à la DB
$link = mysql_connect ($host,$user,$pass) or die ('Erreur : '.mysql_error() );
mysql_select_db($db) or die ('Erreur :'.mysql_error());
```

Explications :

Cette partie-là ne change pas et ne comporte que des choses que vous connaissez.

- **$host**,**$pass**,**$user** et **$db** contiennent les informations pour se connecter à la base de données.
- Ensuite on se connecte à la **base de données**.

De quelles variables avons-nous besoin ?

Pour notre **affichage page par page**, nous avons besoin de connaître certaines données que l'on va stocker dans des variables.

- **$nombre** : Nombre d'enregistrements que l'on veut afficher par page.
- **$total** : Le nombre total d'enregistrements de la table.
- **$limite** : Où en sommes-nous dans notre affichage.

Nous allons reprendre notre script de base et y ajouter ce dont nous avons besoin pour travailler. Au début du script nous allons initialiser les variables **$nombre** et **$limite**.

Initialisation des variables.

```
$nombre = 5;  // on va afficher 5 résultats par page.
if (!isset($limite)) $limite = 0; // si on arrive sur la page pour la première fois
// on met limite à 0.
$path_parts = pathinfo($_SERVER['PHP_SELF']);
$page = $path_parts['basename'];
```

Explications :

- **$nombre** est initialisé à 5 (5 enregistrements affichés à la fois)
- Si **$limite** n'existe pas on l'initialise à 0.
- Les deux lignes de codes suivantes permettent de récupérer le nom de la page qui sera utilisée dans les liens.
- La variable **$total** contient le nombre d'enregistrement à afficher au total.

Compte le nombre d'enregistrements total de la table.

```
$select = 'SELECT count(id) FROM vaches';
$result = mysql_query($select,$link) or die ('Erreur : '.mysql_error() );
$row = mysql_fetch_row($result);
$total = $row[0];
```

Ajout de la requête select avec le limit

Maintenant, nous avons besoin de faire notre requête de sélection qui va afficher **$nombre** enregistrements à partir de **$limite**.

Sélectionne la portion d'enregistrements à afficher.

```
// requête SQL qui ne prend que le nombre d'enregistrements nécessaire à l'affichage
$select = 'select prenom,surnom FROM vaches ORDER BY prenom ASC limit '.$limite.','.$nombre;
$result = mysql_query($select,$link)  or die ('Erreur : '.mysql_error() );
```

Explications

On utilise l'opérateur de concaténation pour compléter notre requête de base par le tri et la limite de sélection. Notre requête donnera donc au premier affichage de la page (**$limite** étant initialisé à 0 la première fois):

Requête de sélection de la première page

```
Select  prenom,surnom from vaches order by prenom asc limit 0,5
```

Qui donnera ceci à l'écran :

Prénom	Surnom
Blanchette	La starlette
Cowchenille	La gentille
Cowpilot	Tête de linotte
Haricow	Le plus bizarre des cows
Kancowroo	Qui saute partout

Calcul de la nouvelle limite d'affichage

Imaginons que nous avons affiché la page pour la première fois. Notre **$limite** est à zéro, et notre script affiche donc cinq enregistrements à partir du début. C'est pas mal, mais maintenant il faut savoir s'il reste d'autres enregistrements à afficher.Si oui il nous faut mettre un lien ou un bouton permettant de les afficher. Et pour cela, il faut recalculer la limite d'affichage. Nous voulons afficher les **$nombre** enregistrements suivants et/ou les **$nombre** enregistrements précédents: il s'agit d'un calcul très simple.

Calcul des limites précédente et suivante

```
$limitesuivante = $limite + $nombre;
$limiteprecedente = $limite - $nombre;
```

Voilà à quoi notre **requête SQL** ressemblera si on est sur la première page et que l'on clique sur le lien suivant.

Requête de sélection de la deuxième page

```
SELECT prenom,surnom FROM vaches ORDER BY prenom ASC limit 5,5
```

Qui donnera ceci à l'écran :

Prénom	Surnom
Margueritte	Qui aime les frites
Noireaude	Petite Pataude
Paquerette	Toujours chouette
Roussette	Belle clochette

Pour avoir la limite d'affichage de la portion suivante, on ajoute **$nombre** à **$limite**
Pour avoir la limite d'affichage de la portion précédente, on retire **$nombre** à **$limite**

Affichage des boutons ou liens page précédente, page suivante

Maintenant, il va nous falloir afficher les liens ou les boutons pour afficher le reste des enregistrements. Pour cela nous allons devoir réfléchir à deux choses.

- Sommes-nous sur la première page de notre affichage ? C'est à dire est ce que **$limite** vaut 0.
  - oui : il ne faut donc pas afficher de bouton ou de lien **page précédente**.
  - non : il faut afficher un bouton ou un lien **page précédente**.

Affichage du lien précedent si besoin

```
if($limite != 0) {
   echo '<a href="'.$page.'?limite='.$limiteprecedente.'">Page précédente</a>';
}
```

Ou bien affichage d'un bouton formulaire

```
if($limite != 0) {
   echo '<form action="'.$page.'" method="post">';
   echo '<input type="submit" value="précédents">';
   echo '<input type="hidden" value="'.$limiteprecedente.'" name="limite">';
   echo '</form>';
}
```

Y a-t-il encore une page à afficher après la page courante? C'est à dire : est-ce que notre limite recalculée (**$limitesuivante**) est plus petite que notre **$total** ?

- oui : il faut donc afficher un bouton ou un **lien page suivante**.
- non : il ne faut pas afficher de bouton ou de **lien page suivante** car on est arrivé à la fin de notre affichage.

Affichage du lien suivant si besoin

```
if($limitesuivante < $total) {
   echo '<a href="'.$page.'?limite='.$limitesuivante.'">Page Suivante</a>';
}
```

Ou bien affichage d'un bouton formulaire

```
//Ou bien affichage d'un bouton formulaire
if($limitesuivante < $total) {
   echo '<form action="'.$page.'" method="post">';
   echo '<input type="submit" value="suivants">';
   echo '<input type="hidden" value="'.$limitesuivante.'" name="limite">';
   echo '</form>';
}
```

Maintenant, Si vous voulez afficher + de cinq enregistrements par page, il vous suffit de changer la valeur de **$nombre**.

<mode type="hors sujet" voix="voix d'hôtesse">
Nos hôtesses vont passer vous proposer des pizzas et du coca ou des jus de fruits. C'est le moment de faire une petite pause avant d'aller plus loin.
</mode>

Comment utiliser ce que nous venons d'apprendre de facon intelligente. Il serait particulièrement intéressant de créer des fonctions pour notre affichage page/page pour pouvoir les réutiliser dans tous nos scripts.

<mode type="hors sujet" voix="voix d'hôtesse">
Le capitaine vous prie de bien vouloir attacher vos ceintures, nous passons à la vitesse supérieure.
</mode>

Valider la $limite passée par l'url.

Comme pour tous les scripts, il est important de penser à la sécurité, particulièrement quand des variables sont passées par **url** ou par un formulaire. Nous devons récupérer la variable puis tester si la valeur de **$limite** :

- est bien numérique.
- se trouve dans une fourchette de valeurs possibles (de 0 à **$max**)
- est un multiple de **$nombre**.

Pourquoi faire tout ces tests ? Parce qu'il ne faut jamais faire confiance à l'utilisateur, ni aux données qu'il peut entrer dans un formulaire ou via l'url.

Fonction qui vérifie la validité de $limite

```
//récupération de $limite
if(isset($_GET['limite']))
$limite=$_GET['limite'];
else   $limite=0;
function verifLimite($limite,$total,$nombre) {
// je verifie si limite est un nombre.
if(is_numeric($limite)) {
// si $limite est entre 0 et $total, $limite est ok
// sinon $limite n'est pas valide.
if(($limite >=0) && ($limite <= $total) && (($limite%$nombre)==0)) {
// j'assigne 1 à $valide si $limite est entre 0 et $max
$valide = 1;
}
else {
// sinon j'assigne 0 à $valide
$valide = 0;
}}
else {
 // si $limite n'est pas numérique j'assigne 0 à $valide
 $valide = 0;
} // je renvois $valide
return $valide;
}
```

Explication des arguments passés en paramètres à la fonction verifLimite

- **$limite** : la limite passée en paramètre.
- **$total** : le nombre total d'enregistrements à afficher.
- **$nombre** : nombre total d'enregistrements à afficher par page.

Fonction qui affiche les boutons précédent/suivant

```
<?php
function displayNextPreviousButtons($limite,$total,$nb,$page) {
$limiteSuivante = $limite + $nb;
$limitePrecedente = $limite - $nb;
echo  '<table><tr>'."\n";
```

```
if($limite != 0) {
        echo  '<td valign="top">'."\n";
        echo  '<form action="'.$page.'" method="post">'."\n";
        echo  '<input type="submit" value="précédents">'."\n";
        echo  '<input type="hidden" value="'.$limitePrecedente.'"
name="limite">'."\n";
        echo  '</form>'."\n";
        echo  '</td>'."\n";
}
if($limiteSuivante < $total) {
        echo  '<td valign="top">'."\n";
        echo  '<form action="'.$page.'" method="post">'."\n";
        echo  '<input type="submit" value="suivants">'."\n";
        echo  '<input type="hidden" value="'.$limiteSuivante.'"
name="limite">'."\n";
        echo  '</form>'."\n";
        echo  '</td>'."\n";
}
echo  '</tr></table>'."\n";
}
```

Explication des arguments passés en paramètres à la fonction displayNextPreviousButtons.

- **$limite** : la limite d'affichage courante.
- **$total** : nombre total d'enregistrements retournés par le query
- **$nb** : nombre d'enregistrements à afficher par page.
- **$page** : nom de la page php.

Explication de la fonction displayNextPreviousButtons.

Elle se comporte exactement comme les bouts de script expliqués ci-dessus. On calcule la limite précédente et suivante, et, selon la page sur laquelle on se trouve ainsi que le nombre d'enregistrements total à afficher, on affiche ou pas les boutons **next/previous.**

Comment afficher les numéros de page.

Vous vous demandez sûrement comment faire pour afficher des liens vers les différents **pages à afficher.** Cette petite fonction va vous aider.

Fonction qui affiche les liens vers les pages

```
function affichePages($nb,$page,$total) {
        $nbpages=ceil($total/$nb);
        $numeroPages = 1;
        $compteurPages = 1;
        $limite  = 0;
        echo '<table border = "0" ><tr>'."\n";
        while($numeroPages <= $nbpages) {
```

```
        echo '<td ><a href =
"'.$page.'?limite='.$limite.'">'.$numeroPages.'</a></td>'."\n";
        $limite = $limite + $nb;
        $numeroPages = $numeroPages + 1;
        $compteurPages = $compteurPages + 1;
           if($compteurPages == 10) {
           $compteurPages = 1;
           echo '<br>'."\n";
           }
        }
        echo '</tr></table>'."\n";
}
```

Explication des arguments passés en paramètres à la fonction affichePages

- **$nb** : $nb : nombre d'enregistrements à afficher par page.
- **$page** : nom de la page php.
- **$total** : nombre total d'enregistrements retournés par le **query**

Explication de la fonction affichePages

1. On calcule le nombre de pages en divisant le nombre total d'enregistrements (retournés par le **query**) par le nombre d'enregistrements à afficher par page.Si nous avons 20 enregistrements à afficher il nous faudra 20/5 = 4 pages. Si nous avons 21 enregistrements à afficher, comme 21/5 =4,2, cinq pages seront nécessaires. On doit donc arrondir à l'entier supérieur grâce à la fonction **ceil()**;
2. On initialise **$numeroPages** à **1** (première page)
3. On fait une boucle qui va faire varier **$numeroPages** tant qu'il est inférieur ou égal à **$nbpages**.
4. A chaque passage dans la boucle, on incrémente **$numeroPages**.
5. Pour ne pas se retrouver avec trop de numéros de pages sur la même ligne, on met en place un compteur **$compteurPages** qui va varier dans la boucle lui aussi.
6. Une fois que sa valeur arrivera à **10**, on ajoute un retour à la ligne et on remet notre compteur à **1**.

Modification de notre script de base afin qu'il utilise nos deux nouvelles fonctions.

Si le nombre total d'enregistrements à afficher est plus grand que le nombre d'enregistrements à afficher par page, on appelle la fonction qui affiche les numéros de page ainsi que celle qui affiche les boutons.

Modification du script de base pour utiliser nos deux fonctions

```
<html>
<body>
<?php
//=======================================
```

```php
// includes du fichier fonctions
//=========================================
require 'fonctions.php';
//=========================================
// information pour la connection à le DB
//=========================================
$host = 'localhost';
$user = 'root';
$pass = '';
$db = 'test';
//=========================================
// initialisation des variables
//=========================================
// on va afficher 5 résultats par page.
$nombre = 5;
// si limite n'existe pas on l'initialise à zéro
if (!$limite) $limite = 0;
// on cherche le nom de la page.
$path_parts = pathinfo($_SERVER['PHP_SELF']);
$page = $path_parts["basename"];
//=========================================
// connection à la DB
//=========================================
$link = mysql_connect ($host,$user,$pass) or die ('Erreur : '.mysql_error() );
mysql_select_db($db) or die ('Erreur :'.mysql_error());
//=========================================
// requête SQL qui compte le nombre total
// d'enregistrements dans la table.
//=========================================
$select = 'SELECT count(id) FROM vaches';
$result = mysql_query($select,$link)  or die ('Erreur : '.mysql_error() );
$row = mysql_fetch_row($result);
$total = $row[0];
//=========================================
// vérifier la validité de notre variable
// $limite;
//=========================================
$verifLimite= verifLimite($limite,$total,$nombre);
// si la limite passée n'est pas valide on la remet à zéro
if(!$verifLimite)  {
   $limite = 0;
}
//=========================================
// requête SQL qui ne prend que le nombre
// d'enregistrement necessaire à l'affichage.
//=========================================
$select = 'select prenom,surnom FROM vaches ORDER BY prenom ASC limit
'.$limite.','.$nombre;
$result = mysql_query($select,$link)  or die ('Erreur : '.mysql_error() );
//=========================================
```

```
// si on a récupéré un resultat on l'affiche.
//=========================================
if($total) {
   // début du tableau
   echo '<table bgcolor="#FFFFFF">'."\n";
      // première ligne on affiche les titres prénom et surnom dans 2 colonnes
      echo '<tr>';
      echo '<td bgcolor="#669999"><b><u>Prénom</u></b></td>';
      echo '<td bgcolor="#669999"><b><u>Surnom</u></b></td>';
      echo '</tr>'."\n";
   // lecture et affichage des résultats sur 2 colonnes
 while($row = mysql_fetch_array($result)) {
      echo '<tr>';
      echo '<td bgcolor="#CCCCCC">'.$row['prenom'].'</td>';
      echo '<td bgcolor="#CCCCCC">'.$row['surnom'].'</td>';
      echo '</tr>'."\n";}
 echo '</table>'."\n";
} else echo 'Pas d\'enregistrements dans cette table...';
mysql_free_result($result);
//=========================================
// si le nombre d'enregistrement à afficher
// est plus grand que $nombre
//=========================================
if($total > $nombre) {
// affichage des liens vers les pages
 affichePages($nombre,$page,$total);
// affichage des boutons
displayNextPreviousButtons($limite,$total,$nombre,$page);
}
?>
</body>
</html>
```

Que pourriez-vous ajouter?

- Signaler sur quelle page l'utilisateur se trouve.
  "**vous êtes sur la page n°1**","**vous êtes sur la page n°2**",...
- Changer la **couleur du numéro de page active**.
- Afficher où on en est dans l'affichage des enregistrements :
  Affichage des 5 premiers enregistrements sur / 32.
  Affichage des 10 premiers enregistrements sur / 32.
  etc.

<mode type="hors sujet" voix="voix d'hôtesse">

J'espère que vous avez bon voyage en notre compagnie. Nous espèrons vous revoir très bientôt sur nos lignes :p. Le capitaine du tuto **jeanmichelm**.
</mode>

21)Images dynamiques

Prérequis: Les bases du **PHP**, quelques notions élémentaires sur les images.

Objectifs: apprendre à se servir des différentes fonctions de la librairie pour générer dynamiquement des images.

La **librairie GD** est un ensemble de fonctions relatives aux images. Il est ainsi possible de créer ses propres images dans un script, mais aussi d'obtenir des informations sur des images existantes, les redimensionner, etc...

Attention: Avant de commencer, il faut vérifier un certain nombre de choses. En effet, cette extension de **PHP** existe dans plusieurs versions, et les hébergeurs choisissent parfois de désactiver certaines fonctions voire même la librairie **GD** dans son intégralité, car le traitement d'images est assez lourd pour le serveur.

Il vous faudra donc vérifier quelle est la version installée chez votre hébergeur favori et quelles sont les fonctions activées. Première information importante : le phpinfo. Pour cela éxécutez le script :

phpinfo.php
```
<?php
   phpinfo();
?>
```

Si dans le listing vous obtenez un tableau similaire à celui-ci, alors vous pouvez continuer, surtout si la version est supérieure à 2.0.

GD

GD Support	enabled
GD Version	2.0 or higher
FreeType Support	enabled
FreeType Linkage	with freetype
JPG Support	enabled
PNG Support	enabled
WBMP Support	enabled

Comme on le disait précedemment, certaines fonctions peuvent avoir été désactivées. Dans ce cas, le plus simple est encore d'essayer. Si vous obtenez l'erreur:
Fatal error: **Call to undefined function, v**ous saurez pourquoi.

I) Créer une image vide

Avant tout, il faut commencer par une image vide. Dans tout le tutoriel, nous parlerons uniquement d'images **PNG et JPG**, mais pas de **GIF**, en effet **le format GIF** qui n'était plus

supporté par la **librairie GD** depuis sa version 1.3. a été réintroduit dans la version 2 mais avec des restrictions. Le **PNG** remplacera (avantageusement) le **GIF**.

imagevide.php

```
<?php
   header ("Content-type: image/png");
   $im = ImageCreate (200, 100)
         or die ("Erreur lors de la création de l'image");
   $couleur_fond = ImageColorAllocate ($im, 255, 0, 0);
   ImagePng ($im);
?>
```

Remarque importante : dans le cas de la **GD**, on doit générer cette fois des images, oubliez donc les appels à echo ou print. Par défaut, un **script PHP** génère une **page HTML**, inutile de le préciser. Dans le cas d'une image, par contre, il faut le préciser au début du script en indiquant le type d'image générée, comme par exemple **PNG** :

```
header ("Content-type: image/png");
```

Ensuite, le fonctionnement global est assez similaire à celui d'un logiciel d'images. Vous créez une nouvelle image en choisissant ses dimensions, vous choisissez une couleur de fond et vous l'enregistrez dans le format que vous désirez. Ce qui donne en **PHP** le script suivant: Voici en très peu de lignes un script qui génère un rectangle rouge (ce n'est pas grand-chose mais c'est un bon début). Quelques explications tout de même:

L'appel à la méthode **ImageCreate r**envoie une ressource nommée **$im**. Cela correspond à l'image en cours de réalisation, la variable **$im** devra être passée à toutes les fonctions de dessin. Les paramètres de la fonction **ImageCreate** correspondent respectivement à la largeur et à la **hauteur de l'image à créer** (ici 200x100). Il peut être utile d'intercepter une erreur, c'est la tâche de **or die**. On crée ensuite une couleur via la fonction **ImageColorAllocate**. Cette fonction à deux objectifs: elle crée une couleur stockée dans une variable pouvant être réutilisée ultérieurement et elle enregistre cette couleur dans la palette de l'image **$im**. Les paramètres correspondent aux composantes **Rouge, Vert et Bleu**, qui sont trois valeurs comprises entre **0 et 255**. On a donc créé ici une couleur rouge.

Important: Cette couleur étant la première à être enregistrée dans la palette de l'image, elle correspondra à la couleur de fond.

Un appel au script **imagevide.php** va donc générer une image **PNG**. Pour intégrer celle-ci à votre site web, procédez de la même façon que pour une image classique:

```
<img src="image.php">
```

Vous pouvez même passer des paramètres au script qui va générer l'image (la couleur de fond par exemple, que vous n'oublierez pas de récupérer dans votre **script image.php**:

```
<img src="image.php?couleur=bleu">
```

Voilà, vous avez la base pour dessiner. Si vous voulez approfondir ces quelques notions, voici quelques fonctions intéressantes:

- Vous pouvez remplacer **ImageCreate par ImageCreateTrueColor** pour créer une **image 32 bits** (idéal pour des photos)
- Vous pouvez remplacer **ImagePng par ImageJPEG** pour créer une image **jpeg** (n'oubliez pas de changer le **header en header ("Content-type: image/jpeg"))**
- On verra dans le deuxième exemple qu'il est possible de sauvegarder l'image ainsi générée sur le serveur.

II) Les fonctions de dessin

Tout d'abord, qui dit dessin en informatique, dit coordonnées. Le coin superieur gauche de l'image est **aux coordonnées (x=0, y=0)**, le coin inférieur droit est **(x=largeur_image, y=hauteur_image)**.

Pour dessiner vous aurez donc à donner des coordonnées aux différentes fonctions. Prenez l'habitude de respecter l'ordre des coordonnées. Par exemple, **pour dessiner un rectangle, il faudra donner deux points à la fonction correspondante. L**e premier doit correspondre au coin supérieur gauche du rectangle et le deuxième au coin inferieur droit. Evitez de faire l'inverse car certaines fonctions donneront des resultats érronés. Voici **une liste de fonctions** pour dessiner toutes sortes de formes ou placer du texte sur votre image nouvellement créée. Pour chaque fonction, vous trouverez une courte description, sa syntaxe et un lien vers la documentation officielle. L'objectif n'est pas ici de donner un exemple pour chaque fonction, mais plutôt une explication rapide. La version de la librairie à partir de laquelle une fonction est implémentée est également spécifiée.

Syntaxe avec lien vers la doc.	Version	Description
$im = ImageCreate ($largeur, $hauteur)	Toutes	Crée une image vide (256 couleurs) $im de largeur $largeur et de hauteur $hauteur. (Utiliser de préférence le format PNG)
$im = ImageCreateTrueColor ($largeur, $hauteur)	2.0.2	Idem à ImageCreate mais l'image n'est plus limitée à 256 couleurs. (Utiliser de préférence le format JPEG)
$col = ImageColorAllocate ($im, $rouge, $vert, $bleu)	Toutes	Place dans la variable $col une couleur dont les composantes sont $rouge, $vert, $bleu (compris entre 0 et 255). Voir la partie I pour plus de détails.
ImageEllipse ($im, $x, $y, $l, $h, $col)	2.0.2	Dessine dans l'image $im une ellipse en partant du point ($x,$y), de largeur $l, de hauteur $h et de couleur $col.
ImageFilledEllipse ($im, $x, $y, $l, $h, $col)	2.0.2	Idem à ImageEllipse mais l'ellipse est remplie par la couleur $col
ImageFill ($im, $x, $y, $col)	Toutes	Colorie dans l'image $im avec la couleur $col un rectangle placé entre ($x,$y) et le coin inférieur droit de l'image.
ImageLine ($im, $x1, $y1, $x2, $y2, $col)	Toutes	Trace dans $im une ligne de couleur $col entre les points ($x1,$y1) et ($x2,$y2)

ImagePolygon ($im, $points, $num_points, $col)	Toutes	Trace un polygone dans $im dont la liste des points est donnée dans le tableau $points ($points[0]=$x0, $points[1]=$y0, $points[2]=$x1, $points[3]=$y1, etc...), $num_points est le nombre de points dans le tableau et $col la couleur.
ImageFilledPolygon ($im, $points, $num_points, $col)	Toutes	Idem à ImagePolygon mais le polygone est rempli par la couleur $col
ImageRectangle ($im, $x1, $y1, $x2, $y2, $col)	Toutes	Trace dans $im un rectangle de coin supérieur gauche ($x0, $y0) et de coin inférieur droit ($x1, $y1) de couleur $col
ImageFilledRectangle ($im, $x1, $y1, $x2, $y2, $col)	Toutes	Idem à ImageRectangle mais le rectangle est rempli par la couleur $col
ImageSetStyle ($im, $style) ImageSetBrush ($im, $brush)	2.0.2	Ces fonctions permettent de changer le style de tracé des lignes, consultez la documentation officielle et surtout l'exemple pour la fonction ImageSetStyle qui montre également comment utiliser ImageSetBrush
ImageSetPixel ($im, $x, $y, $col)	Toutes	Dessine dans $im un point de coordonnées ($x,$y) et de couleur $col
ImageSetThickness ($im, $epaisseur)	Toutes	Change l'epaisseur des lignes tracées par toutes les fonctions.
ImageString ($im, $police, $x, $y, $chaine, $col)	Toutes	Dessine dans $im la chaîne de caractère $chaine à partir du point ($x,$y) avec la couleur $col et avec la taille de police de caractère $police (compris entre 0 et 5)
ImageStringUp ($im, $police, $x, $y, $chaine, $col)	Toutes	Idem à ImageString mais la chaîne de caractères est dessinée verticalement

III) Un premier exemple : graphique des visiteurs

Pour appliquer ces quelques notions, voici un premier exemple assez simple. On desire pouvoir générer un histogramme (graphique sous forme de batons) représentant le nombre de visites sur votre site web sur les 12 mois de l'année. On supposera que vous avez déjà réalisé la **partie comptabilisation des visites** (dans une table **MySQL** par exemple). Imaginons donc que le nombre de visites pour chacun des douze mois est placé dans le tableau **$visites**. Ici ce tableau sera rempli à la main, mais ce serait à vous de le remplir en fonction de votre table **MySQL**.

a) Les visites, l'image et les couleurs

On commence donc par créer un tableau contenant les visites sur 12 mois (donc 12 valeurs). On crée ensuite une **image 400x300** et on crée trois couleurs, dont la première (le blanc) sera la couleur du fond.

visites.php

```
<?php
   $visites = array(138, 254, 381, 652, 896, 720, 140, 556, 663, 331, 407, 768);
   header ("Content-type: image/png");
   $largeurImage = 400;
   $hauteurImage = 300;
   $im = ImageCreate ($largeurImage, $hauteurImage)
         or die ("Erreur lors de la création de l'image");
   $blanc = ImageColorAllocate ($im, 255, 255, 255);
   $noir = ImageColorAllocate ($im, 0, 0, 0);
   $bleu = ImageColorAllocate ($im, 0, 0, 255);
```

b) On dessine les axes

On place l'axe vertical du temps (un simple trait noir) en bas de l'image, en laissant une marge de 10 points. On écrit en dessous le numéro de chacun des mois en utilisant une simple **boucle for** et la **fonction ImageString**. Les numeros de mois sont placés tous les 30 points en partant de la gauche de l'image. Enfin on trace un trait vertical représentant l'axe vertical du nombre de visites.

```
// on dessine un trait vertical pour représenter l'axe du temps
   ImageLine ($im, 10, $hauteurImage-10, $largeurImage-10, $hauteurImage-
10, $noir);
   // on affiche le numéro des 12 mois
   for ($mois=1; $mois<=12; $mois++) {
      ImageString ($im, 0, $mois*30, $hauteurImage-10, $mois, $noir);
   }
   // on dessine un trait vertical pour représenter le nombre de visites
   ImageLine ($im, 10, 10, 10, $hauteurImage-10, $noir);
```

c) On dessine les batons

Voici la partie un peu plus difficile au cours de laquelle on dessine les batons.Tout d'abord, il nous faut le nombre de visites maximal que nous aurons à tracer sur le graphe. Ici, la valeur est posée arbitrairement (1000), mais en réalité il faudrait la calculer en parcourant le tableau des visites à la recherche de son maximum, mais faisons simple. Pour chacun des mois, il faut calculer la hauteur du rectangle que l'on veut tracer. Pour cela on fait un produit en croix sachant que la hauteur du rectangle pour le nombre de visites maximum serait égale (pour simplifier) à la hauteur de l'image. A partir du nombre de visites pour un mois donné on en déduit la hauteur voulue. Il ne reste plus qu'à le tracer chaque rectangle, en commençant par son point supérieur gauche et en finissant par son point inférieur droit. Pour améliorer les choses, on ajoute le nombre de visites au dessus de chaque rectangle.

```
// le nombre maximum de visites
$visitesMax = 1000;
// tracé des batons
```

```
for ($mois=1; $mois<=12; $mois++) {
$hauteurImageRectangle = round(($visites[$mois-1]*$hauteurImage)/$visitesMax);
ImageFilledRectangle ($im, $mois*30-7, $hauteurImage-
$hauteurImageRectangle, $mois*30+7, $hauteurImage-10, $bleu);
ImageString ($im, 0, $mois*30-7, $hauteurImage-$hauteurImageRectangle-
10, $visites[$mois-1], $noir);
 }
 // et c'est fini...
ImagePng ($im);
?>
```

IV) D'autres fonctions utiles

Voici une autre série de fonctions utiles, orientée cette fois vers la **gestion des couleurs**, et diverses informations sur les images.

Syntaxe avec lien vers la doc.	Version	Description
$tableau = GetImageSize ($fichier)	Toutes	A partir d'une image dans le fichier $fichier, cette fonction renvoie un tableau contenant les 4 éléments suivants: ▪ $tableau[0] est la largeur de l'image ▪ $tableau[1] est la hauteur de l'image ▪ $tableau[2] donne le type de l'image (consultez la documentation pour la liste) ▪ $tableau[3] donne une chaîne de caractère utilisable directement dans le tag HTML <IMG>
ImageColorTransparent ($im, $col)	Toutes	Déclare la couleur $col de l'image $im comme couleur transparente. Tous les éléments tracés avec la couleur $col seront transparents. Pour garder la propriété de transparence vous devez utiliser le format PNG.
ImageCopy ($dst_im, $src_im, $dst_x, $dst_y, $src_x, $src_y, $src_l, $src_h)	Toutes	Copie une partie rectangulaire de l'image $src_im dans l'image $dst_im. La partie à copier est délimitée par le point ($src_x, $src_y), la largeur $src_l et la hauteur $src_h. La copie est placée dans $dst_im à partir du point ($dst_x, $dst_y)
ImageCopyMerge ($dst_im, $src_im, $dst_x, $dst_y,	Toutes	Idem à ImageCopy mais en effectuant un fondu d'intensité $intensite (entre 0 et 100)

$src_x, $src_y, $src_l, $src_h, $intensite)		entre l'image à copier et le fond actuel.
ImageCopyResampled	2.0.2	Fonctionne de la même façon que **ImageCopy** mais en plus effectue un redimensionnement. Cette fonction est expliquée dans l'exemple qui suit.
ImageCopyResized	Toutes	Idem à ImageCopyResampled mais le résultat du redimensionnement est moins propre mais plus rapide à effectuer.
$im = ImageCreateFromJpeg ($fichier)	Toutes	Crée une image $im contenant l'image JPEG du fichier $fichier.
$im = ImageCreateFromPng ($fichier)	Toutes	Crée une image $im contenant l'image PNG du fichier $fichier.
$largeur = ImageSX ($im)	Toutes	Renvoie la largeur de l'image $im
$hauteur = ImageSY ($im)	Toutes	Renvoie la hauteur de l'image $im

V) Deuxième exemple: générer des miniatures

Dans cet exemple l'objectif est de créer la miniature d'une image existante. Vous pourriez avoir à programmer cette fonctionnalité pour une galerie de photos en **PHP**. Les photos de la galerie sont présentées sous la forme de vignettes sur lesquelles le visiteur peut cliquer pour agrandir. Ici nous allons donc générer une miniature avec un cadre et quelques informations comme le nom de l'image d'origine et ses dimensions.

a) On prépare le tout.

A la différence du premier exemple, ce script ne va pas générer directement une image, celle-ci sera écrite dans un fichier. Il n'est donc plus utile de spécifier un **header(),** le script redevient classique et peut générer du html. On commence donc par créer une image vide de dimensions 200x150, on lit l'image existante avec la fonction **ImageCreateFromJpeg**. Pour la suite nous aurons également besoin des dimensions de l'image source, on utilise donc les fonctions **imagesx et imagesy**.

miniature.php

```
<?php
   $fichierSource = "photo3.jpg";
   $largeurDestination = 200;
   $hauteurDestination = 150;
   $im = ImageCreateTrueColor ($largeurDestination, $hauteurDestination)
   or die ("Erreur lors de la création de l'image");
   $source = ImageCreateFromJpeg($fichierSource);
   $largeurSource = imagesx($source);
   $hauteurSource = imagesy($source);
```

b) Le cadre

Après avoir créé quelques couleurs, nous tracons le cadre. Pour faire simple, il s'agit d'un dégradé de gris concentrique. Pour le réaliser, une méthode simple consiste à tracer plusieurs **rectangles imbriqués** et de couleur différente. On commence par le plus grand (de la même taille que l'image) de couleur foncée. On superpose sur celui-ci un autre rectangle plus clair et légèrement plus petit, et ainsi de suite. L'épaisseur du cadre réalisé ici est de 8 points.

```
$blanc = ImageColorAllocate ($im, 255, 255, 255);
$gris[0] = ImageColorAllocate ($im, 90, 90, 90);
$gris[1] = ImageColorAllocate ($im, 110, 110, 110);
$gris[2] = ImageColorAllocate ($im, 130, 130, 130);
$gris[3] = ImageColorAllocate ($im, 150, 150, 150);
$gris[4] = ImageColorAllocate ($im, 170, 170, 170);
$gris[5] = ImageColorAllocate ($im, 190, 190, 190);
$gris[6] = ImageColorAllocate ($im, 210, 210, 210);
$gris[7] = ImageColorAllocate ($im, 230, 230, 230);
for ($i=0; $i<=7; $i++) {
ImageFilledRectangle ($im, $i, $i, $largeurDestination-$i, $hauteurDestination-$i, $gris[$i]);
}
```

La miniature

Pour créer la miniature on utilise la fonction **ImageCopyResampled**. Cette fonction copie une image dans une autre en procédant à un redimensionnement. C'est en quelque sorte un **copier-coller** intelligent. Cette fonction prend de nombreux paramètres, mais ils sont évidents. Dans l'ordre:

- L'image de destination (**$im**)
- L'image source (**$source**)
- La position sur l'axe horizontal de l'image de destination à partir de laquelle la copie sera placée (8 pour ne pas empieter sur le cadre)
- Idem sur l'axe vertical (8)
- La position sur l'axe horizontal dans l'image source (0 car on veut copier l'intégralité de l'image source)
- Idem sur l'axe vertical (0)
- La largeur de la copie dans l'image destination **($largeurDestination-(2*8)** la largeur de l'image destination moins la largeur du cadre à gauche et à droite)
- Idem pour la hauteur
- La largeur de la partie à copier depuis l'image source ($largeurSource car on veut copier l'intégralité de l'image source)
- Idem avec la hauteur

Ceci fait, on en profite pour ajouter un petit texte d'information sur l'image.

Attention: La fonction **ImageCopyResampled** est assez lourde au niveau du temps d'éxécution sur le serveur.

```
ImageCopyResampled($im, $source, 8, 8, 0, 0, $largeurDestination-(2*8), $hauteurDestination-(2*8), $largeurSource, $hauteurSource);
```

```
ImageString($im, 0, 12, $hauteurDestination-18, "$fichierSource -
($largeurSource x $hauteurSource)", $blanc);
```

d) Sauvegarde du résultat

Il ne reste plus qu'à sauvergarder le résultat dans un fichier commençant par mini_ grâce à la fonction **ImageJpeg**. Lorsque cette fonction n'a qu'un paramètre, elle envoie l'image au navigateur du visiteur. Par contre, si en deuxième paramètre elle a un nom de fichier, rien ne sera envoyé au navigateur, mais l'image sera sauvegardée. Ceci est également vrai pour **ImagePng**.

```
   $miniature = "mini_$fichierSource";
   ImageJpeg ($im, $miniature);
   echo "Image miniature générée: $miniature";
?>
```

VI) FAQ et conclusion

La **librairie GD** laisse de nombreuses possibilités mais qui sont parfois un peu trop gourmandes en temps d'éxécution pour la grande majorité des hébergeurs. Par ailleurs de nombreux scripts existent et permettent de réaliser certaines fonctions encore plus facilement.

Voici une petite série de questions-réponses sur les problèmes courants:

Mon image n'apparaît pas, le navigateur se comporte comme si l'image était inexistante, pourquoi ? Vous avez très certainement une erreur dans votre script. Mettez en commentaire la ligne **header("Content-type: image/png")** (ou **header("Content-type: image/jpeg"))**et la ligne **ImagePng($im) (ou ImageJpeg($im)),** pour laisser apparaitre l'erreur.

Le script met beaucoup de temps à s'exécuter chez mon hébergeur, est-ce normal ? Si vous utilisez des fonctions lourdes comme **ImageCopyResampled**, ce n'est pas étonnant. **Le serveur PHP** de votre hébergeur a surement déjà beaucoup de travail.

Mon image apparait en noir et blanc ou les couleurs sont fausées, pourquoi ?
Vous avez certainement utilisé la fonction **ImageCreate** qui limite le nombre de couleurs à 256 puisqu'il s'agit d'une image indexée (palette de couleurs). Pour résoudre cette limitation, utilisez la fonction **ImageCreateTrueColor** à la place de **ImageCreate.**
Conseils: avec **ImageCreate**, utilisez plutot le **format PNG** (meilleure qualité et petite taille). Avec **ImageCreateTrueColor** utilisez plutot le **format JPEG** pour diminuer le poids de l'image.

22)Les erreurs

Les messages d'erreur.

Première chose importante, il faut comprendre le message d'erreur correctement et ne pas tenter une interprétation personnelle de l'erreur. Partez du principe que **PHP** et **Mysql** ne se

trompent pas. Pour vous aider, voici une liste des erreurs les plus fréquemment rencontrées. Bien sûr, cette liste ne contient pas tous les messages d'erreur.

Message	Signification
Parse error: parse error in xxxx.php on line y	Typique d'une erreur de syntaxe. Vérifiez si vous n'avez pas oublié un ; (point virgule) en fin de ligne. Ou encore un $ (dollar) devant le nom d'une variable. N'hésitez pas à contrôler les lignes avoisinantes. L'erreur se trouve souvent juste au-dessus.
Warning: Oops, php_SetCookie called after header has been sent in xxxx.php on line y	Vous avez tenté d'initialiser un cookie après que l'entête HTTP soit envoyé au client. Vérifiez si une sortie (echo, print, message d'erreur, ligne blanche avant les tags php) ne se fait quelques temps avant.
Warning: MySQL Connection Failed: Access denied for user:	Erreur de connexion à la base **MySQL**. Vérifiez si **host, user** et **password** sont corrects.
Warning: Unable to create [chemin] No such file or directory in your script on line [numero]	Le chemin vers le répertoire censé contenir le fichier ou bien le chemin du répertoire dans lequel le fichier doit être créé n'est pas bon (n'existe pas)
Warning: 0 is not a MySQL result index in xxxx.php on line y	Erreur probable au niveau de la requête SQL. Vérifiez votre requête SQL, en particulier les champs manipulés, le nom de ou des tables impliquées, etc. Un petit truc : il est souvent pratique de stocker ses requêtes dans une chaîne.
Warning: Variable $zzzz is not an array or string in xxxx.php on line y	Vous tentez de manipuler une valeur numérique avec une fonction dédiée aux chaînes ou aux tableaux.
Warning: Variable $zzzz is not an array or object in xxxx.php on line y	Vous tentez de manipuler une valeur numérique avec une fonction dédiée aux tableaux ou aux objets.
Warning: Cannot add header information - headers already sent by (output started at **.php:21) in ***.php on line 54**	Vous avez tenté d'effectuer un header après que **l'entête http** ait été envoyé au client. Le message est clair : Dans notre exemple votre header est en ligne 54 et l'envoi a commencé ligne 21. Vérifiez si une sortie (echo, print,

	message d'erreur) ne se fait dans les lignes 21 et suivantes.
Fatal error: Maximum execution time exceeded in xxxx.php on line y	**PHP** dispose d'un mécanisme permettant de se prémunir des scripts susceptibles d'engendrer un temps d'exécution trop important pouvant saturer un serveur. Par défaut, ce temps est de 30 secondes.
Fatal error: Allowed memory size of 8388608 bytes exhausted (tried to allocate x bytes) in yyyy.php on line z	**PHP** dispose d'un mécanisme permettant de se prémunir des scripts susceptibles d'engendrer une consommation mémoire trop importante pouvant saturer un serveur. Par défaut, une limite est fixée à environ 8 Mo (8388608 octets).
Fatal Error: Call to undefined function: xxxx() in yyy.php on line z	La fonction que vous appelez n'existe pas. Ce peut-être une fonction liée à une librairie externe (**GD, Zlib, PDF, etc**.). Dans ce cas, un simple **phpinfo()** vous renseignera sur les paramètres de compilation de votre version de PHP. Peut-être s'agit-il sinon d'une de vos propres fonctions. Vérifiez alors qu'elle existe. Et dans tous les cas, contrôlez de plus près le nom de la fonction appelée (orthographe, etc.). Une erreur de frappe est vite arrivée.
Fatal Error: Cannot redeclare xxxx() in yyy.php on line z	Vous avez certainement déclaré plusieurs fois la même fonction. Contrôlez à nouveau l'ensemble des fonctions que vous avez créées. Et n'hésitez pas à vérifier également dans les éventuels fichiers inclus. C'est souvent dans un script secondaire que vous trouverez le doublon. Veillez aussi à ne pas utiliser le nom d'une fonction propre à **PHP** ou à l'une de ses librairies.
Fatal error: Input in flex scanner failed in xxxx on line y	Vérifiez vos include et require. Il y a fort à croire que vous avez indiqué un chemin incomplet (genre /usr/local/ sans préciser de fichier).
Fatal error: Input in flex scanner failed in xxxx on line y	Vérifiez vos include et require. Il y a fort à croire que vous avez indiqué un chemin incomplet (genre /usr/local/ sans préciser de fichier).
Failed opening '%s' for	Le fichier n'a pas pu être inclus dans

inclusion (include_path='%s')	votre script, car **PHP** n'a pas pu y accéder : vérifiez les droits (utilisateur PHP, droits du fichier), les noms et chemins du fichier inclus.
file("%s") - Bad file descriptor	Problème d'accès à un fichier avec la fonction **file()**. Vérifiez bien que l'url est valide. (l'url "http://www.super.php") est invalide alors qu'une erreur de type 404 sera valide.
Wrong parameter count for %s()	La fonction est appelée avec un nombre insuffisant de paramètres, ou bien avec trop de paramètres. Certaines fonctions ont besoin d'un minimum de paramètres (array()), et généralement d'un maximum.
stat failed for %s (errno=%d - %s)	Impossible d'accéder au fichier (problème de droits ou de chemin d'accès).

A noter que si vous avez une parse error à la ligne 30, l'erreur peut se trouver à la ligne de code précédente (par exemple il peut manquer un ; à la fin de la ligne). Vous pouvez aussi obtenir une **parse error** bizarre en fin de page. Vérifiez bien si vous avez fermé toutes vos accolades. Vous devez avoir autant d'accolades ouvrantes **{que d'accolades fermantes}.** Sinon php patiente jusqu'à la fin du script et signale l'erreur en dernière ligne.

Problèmes de variables

Parfois, vous pouvez avoir certains problèmes à cause de variables extérieures au script, elles ne semblent pas passer. D'abord assurez vous de ce qui passe vraiment, en imprimant vos variables en début de script, vous verrez ainsi d'un coup d'oeil, si certaines (ou toutes les) variables sont vides. N'hésitez pas non plus à faire un echo ou un print chaque fois que vous avez manipulé une variable. Vous l'enlèverez après. Vérifiez bien que vous récupérez les variables par les bons tableaux (d'après la **version de PHP** sur le serveur).

Vous pouvez utiliser **var_dump()** ou **print_r()** pour afficher d'un coup le contenu des tableaux.

Problèmes avec mysql

Ca ne sert à rien de tourner en rond et de jurer qu'une DB ou une table existe, si php vous dit le contraire. Avec **Mysql** il y a quelques bonnes habitudes à prendre.

die()

Cette fonction permet d'afficher un message et d'arrêter le script. Vous devriez toujours l'utiliser.

```
mysql_connect('localhost','root','')
or  die('Problème de connection à mysql');
```

En cas d'erreur, le **script s'arrêtera** et affichera **Problème de connection à mysql**, de cette façon vous saurez exactement ou se trouve le problème dans le script.

```
mysql_connect('localhost','root','')
or  die('Problème de connection à mysql');
$link = mysql_select_db('tes')
or die('Problème de DB');
```

Construire votre requête dans une variable

Prenez l'habitude de construire votre requête dans une variable à part, afin de pouvoir afficher votre requête et débogger facilement. En effet votre requête sera affichée avec le contenu de vos variables, ce qui vous permettra de repérer par exemple une variable qui n'est pas passée, une variable avec un ' non échappé etc.

```
$select = "select nom, prenom from aches where race='$race' ";
$result = mysql_query($select,$link)
 or die('requete =>'.$select.'<br>');
```

Utiliser mysql_error()

Cette fonction retournera une **erreur mysql** sous la forme d'un texte, vous pouvez l'utiliser avec la fonction **die()**

```
$select = "select nom, prenom from aches where race='$race' ";
$result = mysql_query($select,$link)
or die('requete =>'.$select.'<br>
error->'.mysql_error());
```

Résultat

```
requete =>"select nom, prenom from aches where race='sallers' ";
error->Table 'test.aches' doesn't exist
```

Eviter l'affichage si pas de résultat

Si vous tentez d'afficher le résultat d'un select, alors que celui ci ne retourne pas de résultat, vous serez confronté à un message d'erreur. Il vaut mieux, donc, prendre l'habitude de vérifier si il y à un résultat à afficher. Pour cela vous pouvez utiliser la fonction **mysql_num_rows()**

```
$select = "select nom, prenom from vaches where race='$race' ";
$result = mysql_query($select,$link)
or die('requete =>'.$select.'<br> error->'.mysql_error());
$compte = mysql_num_rows($result);
if($compte>0) {
```

```
while($row = mysql_fetch_row($result)) {
 echo $row[0].'<br>';
 }
}
else {
 echo 'aucun résultat';
}
```

Instruction conditionnelles

if...else

Votre script n'a pas le comportement attendu, il passe dans un **if** alors qu'il ne devrait pas, ou il passe dans un **else** alors que vous vous attendiez à ce qu'il passe dans le **if**.

Alors, affichez les variables utilisées dans le test avant le if afin d'être certain qu'elles ont les valeurs attendues.

Etes-vous sûr d'avoir utilisé **l'opérateur de comparaison (==)** et non l'opérateur d'assignation **(=)** ?

Les espaces dans les chaînes peuvent être une source de perte de temps également. Imaginez que vous compariez deux variables de type string dans votre if, mais que l'une d'elle, pour l'une ou l'autre raison, ait un espace en début ou en fin de chaîne. Ce qui la rend non égale à l'autre chaîne.

Afin de vérifier ce genre de petites erreurs bêtes, faites un écho de vos deux variables et entourez-les avec des *, de cette façon, du premier coup d'oeil, vous verrez si un espace s'est glissé dans votre chaîne.

```
$var1 = 'lala';
$var2 = 'lala ';
echo '*'.$var1.'*<br>';
echo '*'.$var2.'*<br>';
if($var1 == $var2) {
 echo 'identiques';
}
else {
echo 'pas identiques';
}
```

Les boucles:

Mon script devient fou et ne s'arrête pas. Vous avez sans doute un problème avec la condition d'arrêt de la boucle (elle n'est jamais atteinte). Dans le cas d'une **boucle while**, êtes-vous certain de ne pas avoir oublié l'itération de votre variable ?

Pour voir le nombre de passages dans la boucle, vous pouvez ajouter un echo '.'; , cela vous permettra de voir combien de fois la boucle est exécutée et voir si c'est normal. Rien ne vous empêche d'utiliser une **variable $debug** servant d'interrupteur au debug. Exemple:

```
$i = 0;
$debug = 1;
while($i < 100) {
if($debug) {
        echo '.';
}
$i = $i + 1;
}
```

Si vous passez **$debug à 0**, les **echo de debug** ne seront pas affichés, si vous avez besoin d'afficher des variables afin d'en contrôler la valeur, vous n'avez plus qu'à réactiver votre interrupteur.

Les fichiers :

Une erreur d'**include,** de **require** ou de **fopen** peut avoir plusieurs causes.

- Le chemin vers le fichier n'est pas correct.
- Les permissions sur le fichier ne sont pas suffisantes.
- Certaines fonctions d'information sur les fichiers ne fonctionnent pas sous **windows** (ce qui est en général est indiqué dans la doc).

La fonction mail()

Pour **envoyer un mail**, vous avez besoin ou bien :

- d'avoir un serveur **SMTP** sur le serveur qui héberge votre script
- ou bien de configurer votre **php.ini** avec un serveur **SMTP** que vous pouvez utiliser (celui de votre **FAI** par exemple)
- Et surtout d'avoir configuré le **php.ini** en conséquence

```
[mail function]
; pour windows
SMTP = smtp.monfai.fr

; pour windows
sendmail_from = test@moi.fr

; pour unix.
;sendmail_path =
```

Si vous obtenez cette erreur en utilisant la fonction **mail(),** rappellez vous de mon histoire de vaches volantes et vérifiez la **configuration de php**. N'oubliez pas de redémarrer **Apache** pour que les infos soient prises en compte.

Warning: Failed to Connect in [chemin] on line [nombre]

Pas à pas

Rien ne fonctionne comme vous voulez. Debuggez votre script partie par partie, prenez le **temps de tracker vos variables** en les affichant avant et après certains traitements.

Conclusion

Avec un peu d'expérience et de patience, vous allez développer des automatismes de deboggage. Et bientôt ces **parses error** qui vous faisaient perdre un temps fou n'auront plus de secret pour vous.

23)Variables globales à OFF

Travailler avec les variables globales à OFF

Depuis peu, peut être avez vous rencontré certains problèmes avec les versions récentes de **PHP** pour la manipulation de variables. Avant, pour récupérer vos variables, vous pouviez le faire directement par leur nom, quel que soit le type de variables (passées par la méthode **POST/GET, cookie, session**, etc.)
Cela nous permettait de faire ceci :

Récupération directe de variables
un appel au script :
http://www.mondomaine.fr/monscript.php
?truc=coucou

```
<?php
 echo 'Ma variable passée par url =>'.$truc;
?>
```

affichant :
Ma variable passée par url =>coucou

La variable **$truc**, passée par **URL** est directement accessible depuis le **script PHP.** Cette façon de travailler a l'air, certes, bien pratique, elle laisse néanmoins la porte ouverte à des trous de sécurité. Ce qui nous permettait d'utiliser nos variables de cette façon-là est en fait une option se trouvant dans le fichier **php.ini** qui s'appelle : **register_globals** et qui est par défaut initialisée à **ON** dans ce fichier. Depuis la **version 4.2.0 de PHP**, cette option est par défaut initialisée à **OFF**.

C'est pourquoi il est important de prendre dès maintenant l'habitude de travailler avec **register_globals à OFF**. De plus, beaucoup d'hébergeurs ayant upgradé PHP avec la nouvelle version laissent **register_globals à OFF**. Et puis, cette méthode permet de résoudre pas mal de trous de sécurité dans les scripts.

Une autre option importante de la **configuration PHP** : **track_vars**, positionnée à **ON**, celle ci nous permet de récupérer les variables passées par formulaire, url et cookies dans des tableaux prédéfinis. Avec ces deux options configurées telles que décrites ci-dessus, nous pouvons utiliser les tableaux associatifs suivants pour récupérer nos variables :

Tableaux associatifs	
$HTTP_GET_VARS (désormais $_GET)	**Permet de récupérer les variables passées par url ou par la méthode GET d'un formulaire**
$HTTP_POST_VARS (désormais $_POST)	**Permet de récupérer les variables envoyées par la méthode POST d'un formulaire**
$HTTP_POST_FILES (désormais $_FILES)	**Permet de récupérer les variables de fichiers envoyés par un formulaire**
$HTTP_COOKIE_VARS (désormais $_COOKIE)	**Permet de récupérer les cookies**
$HTTP_SESSION_VARS (désormais $_SESSION)	**Permet de récupérer les variables de session**
$HTTP_ENV_VARS (désormais $_ENV)	**Permet de récupérer les variables d'environnement données par PHP**
$HTTP_SERVER_VARS (désormais $_SERVER)	**Permet de récupérer les variables serveur envoyées par le serveur HTTP**

A partir de **PHP 4.1.0**, ces variables changent de nom pour prendre des noms plus courts et plus faciles à retenir.

Avant 4.1.0	A partir de 4.1.0
$HTTP_GET_VARS	**$_GET**
$HTTP_POST_VARS	**$_POST**
$HTTP_POST_FILES	**$_FILES**
$HTTP_COOKIE_VARS	**$_COOKIE**
$HTTP_SESSION_VARS	**$_SESSION**
$HTTP_ENV_VARS	**$_ENV**
$HTTP_SERVER_VARS	**$_SERVER**

Mise en pratique

Nous allons voir quelques petits exemples dans lesquels nous allons manipuler ces tableaux magiques.

1. Formulaires

Récupération d'informations provenant d'un formulaire utilisant la méthode POST.

Récupération d'un formulaire exemple1.php

```
<?php
$bouton = $_POST['send'];
if(!empty($bouton)) {
 $nom = trim($_POST['nom']);
 $prenom = trim($_POST['prenom']);
 if(!empty($nom) && !empty($prenom)) {
 echo 'Bonjour, '.$prenom.' '.$nom;
 }
 else {
 echo 'vous n\'avez pas rempli tous
 les champs';
 }
}
?>
```

Formulaire utilisé form.html

```
<form action="exemple1.php" method="post">
<table>
<tr>
<td>Nom :</td>
<td>
<input type="text" name="nom"
value="">
</td>
</tr>
<tr>
<td>Prénom :</td>
<td>
<input type="text" name="prenom"
value="">
</td>
</tr>
</table>
<input type="submit" value="envoyer"
name="send">
</form>
```

Comme vous le voyez, nous récupérons la valeur des variables grâce au **tableau associatif $_POST** dans lequel une entrée à été créée avec le nom de chaque variable du formulaire. Pour ceux qui se demandent à quoi sert la **fonction trim(),** elle retire les espaces en début et fin de chaîne, cela nous permet de ne pas traiter des informations vides (un utilisateur pourrait très bien n'entrer que des espaces dans le formulaire)

Récupération d'un fichier.

Bien, maintenant voyons comment récupérer les informations concernant l'envoi de fichiers par un formulaire. Voici comment récupérer toutes les informations du fichier envoyé avec le tableau associatif **$_FILES.**

Entrée	Explications
$_FILES['variable']['name']	Le nom original du fichier qui provient de la machine de l'utilisateur
$_FILES['variable']['type']	Le type mime du fichier
$_FILES['variable']['size']	Le taille du fichier en bytes
$_FILES['variable']['tmp_name']	Le nom temporaire du fichier stocké sur le serveur
$_FILES['variable']['error']	le code erreur associé à l'upload (attention cette option à été ajoutée en PHP 4.2.0)

Cet exemple ne fait qu'afficher les **informations** concernant le fichier que l'on a uploadé.

Récupération de fichiers

```
<?php
$bouton = $_POST['bouton'];
if(!empty($bouton)) {
$fichier = $_FILES['fichier']['name'];
$size = $_FILES['fichier']['size'];
$tmp = $_FILES['fichier']['tmp_name'];
$type = $_FILES['fichier']['type'];
$error = $_FILES['fichier']['error'];
$max = 10000;
```

Récupération de fichiers

```
if(isset($fichier)) {
 if($size <= $max) {
 echo 'Nom d\'origine =>'.$fichier.'<br />';
 echo 'Taille =>'.$size.'<br />';
 echo 'Nom sur le serveur =>'.$tmp.'<br />';
 echo 'Type de fichier =>'.$type.'<br />';
 echo 'Code erreur =>'.$error.'<br />';
 }
 else {
 echo 'Le fichier est trop volumineux';
 }
}
else {
 echo 'aucun fichier envoyé';
}
}
?>
<form enctype="multipart/form-data"
action="exemple2.php"
method="post">
```

```
<input name="fichier" type="file">
<input type="submit" value="send" name="bouton">
</form>
```

Récupérer les informations d'un formulaire facilement.

Imaginons que vous vouliez simplement afficher les informations entrées dans un formulaire. Vous pouvez parcourir le **tableau $_POST** afin d'afficher tout ce qu'il contient.

La **variable $key** contient le nom de la variable formulaire et la **variable $val** contient la valeur entrée dans le formulaire.

Afficher toutes les infos d'un formulaire

```
<?php
if(isset($_POST)) {
foreach($_POST as $key=>$val) {
echo $key.'=>'.$val.'<p>';
 }
}
else {
 echo 'le formulaire n\'a pas été envoyé';
}
?>
<form action="exemple3.php" method="post">
Nom :<input type="text" name="nom"
value="">
<br />
Prénom:<input type="text" name="prenom"
value="">
<br />
Adresse:<input type="text" name="adresse"
value="">
<br />
Ville:<input type="text" name="ville"
value="">
<br />
Pays:<input type="text" name="pays"
value="">
<br />
<input type="submit" value="envoyer">
</form>
```

2. La méthode get et le passage par url

Rien de bien compliqué, il suffit de récupérer les **variables passées en get** par le tableau **$_GET**.Un petit exemple pour illustrer quand même.

Passage par url

```
http://www.monsite.fr/exemple4.php?nom=cow&prenom=flying
<?php
```

```
$nom = $_GET['nom'];
$prenom = $_GET['prenom'];
echo 'Bonjour, '.$prenom.' '.$nom;
?>
```

3. Les cookies

Récupération d'un cookie

```
<?php
$cookie = $_COOKIE['moncookie'];
if(!empty($cookie)){
 echo 'Valeur du cookie=>'.$cookie;
}
?>
```

4. Les variables session

En récupérant vos variables sessions par le **tableau associatif $_SESSION** vous êtes sûrs que la valeur est bien une variable session. Voyons un exemple ou nous récupérons une variable session directement. Le fichier **session.php**, débute une session s'il n'en existe pas encore pour l'utilisateur. Fichier ou l'on crée une entrée en session si le login et mot de passe entrés sont corrects, auquel cas, on initialise la variable **$_SESSION['login'],** avec la chaîne **ok**.

Variables session

```
<?php
include 'session.php';
$bouton = $_POST['bouton'];
if(!empty($bouton)) {
$log = trim($_POST['log']);
$pass = trim($_POST['pass']);
 if ($log=='jenny' && $pass == 'foo') {
 $_SESSION['login'] = 'ok';
 echo '<a href="exemple7.php"> >> suite >></a>';
 }
 else {
 echo 'mauvais login/pass';
 }
}
?>
<form action="exemple6.php" method="post">
Login :<input type="text" name="log"
value=""><br />
Pass :<input type="text" name="pass"
value=""><br />
<input type="submit" value="envoyer"
name="bouton">
</form>
```

Voici comment récupérer les variables session par le bon tableau.

Récupération d'une session

```
<?php
include 'session.php';
if($_SESSION['login'] == 'ok') {
 echo 'vous êtes bien connecté';
}
else {
 die('pas en session');
}
?>
```

5. Les variables d'environnement

Utilisation du **tableau $_ENV** pour récupérer les variables d'environnement : en voici quelques unes d'utiles.

Les variables d'environnement

```
<?php
echo 'Nombre de process actifs=>';
echo $_ENV['NUMBER_OF_PROCESSORS'].'
';
echo 'Système d\'exploitation=>';
echo $_ENV['OS'].'
';
echo 'Chemin du répertoire temporaire=>';
echo $_ENV['TMP'].'
';
echo 'Chemin du profil utilisateur=>';
echo $_ENV['USERPROFILE'].'
';
?>
```

7. Les variables serveurs

Utilisation du **tableau $_SERVER** pour récupérer les variables serveur : voici quelques exemples.

Les variables serveur

```
<?php
echo 'Chemin du script courant=>';
echo $_SERVER['PHP_SELF'].'
';
echo 'Nom du serveur=>';
echo $_SERVER['SERVER_NAME'].'
';
```

```
echo 'Variables passées au script=>';
echo $_SERVER['QUERY_STRING'].'
';
echo 'Document root=>';
echo $_SERVER['DOCUMENT_ROOT'].'
';
echo 'Référant=>';
echo $_SERVER['HTTP_REFERER'].'
';
echo 'Adresse ip de l\'utilisateur=>';
echo $_SERVER['REMOTE_ADDR'].'
';
?>
```

Vous pouvez avoir la liste des **variables serveurs** et d'**environnement** en créant le **petit fichier PHP** ci-dessous et en l'exécutant.

```
<?php
phpinfo();
?>
```

7. Récupérer facilement les variables

Pour récupèrer facilement les variables, vous pouvez utiliser la **fonction PHP extract**. Elle va exporter votre tableau associatif et créer une variable pour chaque clé du tableau.

```
extract($_POST,EXTR_OVERWRITE);
```

Cette fonction va créer une variable pour chaque clé du tableau associatif **$_POST**.Si on a :

- **$_POST['nom']**
- **$_POST['prenom']**
- **$_POST['age']**

La **fonction extract()** va créer les variables suivantes :

- **$nom**
- **$prenom**
- **$age**

Le second argument sert à gérer les collisions de variables. Donc, il sert à dire ce que l'on fait si une variable existe déjà. Par défaut les variables qui existent déjà seront écrasées.

Type	signification
EXTR_OVERWRITE	Ecrase les variables existantes
EXTR_SKIP	N'écrase pas les variables existantes
EXTR_PREFIX_SAME	Si une variable existe déjà, une nouvelle variable est créée avec

	un préfixe donné en 3ème argument à la fonction
EXTR_PREFIX_ALL	**Crée de nouvelles variables avec le préfixe passé en 3ème argument pour toutes les clés du tableau**
EXTR_PREFIX_INVALID	**Crée de nouvelles variables avec le préfixe passé en 3ème argument pour les noms de variable invalides (par exemple $1)**

Voilà un petit exemple qui montre comment fonctionne la fonction extract. La **variable $nom** existant déjà, une nouvelle variable va être créée **$new_nom**.

```
<?php
$nom = 'blabla';
$_POST['nom'] = 'jenny';
extract($_POST,EXTR_PREFIX_SAME,'new');
echo 'variable "nom" =>'.$nom.'<br>';
echo 'variable "new_nom" =>'.$new_nom.'<br>';
?>
```

Résultat de cet exemple :

```
variable "nom" =>blabla
variable "new_nom" =>jenny
```

24)Les variables dynamiques

Introduction

Les **variables dynamiques** sont un aspect de **PHP** peu connu par les débutants, cependant, elles peuvent s'avérer vraiment utiles dans certaines situations.

Notre premier exemple

Pour commencer prenons un exemple, tout ce qu'il y a de plus simple, c'est juste de mieux vous préparer à la suite :

```
<?php
$variable =  'nom';
$$variable =  'phpdeb';
?>
```

Vous vous demandez certainement à quoi peut nous servir ce code, supposons que vous vouliez afficher une variable choisie par le visiteur et passée par l'**url** (soit **$_GET[]**), vous devriez utiliser des **if** pour cerner toutes les possibilités, mais en

utilisant cette astuce vous réduisez votre code à seulement quelques lignes et vous obtenu le résultat voulu tout en gardant un code propre et net.

Notre visiteur clique sur un lien par exemple et il est mené à cette url : **'http://www.poli.org/test.php?variable=nom'**, il demande donc à voir le contenu de la variable nom, voyons comment procède le code de la page test.php :

```php
<?php
// Pour cet exemple nous supposerons que PHP_VERSION => 4.1.0
$variable = $_GET['variable'];
$nom = 'phpdeb';
// affichage de la variable demandée par le visiteur
if($variable == 'nom'){
echo $$variable;
}
// ce qui affiche 'phpdeb'
?>
```

Il existe deux façons de déclarer ou d'afficher les variables dynamiques, ainsi le code suivant aurait tout à fait le même résultat à l'affichage que celui cité plus haut :

```php
<?php
// Pour cet exemple nous supposerons que PHP_VERSION => 4.1.0
$variable = $_GET['variable'];
$nom = 'phpdeb';
// affichage de la variable demandée par le visiteur
if($variable == 'nom'){
   echo ${$variable};
}
// ce qui affiche aussi 'phpdeb'.
?>
```

Il faut aussi savoir qu'il n'y a pas de limite à l'utilisation de ce type de variables, nous pourrions donc faire d'une variable dynamique une autre variable dynamique et ceci à l'infini, voyons un exemple :

```php
<?php
$variable = $_GET['variable'];
$nom = 'phpdeb';
$phpdeb = 'autre_var';
$autre_var = 'Et voilà le travail';
// affichage de la variable $autre_var
if($variable == 'nom'){
   echo $$$$variable;
}
// ce qui va afficher 'Et voilà le travail'
?>
```

Ces exemples affichent des variables en fonction de ce que contient l'url. N'oubliez pas qu'il ne faut **jamais** se fier à des données provenant de **GET (par l'url)** ou de **POST**

(par des formulaires). Il vous faut absolument contrôler le contenu de ces variables pour s'assurer qu'il n'y a pas de risque d'afficher des variables contenant des données sensibles. C'est ce qu'on a fait en ajoutant des **if** avant d'afficher. Cependant les variables dynamiques ne sont pas dangereuses en elles-mêmes, c'est le fait d'utiliser une variable provenant de **GET**, le problème est le même avec les **pseudos-frames.**

Tableaux dynamiques

Il est aussi possible d'utiliser des noms dynamiques pour les variables tableaux, le fonctionnement ne change presque pas par rapport à ce qu'on a vu en haut pour les variables (sachant qu'un **array()** est une forme de variable étendue), voici un petit exemple d'utilisation :

```php
<?php
// Déclaration des différents array()
$pizzas = array('royale','vegetarienne');
$fruits = array('fraise','orange');
$variable = 'pizzas';
$i = 0;
while ($i < count($$variable)) {
   echo $$variable[$i];
   echo '<br/>';
   $i++;
}
?>
```

Ce code ne produit pas l'effet attendu, voici un screenshot du résultat obtenu, ça n'a vraiment rien à voir avec ce qu'on voulait, et c'est là que diffère l'utilisation des array :

Modification du code :

Le déboggage de ce code n'est pas très difficile, car la première chose qu'on essaie c'est de changer **$variable** en **${$variable}** et on voit que ca marche bien, il est donc indispensable d'utiliser les accolades **{}** pour l'utilisation de tableaux dynamiques, le code devient donc :

```php
<?php
// Déclaration des différents array()
$pizzas = array('royale','vegetarienne');
$fruits = array('fraise','orange');
$variable = 'pizzas';
for ($i=0; $i < count(${$variable}); $i++) {
echo ${$variable}[$i].'<br/>';
}
?>
```

Dans ce cas, le script ne produira pas d'erreur et affichera bien le contenu du tableau :

Les fonctions dynamiques

Il est aussi possible aussi d'utiliser des **fonctions dynamiques**, c'est à dire que les fonctions sont déjà définies par exemple et l'on ne sait pas encore laquelle utiliser, pour éclaircir tout ça, rien de mieux qu'un exemple :

```
<?php
// Recuperation de l'action demandée par l'url
$action = $_GET['action'];
function ajoute($x, $y) {
   return $x+$y;
}
function soustrait($x, $y) {
   return $x-$y;
}
// exécution de la fonction
if($action == 'soustrait' or $action == 'ajoute'){
   echo $action('4', '2');
}
?>
```

Cette page **test-maths.php** si elle appelée de cette facon : **'http://localhost/test-maths.php?action=ajouter'** retournera 6; bien sûr vous n'êtes toujours pas convaincus de l'utilité de ces variables, il faudra que vous y soyez confrontés lors d'un de vos projets, c'est là que vous verrez à quelle point elles sont parfois indispensables.

Exemple d'utilisation des fonctions dynamiques :

Pour mettre en évidence l'utilité des fonctions dynamiques nous allons considérer un cas spécial. Nous disposons d'un système basique de **gestion de shopping cart**, utilisé dans un **site de e-commerce**, le client a le choix entre cliquer sur plus ou moins.
Le premier lien mène vers la page **'ajouter-qte-produit.php'** dont voici le code :

```
<?php
$fonction = $_GET['fonction'];
// on suppose qu'une connexion à mysql est déjà ouverte
function plus() {
   global $membre;
   $sql = 'UPDATE shopping SET quantite=quantite+1 WHERE
membre='.$membre;
   mysql_query($sql);
}
function moins() {
   global $membre;
   $sql = 'UPDATE shopping SET quantite=quantite-1 WHERE
membre='.$membre;
}
if ($fonction == 'plus') {
   plus();
}elseif ($fonction == 'moins') {
```

```
    moins();
}
?>
```

Imaginez que vous ayez une dizaine d'actions possibles, il vous faudrait toutes les passer par **les if,** et cela pourrait s'avérer long et surtout facilement contournable, voici ce que serait le code si on utilisait les **fonctions dynamiques**:

```
<?php
$fonction = $_GET['fonction'];
function plus() {
    global $membre;
    $sql = 'UPDATE shopping SET quantite=quantite+1 WHERE
membre='.$membre;
    mysql_query($sql);
}
function moins() {
    global $membre;
    $sql = 'UPDATE shopping SET quantite=quantite-1 WHERE
membre='.$membre;
    mysql_query($sql);
}
// on doit quand même controler si c'est une fonction que l'on autorise
// à éxécuter, sinon gare à la sécurité
if($fonction == 'plus' || $fonction == 'moins'){
    $fonction();
}
?>
```

Les interdits

Comme **toujours en PHP**, il y a quelques limitations à l'utilisation de ces variables, la plus importante est qu'on ne peut pas coller deux variables, le code suivant donnera un **parse error** :

```
<?php
$variable='pizza';
$action='acheter';
$acheter_pizza = 'miam';
// ceci est invalide
echo $action_$variable;
// il faut faire
echo ${$action.'_'.$variable};
?>
```

Partie 2 : PHP avancé

1) Qu'est-ce qu'un array ?

La **fonction array()** permet de créer des **tableaux dynamiques** qui seront exploités par vos **scripts PHP**, à la différence d'une **table MySQL**, l'**array** est temporaire et ne reste généré

que le temps du déroulement de votre script. Attention, les **tableaux dynamiques ou arrays** n'ont rien à voir avec les **tableaux HTML**, qui servent à la mise en forme de données.

Pourquoi utiliser les Arrays et comment les créer ?

C'est souvent un moyen très pratique de stocker de nombreuses informations venant de d'une base de données ou tout autres sources. Les tableaux sont aussi une méthode pour pouvoir stocker des informations souvent utlisées (balises html par exemple), et les appeller plus facilement ensuite dans le code. Pour comprendre le fonctionnement du tableau voici un exemple simple qui va nous premettre de stocker des balises **html** souvent utilisées (table,tr,td, align...) :

Code PHP

```
// ci-dessous nous stockons les balises qui servent à créer les tableaux HTML
$table = array(
'D_TABLE' => '<table border="0" cellpadding="0" cellspacing="2"
width="90%" bgcolor="yellow">\n',
'D_ROWS'  => '<tr><td>',
'F_ROWS'  => '</tr></td>\n',
'F_TABLE' => '</table>\n'
);
// cette fois nous stockons les balises d'alignement HTML
$align = array(
'D_CENTER' => '<div align=center>\n',
'F_CENTER' => '</div>\n',
'D_RIGHT'  => '<div align=right>\n',
'F_RIGHT'  => '</div>\n'
);
// enfin nous utilisons les valeurs des tableaux (Array) pour créer nos tableaux
HTML
echo $align['D_CENTER'];
echo $table['D_TABLE'];
echo $table['D_ROWS'];

echo 'Le texte dans la première cellule ...';

echo $table['F_ROWS'];
echo $table['D_ROWS'];

echo 'Le texte dans la deuxième cellule ...';

echo $table['F_ROWS'];
echo $table['F_TABLE'];
echo $align['F_CENTER'];
?>
```

Un avantage énorme des **tableaux en PHP** est vraiment la possibilité de stocker des informations une fois pour toutes et de pouvoir les appeller quand vous le souhaitez sans avoir à retaper les lignes de codes entières. Au même titres que les fonctions utilisateurs, il est judicieux de placer les tableaux pré-remplis dans un fichier de type : **fonctions.inc.php** et de l'appeller dans toutes les pages avec un **require() ou include()**. Bien-sûr ce n'est pas le cas d'un tableau que vous alimentez dynamiquement avec des informations spécifiques au script

en cours. Voici un autre exemple simple qui consiste à alimenter un tableau avec des données mais sans spécifier d'identifiant :

Code PHP

```
// cette fois nous stockons des valeurs sans les identifier
$prenom = array('didier','pierre','julien','xavier');
// Nous les affichons
echo $prenom[0].' - '.$prenom[1].' - '.$prenom[2].' - '.$prenom[3].'<br>';
/*
ce qui affiche :
didier - pierre - julien - xavier
*/
// Le même affichage avec une boucle for (cf exo 6)
// la fonction sizeof() renvoie le nombre d'éléments d'un tableau, ici 4
for($i=0;$i<sizeof($prenom);$i++) // tant que $i est inferieur au nombre d'éléments du tableau...
   {
   echo $prenom[$i].'<br>'; // on affiche l'élément du tableau d'indice $i
   }
/*
ce qui affiche :
didier
pierre
julien
xavier
*/
?>
```

Si aucun identifiant n'est spécifié, c'est le chifre **0** par défaut qui servira de base. A contrario, vous pouvez spécifier l'identifiant de base, qui peut être **1** par exemple et non pas **0**, dans ce cas les valeurs suivantes auront un identifiant égale à l'identifiant **précédent + 1** (comme un **auto_increment** dans mySQL), voici ce que cela donne:

Code PHP

```
$animaux = array(1 => 'chien','chat','vache','cochon');
// L'affichage commence cette fois à partir de : [1]
echo $animaux[1].' - '.$animaux[2].' - '.$animaux[3].' - '.$animaux[4];
?>
```

Parcourir un tableau avec la **boucle foreach**

Vous avez vu une première methode pour parcourir un tableau : il s'agit d'utiliser une boucle for. Mais seulement cela ne marche que pour un tableau dont les clés sont numériques **($animaux[1] par exemple).** Maintenant comment faire pour parcourir un tableau ayant des expressions litterales comme clés ? Pour cela il suffit d'utiliser la **boucle foreach**. Voici un exemple qui illustre cette boucle :

Code PHP

```
$individu = array(
'Nom'=>'Dupond',
'Prenom'=>'Martin',
'Age'=>18,
'Nationalité'=>'Francais',
);
```

```php
// pour parcourir ce tableau, utilisons la boucle foreach
foreach($individu as $cle=>$valeur)
   {
   echo $cle.' : '.$valeur.'<br>';
   }
 /*
Cet exemple affiche :
Nom : Dupond
Prenom : Martin
Age : 18
Nationalité : Francais
*/
?>
```

Les tableaux multi-dimensionnels

Dans l'exemple ci-dessus, à chaque tour, la boucle accède à l'élément suivant du tableau et met dans **$cle** le nom de la cle de l'élément et dans **$valeur** la valeur de l'élément. Evidemment vous pouvez choisir d'autres noms de variable. Dans notre exemple, la boucle aura fait quatre tours. Nous allons maintenant étudier les **tableaux multi-dimensionnels**. En effet vous risquez de rencontrer parfois des tableaux qui auront cette allure la : **$constructeur['Renault']['Citadine'] = 'Clio';** par exemple ou bien **$constrcteur['Citroen']['Monospace'] = 'Picasso';** dans le même style. Avant de passer à un exemple précis, voici un petit rappel sur le type d'une variable:

$fleur = 'rose';** On dit que **$fleur est de type string car 'rose' est une chaîne de caractère **$loisir[1] = 'Football';** On dit alors que $loisir est de type array, puisque c'est un tableau. Mais **$loisir[1]** est de type string car Football est une chaîne de caractères (essayez de considérer **$loisir[1]** comme une seule variable, un peu comme si s'était **$loisir1**). Si vous avez compris cela, alors vous n'aurez pas de mal à comprendre les tableaux à plusieurs dimensions. En effet, il suffit de comprendre que chaque élément du tableau peut lui même être un tableau, et ce jusqu'à l'infini. Reprenons notre premier exemple: **$constructeur** est un tableau. **$constructeur['Renault']** est aussi un tableau puisque il des clés, dont la **clé 'Citadine'** mais **$constructeur['Renault']['Citadine']** est de type string car 'Clio' est une chaîne de caractère.

Pour connaitre le type d'une variable, utlisez la fonction **gettype()**. Par exemple, **echo gettype($constructeur['Citroen']);**affichera **Array.** Sachez aussi qu'il existe d'autres types de variable : **boolean (true ou false),integer (nombre entier : 1,2,etc.) ou encore object**. Maintenant cela va se poser un problème lorsque que vous voudrez parcourir un tableau multi-dimensionnel. En effet si vous savez que le tableau n'a qu'un seul niveau **($tab['maison']** par exemple), alors vous allez utiliser une **boucle foreach**. Et puis si le tableau a deux niveaux **($tab['maison']['piece'] par exemple**) alors vous allez devoir à nouveau faire **deux boucles foreach** incluses l'une dans l'autre pour parcourir les deux niveaux du tableau. Ce qui pose problème lorsqu'on ne connaît pas le nombre de niveau d'un tableau. Pour cela il faut utiliser une **fonction dite récursive**, c'est à dire une fonction qui va être capable de parcourir tout un tableau quel que soit son nombre de niveaux. Pour rappel, **récursif signifie ce qui peut être répété un nombre infini de fois**. Voici donc un exemple de **fonction récursive** pour parcourir un tableau, utilisant le tableau que l'on a vu précédemment.

Code PHP

```php
// $constructeur contient la gamme Renault et Citroen
// on se contentera de deux constructeurs : ça suffira pour l'exemple :
$constructeur = array();
$constructeur['Renault']['Citadines'] = array('Twingo','Clio');
$constructeur['Renault']['Berlines'] = array('Megane','Laguna');
$constructeur['Renault']['Monospaces'] = array('Scenic','Espace');
$constructeur['Citroen']['Citadines'] = 'Saxo';
$constructeur['Citroen']['Berlines'] = array('Xsara','Xantia');
$constructeur['Citroen']['Monospaces'] = array('Picasso','Evasion');
function afficher_tableau($tableau)
   {
   // on fait une boucle qui lit les éléments du tableau
   foreach ($tableau as $cle=>$valeur)
      {
      // si l'un des éléments est lui même un tableau
      // alors on applique la fonction à ce tableau
      if(is_array($valeur))
         {
         // on affiche le nom de la clé et
         // le début d'une liste pour
         //  décaler le contenu vers la droite
         echo $cle.' : <ul>';
         // ici se réalise la récursivité
         // c'est à dire qu'on applique la fonction
         // à l'élément en cours car c'est lui aussi un tableau
         afficher_tableau($valeur);
         // on ferme la liste
         echo '</ul>';
         }
       // si ce n'est pas un tableau
      // alors on affiche le contenu de l'élément
      else
         {
         echo $cle.' = '.$valeur.' <br>';
         }
      }
   }
 afficher_tableau($constructeur);
/*
ce qui va afficher :
Renault :
   Citadines :
      0 = Twingo
      1 = Clio

   Berlines :
      0 = Megane
      1 = Laguna
```

```
   Monospaces :
      0 = Scenic
      1 = Espace

Citroen :
   Citadines = Saxo
   Berlines :
      0 = Xsara
      1 = Xantia

   Monospaces :
      0 = Picasso
      1 = Evasion
*/
?>
```

2) Les Templates

Introduction :

Je vais vous présenter les **templates** grâce à la **librairie PHPlib**, sous la forme d'un exemple simple. Nous allons faire une sélection de la **table clients** et afficher le contenu. Les **templates** vont nous permettre de mieux organiser le travail en équipe et d'augmenter la rapidité des mises à jour.

Arborescence :

Fichier liste.php
Dossier [gabarits]
Fichier liste.html
Dossier [librairies]
Fichier template.inc
le fichier liste.html

Ce fichier sera créé par **l'intégrateur html** en théorie (souvent le graphiste et l'intégrateur **html** ne font qu'un) et développé normalement en **html** grâce à un **éditeur type dreamweaver** ou **bloc notes.**

Exemple (code HTML simple)

```
 <html>
<head>
<title>Ma page client</title>
</head>

<body>
<table width="600" border="0">

  <tr bgcolor="#FFFF99">
   <td><font face="Verdana" size="2" color="#000000">Numero clients</font></td>
```

```
<td><font face="Verdana" size="2" color="#000000">Nom clients</font></td>
<td><font face="Verdana" size="2" color="#000000">Tel. clients</font></td>
<td><font face="Verdana" size="2" color="#000000">Adresse clients</font></td>
<td>

<tr bgcolor="#000066">
<td><font face="Verdana" size="2" color="#FFFFFF"></font></td>
<td><font face="Verdana" size="2" color="#FFFFFF"></font></td>
<td><font face="Verdana" size="2" color="#FFFFFF"></font></td>
<td><font face="Verdana" size="2" color="#FFFFFF"></font></td>
</tr>

</table>
</body>
</html>
```

Une fois le fichier fini, il va falloir rajouter un peu de code. On peut noter que le code à insérer n'est que du commentaire **html**.

Pour commencer un bloc, on utilise la balise : **<,!-- BEGIN nom_du_bloc -->**

Pour fermer un bloc, on utilise la balise : **<!-- END nom_du_bloc -->**

Pour insérer un champ, on utilise la balise : {nom_du_champ}

Les modifications pour notre exemple

```
 <html>
<head>
<title>Ma page client</title>
</head>

<body>
<table width="600" border="0">

<tr bgcolor="#FFFF99">
   <td><font face="Verdana" size="2" color="#000000">Numero clients</font></td>
   <td><font face="Verdana" size="2" color="#000000">Nom clients</font></td>
   <td><font face="Verdana" size="2" color="#000000">Tel. clients</font></td>
   <td><font face="Verdana" size="2" color="#000000">Adresse clients</font></td>
  <td>



<!-- BEGIN liste_clients -->


  <tr bgcolor="#000066">
```

```
    <td><font face="Verdana" size="2" color="#FFFFFF">{id}</font></td>
    <td><font face="Verdana" size="2" color="#FFFFFF">{nom}</font></td>
    <td><font face="Verdana" size="2" color="#FFFFFF">{tel}</font></td>
    <td><font face="Verdana" size="2" color="#FFFFFF">{adresse}</font></td>
  </tr>
 <!-- END liste_clients -->

</table>
 </body>
 </html>
```

La gestion du fichier **liste.html** est finie et implicitement le travail de l'intégrateur html. A noter qu'il n'y a eu que du code html, donc totalement gérable grâce à **dreamweaver**.

Le fichier liste.php

Le fichier liste.php

```php
 <?php
 //1
 include "librairies/template.inc";

 //2
 $db = mysql_connect('localhost', 'root', '');
 mysql_select_db('mabase',$db);
 $sql = 'SELECT id,nom,tel,adresse FROM clients';
 $req = mysql_query($sql) or die('Erreur SQL
!<br>'.$sql.'<br>'.mysql_error());

 //3
 $tpl = new template("gabarits");
 $tpl->set_file("gliste","liste.html");

 //4
 $tpl->set_block('gliste','liste_clients','liste_bloc');

 //5
 while($data = mysql_fetch_array($req)) {

 //6
 $tpl->set_var("id", $data['id']);
 $tpl->set_var("nom", $data['nom']);
 $tpl->set_var("tel", $data['tel']);
 $tpl->set_var("adresse", $data['adresse']);

 //7
 $tpl->parse('liste_bloc','liste_clients',true);
 }

//8
```

```
mysql_close();

//9
$tpl->pparse("affichage","gliste");
?>
```

Explications :

//1
On inclut le fichier de **template.inc** de **PHPLIB.**

//2
On crée une connexion et on fait une **requête select** de clients.

//3
Pour utiliser la commande **new template,** il faut indiquer le chemin vers les **gabarits (ici = ../gabarits).**
template (string $path)

Ensuite **set_file** va servir à choisir le gabarit et lui attribuer un nom dans le fichier PHP. Son utilisation se fait sous cette forme : nom du gabarit, fichier gabarit.
set_file (string $name_gabarit, string $path)

//4
Avec la commande **set_block** nous allons créer un bloc qui va correspondre au nom du traitement créé précédemment dans la **page html**. Son utilisation se fait sous cette forme : nom du gabarit, nom du traitement (dans **html**), nom du bloc.
set_block (string $name_gabarit, string $name_HTML,string $name_bloc)

//5
On fait une boucle qui va faire un tour pour chaque enregistrement.

//6
Grâce à **set_var** nous allons définir les champs notés dans le code **html**.
Cette commande s'exécute sous cette forme : nom du champ (dans **html**), valeur (ici récupéré dans la base. Pour chaque différent champ nous exécutons la commande **set_var**.
set_var(string $name_HTML, string $value)

//7
Ensuite nous utilisons la commande **parse** qui va nous servir à parcourir la boucle et à remplacer les champs par les valeurs de la db. Sa syntaxe est la suivante : nom du bloc, nom du traitement (dans html), true.
parse(string $name_bloc, string $name_HTML, true)

//8
on ferme la connexion à MySql.

//9
On finit par l'affichage, avec la commande pparse, qui s'écrit sous la forme :
Nom de l'affichage, nom du gabarit.

pparse(string $name, string $name_gabarit);

Pour tester :

Il va falloir créer une base **mabase** avec une table clients de cette forme (3 clients y sont insérés) :

Création de la table dans phpMyAdmin

```
CREATE TABLE clients (
id int(11) NOT NULL auto_increment,
nom varchar(30) NOT NULL,
tel varchar(255) NOT NULL,
adresse varchar(255) NOT NULL,
PRIMARY KEY (id));

INSERT INTO clients VALUES( '', 'Dupond pierre',
'02-98-56-69-32', '11 rue des lilas');
INSERT INTO clients VALUES( '', 'Duf marc', '02-98-54-78-03',
'15 route de la liberter');
INSERT INTO clients VALUES( '', 'legrand paul',
'02-98-03-70-96', '156 route des fleurs');
```

Ensuite il va falloir insérer le templates.inc dans le dossier librairies puis exécuter le fichier **liste.php**.(**http://phplib.sourceforge.net/)**

Alternatives : Il n'y a pas que **PHPlib** qui a un **système de templates**. Voici une petite liste des **différents systèmes de templates** :
XTemplates
Smarty
ModeliXe
Phemplate

3)LES COOKIES

Qu'est-ce qu'un cookie ?

Un **cookie est un fichier texte de taille limitée (65ko)** permettant de stocker certaines informations. Le but premier de ce type de fichier et de pouvoir garder en mémoire les informations d'un visiteur afin de pouvoir les réutiliser à chacune de ses visites. Par exemple on y stocke son **Nom**, son **Prenom**, son **Nickname**, etc. Bien sûr cela n'est possible qu'à partir du moment où le visiteur a entré lui-même ses informations dans un formulaire sur le site.

N.B: La légende qui dit que le cookie est capable de savoir quels sont les logiciels installés sur votre PC n'est pas fondée en soi, dans le sens où un fichier texte

n'exécute aucun programme, il est passif. Ceci dit, une fonction cachée dans un programme qui fait plusieurs MégaOctets peut très bien checker votre disque dur puis écrire des infos dans un cookie qui pourra ensuite être récupéré sur le web.

Où sont stockés les cookies ?

Cela dépend du navigateur que vous utilisez, pour cet exemple nous travaillerons avec **Internet Explorer** qui représente entre 80 et 90% du marché. **Les cookies** sont donc stockés (comme tous les autres fichiers temporaires du NET) dans **C:\windows\Temporary Internet Files**.

Comment générer un cookie avec PHP ?

C'est la fonction **setcookie()** qui permet l'envoi de cookie :

Code PHP

```
<?php
$expire = 365*24*3600; // on définit la durée du cookie, 1 an
setcookie("nickname","xeros",time()+$expire);  // on l'envoi
?>
```

Comme vous le voyez, **l'envoi d'un cookie** est vraiment simple, ici on envoie un cookie portant le nom : **nickname** et ayant la valeur : **xeros**, nous avons également indiqué que ce cookie avait une date de validité de 1 an **(date en cours + 365*24*3600)**, attention il faut toujours exprimer cette valeur en secondes !

N.B: Si vous souhaitez envoyer plusieurs cookies il vous suffit de mettre plusieurs setcookie() les uns à la suite des autres, je vous rappelle qu'ils sont tous stockés dans un seul et même fichier, comme ceci :

Code PHP

```
<?php
$expire = 365*24*3600;
setcookie("nickname","xeros",time()+$expire);
setcookie("nom","potec",time()+$expire);
setcookie("prenom","fabrice",time()+$expire);
?>
```

Comment lire **les cookies** quand un visiteur revient sur mon site ?

Là encore c'est un jeu d'enfant car **PHP** va regarder si un fichier contenant des éventuels cookies existe sur la machine du visiteur. Vous il ne vous reste plus qu'à appeler la valeur stockée dans le cookie à l'aide du **tableau global $_COOKIE** :

Code PHP

```
<?php // On affiche la valeur de nickname
echo "Bonjour $_COOKIE["nickname"]";
```

```
?>
```

Notez que si le visiteur n'a pas de **cookie** venant de votre site, rien ne sera affiché et aucun message d'erreur n'apparaîtra, d'où le côté pratique de cette utilisation.

Quelles sont les restrictions à connaître avec les cookies ?

1 .L'envoi d'un cookie doit être la première **fonction PHP** que vous utilisez dans votre script, donc tout en haut du script. Si d'autres fonctions interviennent avant l'envoi du cookie, celui-ci ne fonctionnera pas.

2 . Le script qui envoie le cookie doit être placé sur la racine de votre serveur et jamais dans un sous-répertoire, je ne dis pas que cela ne fonctionnera pas chez certains hébergeurs, mais je l'ai testé à plusieurs endroits et seul l'envoi depuis la racine fonctionnait parfaitement.

3 . Si vous envoyez un cookie sur un poste client celui-ci effacera automatiquement l'ancien cookie qui portait le même nom (si il y en avait un), autrement il le créera.

4 . Voici la page de la documentation traitant de la fonction **setcookie** : http://www.php.net/manual/fr/function.setcookie.php

4) Les fonctions

1 . Introduction

Une fonction est ce que l'on peut appeler un sous programme, une procédure. On distingue deux types de fonctions : les **fonctions intégrées** ou **built-in** qui sont incluses par défaut avec les distributions de **PHP** comme **print**, **echo** et les fonctions définies par le programmeur, dites aussi **fonctions utilisateur.**

Les fonctions ont plusieurs buts:

- Eclaircir le code en regroupant certaines fonctionnalités d'un programme qui se répètent dans une même fonction.
- Pouvoir créer des fonctions génériques qui pourront être utilisées dans d'autres programmes, ce qui évite de répéter pour chaque projet le même code.
- Possibilité d'évolution du code plus facile dans la mesure où lorsque vous modifiez le contenu d'une fonction, les répercussions sont effectuées sur l'ensemble du programme sans que vous ayez à le modifier dans la plupart des cas (sauf si vous rajoutez des paramètres etc).

2 . Syntaxe de déclaration

Une fonction se déclare de la manière suivante :

```php
<?php
function name_fonction ($params1, $params2, $params3, ... , $paramsN)
{
   // code de la fonction
return ($une_variable) // facultatif
}

?>
```

Analysons cette syntaxe de déclaration :

- L'instruction **function** qui déclare une fonction.
- Le nom de la fonction **name_fonction**.
- (...) La liste des paramètres séparés par une virgule, les arguments sont facultatifs, tout dépend du but de votre fonction.
- Une instruction **return** facultative qui permet de renvoyer une variable.

3 . Organiser l'écriture d'une fonction

Voici les différentes étapes nécessaires à la bonne élaboration d'une fonction simple à l'intérieur d'un script PHP:

1- Un en-tête qui regroupe des infos sur la fonction

/*
** **Function** : nom de votre fonction qui doit être court et explicatif
** **Input** : paramètre(s) en entrée entre () (ne pas trop en mettre)
** **Output** : paramètre(s) en sortie (la fonction ne peut renvoyer qu'une seule variable ou alors un tableau).
** **Description** : l'utilité de cette fonction
** **Creator** : le créateur de la fonction
** **Date** : date de création
*/

2- Déclaration de la fonction. La fonction à ici pour nom **ma_fonction**, comme input **$ma_var1**, **$ma_var2** et **$ma_var3**, en paramètre de sortie une variable boléenne (qui ne peut prendre que 2 valeurs true ou false), l'utilité étant de faire une comparaison sur les paramètres d'entrée.

```php
<?php

function ma_fonction ($ma_var1, $ma_var2, $ma_var3)
{
 if ($ma_var1 == $ma_var2)
   return (TRUE);
 else  if ($ma_var1 > $ma_var3)
     return (FALSE);
}

?>
```

4 . Voir l'utilité d'une fonction avec un exemple précis

Alors tout d'abord créer une fonction simple. Prenons l'exemple du code suivant:

1- le code brut dans le programme index.php par exemple :

```
<html>
<body>

<?php

  // afficher un menu de login membre simple
  echo '
  <table>
  <form action="liens.php" method="post" name="login_form">
  <tr>
  <td>Votre login</td>
  <td><input type="text" name="login" value="" size="20"
maxlength="42"></td>
  </tr>
  <tr>
  <td>Votre password</td>
  <td><input type="password" name="pass" value="" size="20"
maxlenght="12"></td>
  </tr>
  <tr>
  <td>
  <input type="submit" name="valider" value="val">
  <input type="reset" name="annuler" value="anu">
  </td>
  </tr>
  </form>
  </table> ';

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

Jusqu'ici rien de compliqué, bon vous me direz pourquoi utiliser une fonction dans ce cas-là vu que le code est court et simple ? Et bien si vous avez besoin de créer des formulaires de login membre dans plusieurs projets, au lieu de répéter le code à chaque fois dans le **.php** il vous suffit de stocker ce code dans une fonction et d'appeler simplement cette fonction avec les paramètres voulus.

2 - Créer une fonction à partir de ce code :

```
<?php

/*
```

```
** Function    : form_login_member
** Input       : aucun pour l'instant
** Output      : aucun
** Description    : affichage d'un formulaire de login de membre
** Creator    : polom
** Date        : 25/10/2001
*/

function form_login_member()
{
  echo '<table>';
  echo '<form action="liens.php" method="post" name="login_form">';
  echo '<tr>';
  echo '<td>Votre login</td>';
  echo '<td><input type="text" name="login" value="" size="20"
maxlength="42"></td>';
  echo '</tr>';
  echo '<tr>';
  echo '<td>Votre password</td>';
  echo '<td><input type="password" name="pass" size="20"
maxlenght="12"></td>';
  echo '</tr>';
  echo '<tr>';
  echo "<td>;
  echo '<input type="submit" name="valider" value="val">';
  echo '<input type="reset" name="annuler" value="anu">';
  echo '</td>';
  echo '</tr>';
  echo '</form>';
  echo '</table>';
}

?>
```

3 - Utiliser cette fonction dans le code précédent vu en 1 :

Plusieurs possibilités s'offrent à vous pour utiliser cette fonction à la place du code dans **index.php**, je vous propose deux possibilités pour intégrer cette fonction, sachant que la deuxième est préférable.

- Déclarer la fonction au début du fichier index.php, elle sera appelée ensuite dans le script.
- Créer un fichier qui regroupera vos fonctions ce qui permet de les retrouver facilement, vous ferez appel à ce fichier à l'aide de **include ("fonctions.inc.php")** ou **require ("fonctions.inc.php")** si votre fichier de fonctions s'appelle fonctions.inc.php bien sur :) toujours avant de faire appel à la fonction dans index.php

4 - le nouvel index.php avec la fonction :

Nous allons opter pour la deuxième solution, cad mettre notre fonction dans le fichier fonctions.inc.php, je ne mettrai pas le code, c'est le même que précédemment :), et faire appel à la fonction dans notre fichier index.php

```
<html>
<body>
<?php
include ("fonctions.inc.php");
form_login_member();
?>
</body>
</html>
```

Vous possédez un fichier **index.php** faisant appel à votre fichier fonctions.inc.php contenant votre fonction d'affichage de formulaire de login. Vous pourrez donc réutiliser cette fonction où bon vous semble en faisant appel au fichier .**inc.php**.

5 .Amélioration de votre fonction, utilisation des paramètres

On appelle paramètres d'une fonction, les variables qui lui sont passées en argument c'est à dire entre (), ils sont séparés par une virgule, la fonction n'est pas obligée de recevoir des paramètres.

Supposons maintenant que nous voulons faire évoluer la fonction précédente avec des paramètres :

- La page vers laquelle sera redirigé le formulaire apres validation.
- Une valeur par défaut pour le champ login.
- Des nouveaux noms pour les boutons submit.
- Les descriptifs des champs à remplir.

1 - Passer les paramètres à la fonction :

L'en-tête change : on définit l'input, il est conseillé de préciser le type de la variable même si php n'est pas un langage extrêmement typé, ça pourra vous aider plus tard si vous ne savez plus à quoi correspondent les différentes variables.

```
<?php
/*
** Function    : form_login_member
** Input       : STRING $action, STRING $login_def, STRING
$sub_name, STRING $reset_name
** Output      : aucun
** Description    : affichage d'un formulaire de login de membre
** Creator    : polom
** Date        : 25/10/2001
*/

function form_login_member($action, $login_def, $sub_name, $reset_nam
```

```
e, $login_txt, $pass_txt)
{
  echo '<table>';
  echo '<form action="'.$action.'" method="post" name="login_form">';
  echo '<tr>';
  echo '<td>'.$login_txt.'</td>';
  echo '<td><input type="text" name="login" value="'.$login_def.'"
size="20" maxlength="42"></td>';
  echo '</tr>'
  echo '<tr>'
  echo '<td>'.$pass_txt.'</td>'
  echo '<td><input type="password" name="pass" size="20"
maxlenght="12"></td>';
  echo '</tr>';
  echo '<tr>';
  echo '<td>';
  echo '<input type="submit" name="'.$sub_name.'" value="val">';
  echo '<input type="reset" name="'.$reset_name.'" value="anu">';
  echo '</td>';
  echo '</tr>';
  echo '</form>';
  echo '</table>';
}

?>
```

2 - Utiliser la fonction avec les paramètres :

Toujours dans notre fichier principal index.php :

```
<html>
<body>

<?php
include ("fonctions.inc.php");
$action     = "ma_page_de_validation.php";
$login_def  = "Ici votre login";
$sub_name   = "Envoyer";
$reset_name = "Annuler la saisie";
$login_txt  = "Votre login :";
$pass_txt   = "Votre password :";

form_login_member($action, $login_def, $sub_name, $reset_name, $login
_txt, $pass_txt);

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

Et voila vous avez une belle fonction qui affiche un formulaire de login paramétrable propre et clair. Rien ne vous empêche de modifier vous-même maintenant cette fonction pour la faire évoluer, vous pouvez par exemple passer les tailles des boutons, des champs etc.

PS : Vous pouvez bien sûr appeler des fonctions à l'intérieur d'autres fonctions.
PS2 : Il n'est pas obligatoire de nommer les variables passées dans la déclaration de la fonction de la même façon que celles passées dans le script principal, cependant cela demeure plus clair d'utiliser les mêmes noms de variables dans les deux cas.
PS3 : Vous pouvez affecter des valeurs par défaut aux paramètres en les déclarant comme ceci : **function toto($nbr, $nbr2 = "42")** et la si vous ne précisez pas le paramètre **$nbr2** il sera mis à **42** par défaut. Ceci est valable uniquement pour les variables passées par valeur (voir ci-dessous).

6 . Paramètres par valeur ou par référence

Les paramètres d'une fonction peuvent être passés de deux façons différentes :

- **Par valeur**, c'est à dire que s'ils ont une valeur à l'extérieur de la fonction, seule la valeur est transmise à la fonction, si la variable subit des modifications à l'intérieur de la fonction, ces modifications ne seront pas perçues dans le programme principal.

- **Par référence**, avec le signe **&** avant la variable (ex : &$cpt). Dans ce cas-là, l'adresse mémoire de la variable dans le programme est passée à la fonction et toute modification de cette variable dans la fonction aura des répercutions à l'extérieur du programme.

Voici un exemple de programme utilisant une **fonction avec un passage par valeur et ensuite par référence** :

```
<?php
function modif_tab($tab)
{
  $tab[0] = "j'aime le perl mais je préfere le php.";
}
?>

<html>
<body>

<?php

$tab = array("salut", "j'aime le fortran et le cobol");
// passage de $tab par valeur, la boucle for affichera
// salut, j'aime le fortran et le cobol

modif_tab($tab);
for ($i = 0; $tab[$i]; $i++)
    echo "$tab[$i] ";
```

```
// passage de $tab par référence, la boucle for affichera
// salut, j'aime le perl mais je préfere le php.

modif_tab(&$tab);
for ($i = 0; $tab[$i]; $i++)
   echo $tab[$i] ;
?>
</body>
</html>
```

7 . Retourner des valeurs dans une fonction

Nous allons utiliser un exemple d'operations sur deux nombres : addition, multiplication, division par exemple toujours dans un fichier **index.php**. Le résultat de l'opération entre ces deux nombres sera calculé dans la fonction puis retourné à l'aide de l'instruction **return($result).**

NB: une fois arrivé à **l'instruction return**, tout traitement qui suit est arreté.

1 - le programme dans **index.php** qui fait appel à la fonction :

```
<html>
<body>
<?php
include ("functions.inc");
// operations "add", "sou" "mul"
$operation = "add";
$nbr1      = 42;
$nbr2      = 69;
$result    = 0;
// recuperation du resultat de 42 + 69
$result = do_op($nbr1, $nbr2, $operation);
// affichage du resultat;
echo $result;
// affiche 111
?>
</body>
</html>
```

Donc ici nous initialisons les deux nombres ainsi que l'opération à effecuter, puis on fait appel à la **fonction do_op** contenue dans le fichier inclus dans **fonctions.inc.php**

2 - La fonction qui effectue les opérations :

```
<?php
/*
** Function    : do_op
** Input       : INT nbr1, INT nbr2, STRING operation
** Output      : INT result
```

```
** Description    : effectue une opération addition, multiplication ou division
** Creator    : polom
** Date       : 25/10/2001
*/
function do_op($nbr1, $nbr2, $operation)
{
 if ($operation == "add")
   return ($nbr1 + $nbr2);
 else if ($operations == "mul")
   return ($nbr1 * $nbr2);
 else if ($operations == "div")
   return ($nbr1 / $nbr2);
 return (NULL);
}
?>
```

La fonction reçoit donc les deux nombres et l'opération à faire, elle retourne le résultat une fois l'opération faite. Vous pouvez aussi ne pas retourner directement le résultat dans la fonction précédente et le stocker, puis le retourner à la fin de la fonction. Sachez que une seule valeur peut être retournée dans une fonction, si vous voulez retourner plusieurs valeurs, il faut à ce moment là utiliser un tableau ou des variables globales (voir ci-dessous).

8 . Variables globales et statiques

Nous avons vu comment passer des variables à une fonction, il existe deux types particuliers de variables utilisées souvent dans les fonctions. Ce sont **les variables globales et statiques.** Une **variable globale** déclarée à l'intérieur d'une fonction à l'aide de l'instruction **global** permet à une variable d'être accessible en dehors de la fonction, vous pouvez accéder à cette fonction par son nom ou à l'aide du tableau **$GLOBALS**.

1 - **Variables globales** :

```
<?php
function glob_fonc()
{
 global $cpt;
$cpt = 1;
} ?>
```

Ici **$cpt** sera accessible en dehors de glob_fonc après son appel et il aura la valeur 1, vous pouvez aussi y accéder avec sa valeur dans le tableau de globales c'est à dire **$GLOBALS["cpt"]**.

2 - Variables statiques :

Une **variable statique** déclarée à l'intérieur d'une fonction à l'aide de l'instruction **static** permet à une variable de garder sa valeur à chaque appel de la fonction.

L'initialisation d'une variable statique se fait au début de la fonction et à chaque appel de la fonction dans le script elle gardera la valeur du dernier appel.

```
<html>
<body>
<?php
function stat_fonc()
{
 static $cpt = 0;
 $cpt++;
 echo $cpt;
}
stat_fonc();
// affiche 1 c'est-à-dire premier appel à la fonction, $cpt est initialisé à 0 une seule fois
// et incrementé de 1
stat_fonc();
// affiche 2, $cpt a gardé la valeur précédente et l'incrémente de 1.
?>
</body>
</html>
?>
```

9 . Récursivité

Qu'est ce que la récursivité ? La récursivité est le fait d'une fonction qui se rappelle elle-même au moins une fois jusqu'à atteindre un résultat voulu. Prenons un exemple simple pour expliquer ce principe à travers le calcul de la puissance d'un nombre entier.

1 - Le code dans **index.php** :

```
<html>
<body>
<?php
$nbr = 2;
$puissance = 4;
$result = puissance($nbr, $puissance);
echo $result;
?>
</body>
</html>
```

On affiche donc ici le résultat de 2 à la puissance 4 c'est à dire 16.

2 - la fonction normale sans récursion :

```
<?php
/*
** Function    : puissance
```

```
** Input       : INT nbr, INT puissance
** Output      : INT result
** Description    : effectue la puissance d'un nombre entier
** Creator    : polom
** Date       : 25/10/2001
*/
function puissance ($nbr, $puissance)
{
  for ($total = $nbr; $puissance > 1; $puissance--)
     $total = $total * $nbr;
  return ($total);
}
?>
```

Donc là rien de bien compliqué on initialise le résultat au nombre donc si on a par exemple **2 puissance 4** on a la trace suivante (une trace est l'état de vos variables tout le long de votre programme, c'est une méthode intéressante pour suivre l'évolution de vos variables et débuger entre autre :

-> départ $total = 2;

4 > 1 ok
$total = 4;
3 > 1 ok
$total = 8;
2 > 1 ok
$total = 16;
1 n'est pas > à 1 donc on arrète le traitement

-> fin $resultat = 16

3 - **La même fonction en récursif** :

Le principe est de rappeler toujours la fonction **recurse_puissance** tant qu'on est pas arrivé à une puissance 1 comme dans le for précédemment. On appellera cette fois-ci la fonction puissance de cette façon toujours dans le même programme vu avant :
-> recurse_puissance(2, 4, 2);

```
<?php
function recurse_puissance($nbr, $puissance, $total)
{
if ($puissance > 1)
return (recurse_puissance($nbr, $puissance - 1, $total * nbr));
return ($total);
}
?>
```

Et encore plus fort et plus beau en une seule ligne :

```
<?php
function recurse_puissance($nbr, $puissance, $total)
```

```
{
return (($puissance > 1) ? recurse_puissance($nbr, $puissances -
1, $total * $nbr) : $total);
}
?>
```

Il s'agit dans le deuxième exemple d'une utilisation d'une expression ternaire qui s'explique de la façon suivante. On teste si **$puissance** est supérieur à 1, si c'est le cas on rappelle la fonction avec **$puissance** décrémentée de 1 et le nouveau total, sinon on renvoie le **$total** final de l'opération.

Voici la trace de notre fonction récursive pour comprendre son utilité et son fonctionnement :

-> départ $puissance = 4, $total = 2;

4 > 1 ok
on renvoie la fonction avec $puissance décrémenté de 1 cad 3 et le $total de 2 * 2 = 4;
$total = 4;
3 > 1 ok
on renvoie la fonction avec $puissance décrémenté de 1 cad 2 et le $total de 4 * 2 = 8;
$total = 8;
2 > 1 ok
on renvoie la fonction avec $puissance décrémenté de 1 cad 1 et le $total de 8 * 2 = 16;
$total = 16;

1 pas supérieur à **1** on arrête de renvoyer la fonction, on renvoie **$total**. **La récursivité** est donc une fonctionnalité qui permet de construire des **algorithmes puissants**, rapides et souvent plus réduits qu'en passant par la technique normale littérale.

10 . Fonctions dynamiques et bonus track :

Vous pouvez vous trouver dans le cas où vous ne savez pas quelle fonction devra être appelée à tel moment du script. Pour pouvoir gérer cela, il vous suffit de placer dans une variable le nom d'une fonction, puis d'utiliser cette variable comme une fonction.

```
<?php
function write($text)
{
 print($text);
}
function writeBold($text)
{
 print("<b>$text</b>");
}
$fonction_var = "write";
$fonction_var("toto"); // affiche toto
$fonction_var = "writeBold";
```

```
$fonction_var("toto"); // affiche toto en gras
?>
```

Il existe trois fonctions utiles dans la gestion des arguments passés dans une fonction dans **php4**. Il s'agit de :

- **func_get_arg** qui permet de lire un argument spécifique.
- **func_get_args** pour obtenir l'ensemble des arguments sous forme d'une matrice ou tableau.
- **func_num_args** pour connaître le nombre d'arguments reçus par la fonction.

5)Classes et objets I

Première partie: Les classes et les objets

Certains trouveront ces concepts difficiles ou même inutiles. Sachez néanmoins que la **POO** est la **méthode majeure de programmation dans de nombreux languages** (**C++, Java, JavaScript...**) et qu'elle possède des avantages indéniables.

Prérequis: Les **bases du PHP**, **déclaration de variables, visibilité des variables, les fonctions en PHP**.

I) Qu'est ce que c'est ?

La POO (Programmation Orientée Objet) est une manière de programmer un peu différente de celle que vous pouviez utiliser jusqu'à présent en **PHP**. En effet, le corps de vos scripts était toujours composé de plusieurs fonctions. La **POO** va un peu bousculer tout cela, mais pour votre bien vous verrez.

La POO a deux buts principaux:

- faciliter la réutilisation du code que vous avez déjà écrit: **c'est l'héritage**
- **l'encapsulation des données** et les traitements correspondants

II) Un premier exemple d'encapsulation et de traitement

Jusqu'ici, tout vous parait obscur, nous allons donc étudier un exemple (il vaut ce qu'il vaut, et de toutes façon, il y a à peu près le même dans la documentation officielle de PHP). Imaginons que vous vouliez gérer les commandes d'une pizzeria pour chaque client. Vous devez donc, pour chaque client: mémoriser son nom, comptabiliser le type des pizzas commandées, leur nombre et enfin calculer le prix total de la commande.

Jusqu'à présent, vous auriez défini de nombreuses variables pour chaque client, ou mieux un tableau. Nous allons voir une autre façon de le faire, en utilisant une classe:

a) Le nom du client

```
<?php
class Commande {
var $nomClient;
}
```

```
?>
```

Voila déjà une première partie. Passons aux explications: la classe **Commande** déclarée ci-dessus contient une variable dite **variable membre** nommée **$nomClient**. L'utilisation est assez simple:

```
<?php
//on indique à PHP que la variable $client contiendra
//des données définies dans la classe Commande
$client = new Commande();
//pour la variable $client, qui est de type "Commande",
//on donne à la variable membre $nomClient la valeur "debut"
$client->nomClient="debut";
?>
```

Et vous pouvez vérifer que:

echo $client->nomClient; renvoie bien **debut**.

b) La liste des pizzas commandées

On va modifier la **classe Commande** pour y ajouter une **autre variable membre** afin de gérer la liste des pizzas. Pour simplifier, on utilisera un tableau:

```
<?php
class Commande {
var $nomClient;
var $listePizzas;
}
$client = new Commande();
$client->nomClient="debut";
$client->listePizzas[0] = 5;  // disons que 0 est le code pour "Pizza Royale", j'en veux 5
$client->listePizzas[1] = 2;  // je veux deux "Campagnarde" (n°1)
?>
```

La encore, pas grand chose de neuf, l'indice du tableau correspondant à un type de pizza. Maintenant, il faut constater quelque chose. Vous avez vu que l'ajout de nouveaux éléments n'était pas très intuitif. On va donc ajouter des **fonctions membres**:

c) Ajouter des pizzas

```
<?php
class Commande {
var $nomClient;
var $listePizzas;
function ajouterRoyale($nombre) {
$this->listePizzas[0] += $nombre;
}
 function ajouterCampagnarde($nombre) {
 $this->listePizzas[1] += $nombre;
}
```

```
}
$client = new Commande();
$client->nomClient="debut";
$client->ajouterRoyale(5);  // je veux 5 "Pizza Royale"
$client->ajouterCampagnarde(2);  // je veux deux "Campagnarde"
?>
```

Et ici, l'interet des classes commence à apparaitre. En effet, en plus **des variables membres** définies depuis le début, on a ajouté des **fonctions membres**. Qu'est ce qu'une fonction membre ? Il s'agit d'une fonction qui a pour but d'intervenir sur les variables membres de la classe. Le **mot clé $this** est en fait identique à la variable d'appel.

```
<?php
$client1 = new Commande();
$client1->nomClient="debut";
$client1->ajouterRoyale(5);  // je veux 5 "Pizza Royale"
$client2 = new Commande();
$client2->nomClient="Anne";
$client2->ajouterRoyale(1);  // elle veux 1 "Pizza Royale"
(régime...)
?>
```

Lors de l'appel **$client1->ajouterRoyale(5);** la **variable $this** de la fonction membre **ajouterRoyale** prendra la valeur **$client1** et c'est donc sur les variables membres de celle-ci que s'effectura l'ajout. Lors de l'appel **$client2->ajouterRoyale(1); $this** prendra la valeur **$client2.**

d) Pour compléter

Pour obtenir ce que nous voulions depuis le début:

```
<?php
class Commande {
// définissons les variables correspondant au prix (en euros bien sur) des pizzas.
// On leur donne une valeur par défaut.
var $prixRoyale = 6;
var $prixCampagnarde = 8;
// et les variables qu'on connait déjà
var $nomClient;
var $listePizzas;
function ajouterRoyale($nombre) {
$this->listePizzas[0] += $nombre;
}
function ajouterCampagnarde($nombre) {
$this->listePizzas[1] += $nombre;
}
function calculerPrix() {
$montant_Royale = $this->listePizzas[0] * $this->prixRoyale;
$montant_Campagnarde = $this->listePizzas[1] * $this->prixCampagnarde;
```

```
return $montant_Royale + $montant_Campagnarde;
}
function afficherCommande() {
echo "Commande du client : ".$this->nomClient;
echo "<br>Pizza(s) 'Royale' : ".$this->listePizzas[0];
echo "<br>Pizza(s) 'Campagnarde' : ".$this->listePizzas[1];
echo "<hr>Totale de votre commande : ".$this->calculerPrix();
// on peut également appeller des fonctions membres
echo " Euros<br>";
}}
$client1 = new Commande();
$client1->nomClient="debut";
$client1->ajouterRoyale(5);
$client1->ajouterCampagnarde(2);
$client1->afficherCommande();
?>
```

Ce qui donne:
Commande du client : debut
Pizza(s) 'Royale' : 5
Pizza(s) 'Campagnarde' : 2

Total de votre commande : 46 Euros

Attention: Une erreur très fréquente est d'écrire les appels comme suit: **$this->$nomClient**
Ceci ne renverra pas d'erreur, mais le résultat ne sera pas celui attendu.
Il ne faut jamais placer, dans cette syntaxe, de signe $ devant le nom du variable membre. Ici, il fallait donc écrire: **$this->nomClient**
Nous avons donc écrit une classe tout à fait fonctionnelle. L'interêt est assez important. Il est vrai que vous avez écrit pas mal de code pour un maigre résultat mais, il y a des avantages indéniables:

- **Clarté du code**: la classe peut être placée dans un fichier séparé que l'on appellera via la commande **include().** L'utilisation de **l'opérateur ->** rend le code clair et compréhensible.
- **Facilité de réutilisation**: la clarté du code facilite la réutilisation du code dans d'autres projets et même par d'autres personnes. Cet aspect sera développé grâce à l'héritage.

III) Constructeur de classe

Pour compléter cette présentation des classes, il reste un aspect important mais simple à traiter: le **constructeur de classe**. En effet, en considérant notre cher exemple des pizzas, vous pouvez remarquer que le champ **$nomClient** peut ne pas être initialisé:

```
<?php
class Commande {
// Identique à ce qui a été défini précedemment
}
$client1 = new Commande();
$client1->ajouterRoyale(5);
```

```
$client1->ajouterCampagnarde(2);
$client1->afficherCommande();
?>
```

Ce qui donne:
Commande du client :
Pizza(s) 'Royale' : 5
Pizza(s) 'Campagnarde' : 2

Total de votre commande : 46 Euros

Le nom du client est vide, en va corriger ce problème en plaçant un nom par défaut (ex: SansNom). C'est le constructeur de classe qui va s'en charger. Le constructeur est en fait une fonction membre qui porte impérativement le même nom que la classe. Cette fonction est appellée automatiquement lors de la **création d'un objet** (donc au moment de l'appel **$client1 = new Commande();)** C'est donc le moment privilégié pour réaliser les initialisations que l'on désire.

a) Un nom par défaut

Pour cela, il suffit de modifier la **classe Commande**:

```
<?php
class Commande {
var $prixRoyale = 6;
var $prixCampagnarde = 8;
var $nomClient;
var $listePizzas;
//Constructeur de la classe car même nom:
function Commande() {
$this->nomClient = "SansNom";
}
function ajouterRoyale($nombre) {
$this->listePizzas[0] += $nombre;
}
 function ajouterCampagnarde($nombre) {
$this->listePizzas[1] += $nombre;
}
 function calculerPrix() {
$montant_Royale = $this->listePizzas[0] * $this->prixRoyale;
$montant_Campagnarde = $this->listePizzas[1] * $this->prixCampagnarde;
return $montant_Royale + $montant_Campagnarde;
}
function afficherCommande() {
 echo "Commande du client : ".$this->nomClient;
echo "<br>Pizza(s) 'Royale' : ".$this->listePizzas[0];
echo "<br>Pizza(s) 'Campagnarde' : ".$this->listePizzas[1];
echo "<hr>Totale de votre commande : ".$this->calculerPrix();
echo " Euros<br>";
   }}
$client1 = new Commande();
```

```
$client1->ajouterRoyale(5);
$client1->ajouterCampagnarde(2);
$client1->afficherCommande();
?>
```

Ce qui donne:
Commande du client : SansNom
Pizza(s) 'Royale' : 5
Pizza(s) 'Campagnarde' : 2

Total de votre commande : 46 Euros
Merci.

La **fonction membre Commande() est un constructeur** car elle possède le même nom que la classe. Elle est appellée en premier et initialise la variable membre à la valeur "SansNom". Bien sûr, il est toujours possible de modifier le nom du client **avec $client1->$nomClient="debut";** De plus tout ceci est équivalent à avoir déclaré une valeur par défaut: **var $nomClient = "SansNom";**

b) Paramètres du constructeur

On vient de voir que le constructeur permet d'initialiser les variables membres et plus généralement, d'effectuer divers traitement initiaux.

Dans notre exemple, le nom du client ne devrait pas changer et on peut donc le choisir à la création de l'objet, et donc se charger de cette initialisation dans le constructeur.

Il suffit donc de modifier le constructeur en lui ajoutant un paramètre. De plus, dans le code qui suit, la variable **$nomClient** contient maintenant une valeur par défaut.

```
<?php
class Commande {
var $prixRoyale = 6;
var $prixCampagnarde = 8;
var $nomClient = "SansNom";
var $listePizzas;
//Constructeur avec paramètre
function Commande($nom) {
if ($nom != "") $this->nomClient = $nom;
}
function ajouterRoyale($nombre) {
$this->listePizzas[0] += $nombre;
}
function ajouterCampagnarde($nombre) {
$this->listePizzas[1] += $nombre;
}
function calculerPrix() {
$montant_Royale = $this->listePizzas[0] * $this->prixRoyale;
$montant_Campagnarde = $this->listePizzas[1] * $this->prixCampagnarde;
return $montant_Royale + $montant_Campagnarde;
```

```
}
function afficherCommande() {
echo "Commande du client : ".$this->nomClient;
echo "<br>Pizza(s) 'Royale' : ".$this->listePizzas[0];
echo "<br>Pizza(s) 'Campagnarde' : ".$this->listePizzas[1];
echo "<hr>Totale de votre commande : ".$this->calculerPrix();
echo " Euros<br>";
}
}
$client = new Commande("debut");
$client->ajouterRoyale(2);
$client->ajouterCampagnarde(3);
$client->afficherCommande();
?>
```

Ce qui donne:
Commande du client : debut
Pizza(s) 'Royale' : 2
Pizza(s) 'Campagnarde' : 3

Totale de votre commande : 36 Euros
Remarques: Il ne peut exister qu'un seul constructeur par classe. Il est conseillé de placer chaque classe dans un fichier portant le nom de cette classe (ex: **Commande.class.php**) et d'utiliser des **include().** Ceci sera très pratique pour la suite.

IV) Vocabulaire, utilité et conclusion

Pour commencer, un peu de vocabulaire:

- On dit que **Commande** est une **classe**.
- **$client = new Commande("debut");**
  On dit alors que **$client** est un **objet** de type **Commande**. On dit également que **$client** est une **instance** de la classe **Commande**.
- **variable membre** et **variable d'instance** sont synonymes.
- **fonction membre** et **méthode** sont synonymes.

Il est tout à fait possible de créer des objets d'une autre classe au sein même d'une classe. Ainsi, on aurait ici pu créer un objet de type **MoyenDePaiement** dans la **classe Commande**. Ces liens entre les classes sont **très utilisés.** De plus la **notion de constructeur** est très importante en **PHP** car elle permet d'initialiser les variables de la classe, notamment avec une **base de données MySQL** (lecture dans une table et affectation aux variables membres). Dans ce cas, il resterait à écrire une **fonction membre** permettant de sauvegarder **les variables membres** dans une **table SQL**.

Ceci dit, comment utiliser les notions de programmation objet vues précedemment dans vos projets ? Le cas de la commande des pizzas peut aisément se rapprocher d'un **shopping cart** comme on en trouve sur les sites marchands.Mais globalement, le **concept de programmation par objet peut s'appliquer à tous les problèmes**.

Quelle que soit l'application, le schéma est souvent le même, un objet est créé à l'entrée sur le site et est conservé tout le long de la navigation via l'utilisation des sessions. Ainsi, vous

pouvez créer une **classe Visiteur** qui mémorisera toutes les actions d'un visiteur (arrivée sur une page, clic sur un lien, etc.) et vous pourrez facilement créer des statistiques ou autres informations plus complexes.

Une fois que toutes ces bases ont été vues, il nous restera à voir:

- **L'héritage:** pour pouvoir créer de nouvelles classes basées sur une classe déjà existante en ajoutant ou modifiant des fonctions membres. (Nous verons également le polymorphisme)
- **La serialisation:** pour sauvegarder/charger des objets à partir de fichiers.

Et eventuellement quelques exemples concrets.

6)Classes et objets II
Partie 2: L'héritage.

Après une introduction sur les classes et une étude d'exemple plutôt gastronomique, un thème essentiel doit maintenant être abordé: l'**héritage**. Ici encore, le **PHP** permet de faire un certain nombre de choses bien pratiques, même si l'on est encore loin des possibilités offertes par d'autres langages. Quant au concept se cachant derrière le terme de **polymorphisme**, mieux vaut attendre d'avoir traité en détail l'**héritage** avant d'en parler.

I) Définition

L'héritage permet de **spécialiser le code et/ou réutiliser du code déjà existant** pour l'adapter à ses besoins. En d'autres termes, il permet, à partir d'une classe déjà existante, d'en créer une nouvelle qui reprendra ses caractéristiques ce qui nous permettra de les adapter à d'autres besoins sans modifier la classe de base. Comme toujours, tout ceci peut paraître abstrait, passons donc à un premier exemple:

Imaginons que vous ayez une classe **CompactDisc** permettant de gérer votre **stock de CD**. Elle pourrait être définie de la manière suivante:

Classe: "CompactDisc"
Variables membres:
$titre, $dateAchat
Fonctions membres:
afficherTitre()
afficherDateAchat()

Pour simplifier, je n'ai pas donné le code de cette classe, mais juste les éléments qui la composent. Admettons que c'est un peu basique et que cette classe ne fait pas grand chose.Mais elle a tout de même l'immense avantage d'être valable quel que soit le type de **CD**. Maintenant, si l'on veut gérer des **CD audio**, il faudrait ajouter certains éléments (genre musical par exemple): c'est **la spécialisation**. Pour cela, nul besoin de modifier directement cette classe (elle pourra toujours reservir), il nous suffit d'en créer une nouvelle qui héritera de toutes les caractéristiques de la classe de base **CompactDisc**. Pour cela, on défini la classe **CDAudio**:

Classe: "CDAudio"
Variables membres:
$genreMusical
Fonctions membres:
afficherGenreMusical()

Il ne reste plus qu'à indiquer à **PHP** que la classe **CDAudio** hérite de la classe **CompactDisc** pour que toutes **les variables et fonctions membres** de **CompactDisc** soient données à **CDAudio** pour arriver au résultat suivant: (**en rouge** sont indiqués les éléments hérités.)

Classe: "CDAudio" hérite de "CompactDisc"
Variables membres:
$genreMusical, $titre, $dateAchat
Fonctions membres:
afficherGenreMusical()
afficherTitre()
afficherDateAchat()

Et c'est en créant un objet de type **CDAudio** qu'on obtiendra le résultat voulu. A partir de là, on pourra définir des classes héritant également de **CompactDisc (CDROMLogiciel, CDROMJeu, etc.)** mais aussi des classes qui héritent de **CDAudio (ex: CDAudioSingle)** et qui hériteront donc également de **CompactDisc**.

Toutes ces notions d'héritage peuvent être representées sur un diagramme (simplifié):

II) Mise en oeuvre de l'héritage

a) Syntaxe

```
class CDAudio extends CompactDisc {
 // ...
}
```

Voila tout simplement la syntaxe à utiliser. Ici, la classe **CDAudio** hérite de la classe **CompactDisc**.

Un peu de vocabulaire:

- on dit que **CDAudio hérite** de **CompactDisc**
- on dit que **CompactDisc** est la **classe de base** de **CDAudio**
- on dit que **CDAudio est un** (**is a** en anglais) **CompactDisc** (terme employé pour le **polymorphisme**)

L'héritage multiple est impossible en PHP. En d'autres termes, CDAudio peut hériter de CompactDisc mais pas de CompactDisc et de Musique simultanément. Bien sur, le contraire est possible (comme on l'a vu précedemment): une classe peut servir de base à plusieurs classes (ex: CDAudio et CDROMLogiciel héritent de

CompactDisc). Notez que l'impossibilité de faire de l'héritage multiple peut poser problème mais cela permet d'éviter de nombreuses confusions.

b) Exemple

Appliquons tout ceci aux classes **CompactDisc** et **CDAudio**. Pour cela, nous allons mettre en pratique **un conseil de premier tutoriel: un fichier PHP par classe**.

Fichier "CompactDisc.class.php"

```
<?php
class CompactDisc {
 var $titre, $dateAchat;
function afficherTitre() {
echo "<br>Le titre est: ".$this->titre;
}
function afficherDateAchat() {
echo "<br>CD acheté le: ".$this->dateAchat;
}
}
?>
```

Fichier "CDAudio.class.php"

```
<?php
class CDAudio extends CompactDisc {
var $genreMusical ;
function afficherGenreMusical() {
echo "<br>Le genre musical est: ".$this->genreMusical;
}
}
?>
```

Fichier "index.php"

```
<?php
include ("CompactDisc.class.php");
include ("CDAudio.class.php");
$cd = new CDAudio();
$cd->titre = "Rock'n PHP";
$cd->dateAchat = "25/12/2001";
$cd->genreMusical = "Rock";
$cd->afficherTitre();
$cd->afficherDateAchat();
$cd->afficherGenreMusical();
?>
```

III) Héritage des constructeurs

Les comportements sont différents selon **PHP3 ou PHP4**. On ne parlera ici que de PHP4. Dans tous les cas, on a vu qu'une classe pouvait posséder un constructeur, appellé à l'initialisation. Voyons ce qu'il se passe dans le cas de l'héritage.

a) Sous PHP 4 (et surement les versions suivantes)

Règle n°1: "Si une classe n'a pas de constructeur, le constructeur de la classe de base est appellé (s'il existe)".

Cette règle est assez claire mais impose tout de même de faire attention. Regardez l'exemple suivant:

```
class Pere {
var $nombreEnfants;
 // constructeur:
function Pere ($n) {
$this->nombreEnfants = $n;
 }
}
class Fils extends Pere {
var $nom;
}
```

Pour créer un objet de type Fils et sachant que la classe Fils ne possède pas de constructeur, vous seriez peut être tenté par la syntaxe suivante:
$objet = new Fils();
Sachez que cela ne marchera pas, car en vertu de la règle n°1, **PHP** va appeller le constructeur de la classe de base puisqu'aucun constructeur n'existe dans la **classe Fils**. Or le constructeur de la classe de base prend un paramètre qu'il est obligatoire de spécifier:
$objet = new Fils(1); // initialise la **variable membre $nombreEnfants** de la **classe Fils (héritée de Pere)** à la valeur 1 via le **constructeur de la classe Pere**.
Je vous accorde que ce comportement peut sembler légèrement dérountant.

Règle n°2: Un constructeur est une fonction portant le même nom que la classe dans laquelle elle est définie.
Attention: cette règle ne fonctionne pas correctement malgré ce qui est annoncé dans la documentation. Pour le moment, le comportement de PHP4 (au moins jusqu'à 4.0.6) est le même que PHP3.

Vous connaissiez déjà une partie de cette règle :

```
class Pere {
var $nombreEnfants;
function Fils() {
$this->nombreEnfants++;
}
}
class Fils extends Pere {
var $nom;
}
```

Si l'on considère ces deux classes de manière indépendante, il n'y a pas de constructeur. Néanmoins, par le jeu de l'héritage, la **fonction membre Fils()** de la **classe de base Pere** se trouvera disponible dans la **classe Fils.** On a vu jusqu'à présent qu'une fonction membre est un

constructeur si elle porte le même nom que la classe dans laquelle elle est. Ceci n'est pas valable si la fonction est héritée. Ainsi, **la classe Fils** n'aura pas de constructeur mais possèdera une fonction membre **héritée nommée Fils()**.

b) Sous PHP 3

Seule et unique règle: Un **constructeur** est une **fonction portant le même nom que la classe** dans laquelle il est appellé, même si celui-ci n'a pas été défini dans cette classe.

Ainsi, si une classe n'a pas de constructeur, aucun constructeur n'est appellé, pas même celui de sa classe de base s'il existe. Dans l'exemple suivant, la **fonction membre Fils()** apparait dans la classe **Fils** par héritage et devient ainsi un constructeur de la **classe Fils**.

```
class Pere {
var $nombreEnfants;
function Fils() {
$this->nombreEnfants++;
}
}
class Fils extends Pere {
 var $nom;
}
```

c) Propagation des appels au constructeur

En **PHP4,** on a vu que s'il n'y a pas de **constructeur dans une classe**, celui de la classe de base est appellé. Par contre, si un constructeur existe, celui de la classe de base ne sera pas automatiquement appellé, vous devrez donc (si nécessaire) appeller vous même le constructeur de la classe de base. **En PHP3**, quoiqu'il arrive, vous devez effectuer tous les appels aux constructeurs de manière manuelle.

IV) Redéfinition de fonctions membres et mot clé parent.

L'**héritage** permet également de redéfinir des fonctions membres. Le but majeur est bien sur d'adapter une classe à ses besoins en modifiant certaines de ses caractéristiques.

Le **mot clé parent** permet quant à lui d'accéder à partir d'une **classe aux fonctions et variables membres** de la classe de base.

Appliquons ces deux concepts à l'exemple de **CDAudio** et d'une nouvelle classe **CDAudioSingle** (héritant de **CDAudio** qui elle même hérite de **CompactDisc**). Le but des modifications que nous allons faire est changer le comportement de la fonction membre **afficherGenreMusical()** de la classe **CDAudio** pour que dans la classe **CDAudioSingle** elle ajoute à l'affichage la ligne: ceci est un single.

Ce qui donne: (les classes **CompactDisc et CDAudio** étant inchangées)

Fichier "CDAudioSingle.class.php"

```
<?php
 class CDAudioSingle extends CDAudio {
 function afficherGenreMusical() {
```

```
echo "<BR>Ceci est un single.";
 parent::afficherGenreMusical();
 }
 }
?>
```

Fichier "index.php"

```
<?php
include ("CompactDisc.class.php");
include ("CDAudio.class.php");
include ("CDAudioSingle.class.php");
$cd = new CDAudioSingle();
$cd->titre = "Rock'n PHP - Le Single";
$cd->dateAchat = "25/12/2001";
$cd->genreMusical = "Rock";
$cd->afficherGenreMusical();
?>
```

Ce qui affiche:

```
Ceci est un single.
Le genre musical est: Rock
```

Quelques explications s'imposent. Dans la classe **CDAudioSingle** la fonction membre **afficherGenreMusical()** normalement héritée de **CDAudio** est redéfinie. Cette nouvelle fonction membre affiche "Ceci est un single" puis appelle la fonction membre **afficherGenreMusical()** de la classe de base (ici **CDAudio**) grâce à l'emploi du mot clé **parent** et de l'opérateur **::** (nommé **ORP: Opérateur de Résolution de Portée**).

On a ainsi redéfini une fonction membre qui remplit des fonctions supplémentaires tout en appelant tout de même la fonction correspondante dans la classe de base. L'opérateur **::** peut également être utilisé avec le **nom d'une classe** afin d'appeller une **fonction membre ou une variable membre** dits **statiques** (ie qui ne dépendent d'aucun objet), à partir de l'extérieur de la classe et en l'absence d'un objet existant du type correspondant.

La syntaxe est: **nomDeLaClasse::nomDeLaFonction()**
Ce genre de chose est à éviter de préférence.

V) Polymorphisme

La **notion de polymorphisme** est assez abstraite et assez peu utile en **PHP**. Cette notion découle directement de l'héritage. En observant les classes **CompactDisc** et **CDAudio**, vous constatez que, à cause de l'héritage, la classe **CDAudio** possède toutes les caractéristiques de la classe **CompactDisc**. Ainsi, deux objets respectivement de type **CompactDisc** et **CDAudio** peuvent tous deux être traités comme des objets de type **CompactDisc**:

```
$obj1 = new CompactDisc();
$obj2 = new CDAudio();
//...
$obj1->afficherTitre();
```

```
$obj2->afficherTitre();
```

L'interêt n'est peut être pas évident sur cet example très simple, mais en reprenant le schéma vu précedemment, on constate que de nombreuses classes héritent de **CompactDisc**. Ainsi, il est possible de créer des fonctions (non membres, indépendantes de toutes classes) qui prennent en paramètre des objets de type **CompactDisc** pour effectuer divers traitements (affichage, etc.). Le **langage PHP** n'étant pas typé (vous n'êtes pas obligé, à la création d'une variable, de spécifier le type de données quelle va recevoir), les avantages du polymorphisme sont forcément moins évidents que dans d'autres langages.

VI) Utilité et conclusion

- L'**héritage** permet d'éviter de réécrire des fonctions déjà existantes qu'il suffit de réadapter (voir notre exemple de **CompactDisc** et de toutes ses classes dérivées)
- L'**héritage** permet d'adapter et/ou spécialiser des classes déjà existantes (que vous avez trouvées sur Internet par exemple) afin qu'elles répondent à vos besoins, et cela, sans modifier les classes originales.
- Sur des projets volumineux, une architecture de **classes/héritage** est plus simple à gérer

Nous passons à la notion de sérialisation (pour sauvegarder vos objets) et étudier plusieurs petits points importants (hygiène de code, **programmation par composition, fonctions particulières, mutateurs, accesseurs.)**

7)Classes et objets III

Partie 3: Sérialisation et points importants

Un dernier thème important reste à aborder, c'est **la sérialisation des objets** afin de pouvoir les manipuler simplement. Nous irons alors plus loin avec un petit morceau sur des fonctions parfois utiles (mais facultatives) pour la sérialisation. Un dernier point très important restera les notions de **composition ou héritage** qui doivent vous guider dans l'étape de conception d'un projet.

Prérequis: Les deux premières parties du tutoriel, passage de variables entre deux pages

I) La sérialisation: fonctionnement et intêrets

La **sérialisation** est un moyen simple de sauvegarder vos objets (dans une variable, un fichier, une base sql, etc.) afin de les réutiliser ultérieurement. Vous pouvez notamment passer un objet d'une page à une autre. Pour cela on dispose des deux fonctions suivantes:

```
$chaine = serialize ($objet);
$objet = unserialize ($chaine);
```

La **fonction serialize** permet donc de convertir un objet passé en paramètre en une variable chaine contenant la valeur de chacune des variables membres de l'objet à sérialiser. La **fonction unserialize** effectue l'opération inverse.

La fonction serialize ne sauvegarde en aucun cas les fonctions membres d'un objet. Ainsi, lors de l'appel à la fonction unserialize, la classe correspondante doit être déclarée. On voit ici l'interêt de mettre les classes dans des fichiers séparés.

Créons deux pages: index.php et genre.php basées sur les classes **CompactDisc et CDAudio** définies dans les fichiers respectifs **CompactDisc.class.php** et **CDAudio.class.php**. Sur la page index, une liste de plusieurs **CDAudio** est présentée et chacun d'entre eux possède un lien vers la page genre.php qui se chargera d'afficher le genre musical du **CDAudio** choisis. Pour réaliser cela, le lien contenu dans index.php pointeras vers genre.php en passant en paramètre un objet sérialisé. Tout sera plus clair avec le code suivant:

Fichier "index.php"

```
<?php
 include ("CompactDisc.class.php");
 include ("CDAudio.class.php");
 $cd = new CDAudio();
 $cd->titre = "Rock'n PHP";
 $cd->dateAchat = "25/12/2001";
 $cd->genreMusical = "Rock";
 $cdSerial = serialize($cd);
 $param = urlencode($cdSerial);
 echo "<A HREF='genre.php?var=$param'>Afficher le genre</A>";
?>
```

Fichier "genre.php"

```
<?php
include ("CompactDisc.class.php");
include ("CDAudio.class.php");
$param = stripslashes(urldecode($var));
$cdDeSerial = unserialize ($param);
echo $cdDeSerial->afficherGenreMusical();
?>
```

Ce qui donne:

Exécution de "index.php"

Afficher le genre (inutile de cliquer sur ce lien...)

Après le clic sur le lien
Le genre musical est: Rock

Pour information, après sérialisation de l'objet **$cd** dans index.php, la variable **$cdSerial** reçoit la chaine de caractères:

O:7:"cdaudio":3:{s:5:"titre";s:10:"Rock'n PHP";s:9:"dateAchat";s:10:"25/12/2001";s:12:"genreMusical";s:4:"Rock";}

Vous y reconnaitrez le nom de la classe et les variables membres de l'objet, composées de leur nom et de leur valeur.

Quelques éclaircissements sont nécessaires sur les appels aux fonctions **urlencode(), urldecode()** et **stripslashes().** Dans notre exemple, nous avons passé notre objet, après serialisation, via une url (ie un appel du type **http://domaine.com/index.php?nomDeVariable=valeur).**
Vous devez savoir que tous les caractères ne sont pas permis. Ainsi, il est nécessaire d'utiliser la fonction urlencode pour les remplacer, puis urldecode pour revenir à l'état précédent. La fonction **stripslashes** permet quant à elle de retirer les "\" ajoutés par **urlencode()** et non retirés par urldecode().

Au travers de cet exemple, nous avons vu le fonctionnement principal de la **serialisation**. Notez qu'un objet serialisé peut être enregistré dans un fichier ou dans un champ d'une base **MySQL** (dans ce cas, préferez des **types text** plutot que varchar car vous ne connaissez pas, à priori, la taille de la variable contenant l'objet sérialisé à enregistrer).

II) Fonctions __sleep et __wakeup

Ces deux fonctions, lorsqu'elles sont déclarées comme fonctions membres d'une classe peuvent s'avérer très pratiques. En effet, la **fonction serialize()** cherche si la **fonction __sleep** est définie et l'appelle automatiquement le cas échéant, **avant** d'effectuer la sérialisation. A l'inverse, la **fonction unserialize()** cherche si la **fonction __wakeup()** est définie et l'appelle automatiquement le cas échéant, **après** avoir effectué la déserialisation.

L'intérêt de la fonction **__sleep()** est de permettre **un nettoyage** de l'objet si nécessaire et/ou de fermer la connection à une base de données. La fonction **__wakeup** est principalement

> La fonction **__sleep()** doit impérativement retourner sous forme d'un tableau la liste des variables membres à serialiser. Ceci permet de décider quelles variables doivent être sauvegardées et lesquelles doivent être ignorées (des variables temporaires ou dépendant du contexte, par exemple).

utilisée pour rétablir une connection à une base de données.

Modifions la classe **CDAudio** pour appliquer tout ceci:

Fichier "CDAudio.class.php"

```
<?php
class CDAudio extends CompactDisc {
var $genreMusical ;
function afficherGenreMusical() {
echo "<br>Le genre musical est: ".$this->genreMusical;
 }
function __sleep() {
 // Fermeture d'une connection MySQL:

 mysql_close();
 $ASauver[0] = "titre";
 $ASauver[1] = "dateAchat";
 $ASauver[2] = "genreMusical";
 return $ASauver;
}
```

```
function __wakeup() {
 // Réouverture d'une connection MySQL:
 mysql_connect("serveur", "utilisateur", "motDePasse");
mysql_select_db("base");
 }
 }
?>
```

Dans ce cas, les trois variables membres **$titre, $dateAchat et $genreMusical** seront incluses dans la **sérialisation**. Ainsi, l'appel à **serialize()** appellera **__sleep()** qui se chargera de fermer la connection **MySQL**; l'appel à **unserialize()** appellera **__wakeup()** qui se chargera d'ouvrir une connection **MySQL**.

III) Opérateur de Résolution de Portée (orp) et mot clé parent

L'orp est noté :: et permet d'accéder à une fonction membre dans les cas suivants: En l'absence d'instance de la classe, on peut ainsi accéder à des fonctions membres de la classe (qui ne devraient donc pas dépendre de variables membres). Pour appeller une fonction membre d'une classe de base à partir de la **redéfinition** de celle-ci dans une classe héritée. (exemple ci-dessous)

```
<?php
class Vehicule {
 function acheter() {
 echo "Fonction membre acheter de la classe Vehicule";
}
}
class Voiture extends Vehicule {
function acheter() {
echo "Fonction membre acheter de la classe Voiture";
 // On appelle également celle de la classe de base:
 Vehicule::acheter();
}
 }
?>
```

De la même façon, le mot clé parent associé à l'**orp** permet de désigner génériquement la classe de base à partir d'une classe héritée.
Ainsi, on aurait pu, dans l'exemple précédent, remplacer la ligne

Vehicule::acheter(); par: parent::acheter();

Le mot clé parent, tout comme le mot clé this ne peut bien sur être utilisé qu'à l'intérieur d'une classe. De plus parent ne peut être utilisé que si la classe courante dérive d'une autre classe.

IV) Programmation par composition

Pour compléter ce tour d'horizon, il me semblait nécessaire d'aborder un sujet très important en **programmation objet**. Vous devez être parfaitement conscient de ces idées lorsque vous **concevrez** une **architecture de classes**. **La notion de composition** (il existe de nombreux termes synonymes) est toute simple. Elle correspond à **l'utilisation, en tant que variable membre d'une classe, d'un objet issu d'une autre classe**. Il existe donc une relation entre ces deux classes puisque la première ne peut vivre sans la deuxième (la réciproque étant fausse).

Exemple simple:

```
<?php
class Homme {
var $nom, $prenom;
function Homme ($strNom, $strPrenom) {
// Constructeur de la classe
$this->nom=$strNom;
$this->prenom=$strPrenom;
 }
function afficherIdentite () {
echo $this->nom."<br>".$this->prenom."<br>";
 }
 }
class Livre {
var $titre, $auteur;
function Livre ($strTitre) {
// Constructeur de la classe
 $this->titre = $strTitre;
 $this->auteur = new Homme ("Lerdorf", "Rasmus");
 }
function afficherInfos() {
echo $this->titre."<br>";
$this->auteur->afficherIdentite();
 }
 }
$php = new Livre("PHP Facile");
$php->afficherInfos();
?>
```

Ce qui donne bien sur le **résultat escompté**.

Gardez toujours en tête les notions d'héritage et de composition lors de la conception de vos classes. Néanmoins, ne vous forcez pas à utiliser ces deux systèmes pour chaque classe.

V) Conclusion: le futur

Le futur de la **programmation objet en PHP** est très chargé. Le moteur d'objets de **PHP 5** sera bien plus performant et plus puissant et ce tout en restant très simple d'emploi. A cela, il faudra ajouter **PEAR** qui deviendra un outil très simple d'emploi et permettra de réaliser

encore plus facilement de nombreuses taches. En effet, **PEAR** sera tout simplement une architecture de classes destinées à être utilisées par les programmeurs PHP que vous êtes. D'après les tendances, on peut prédire que **PEAR** deviendra un composant incontournable et essentiel de **PHP** dans les mois qui viennent. Des connaissances en programmation objet (notamment pour se servir des classes de **PEAR** ou les adapter à vos propres besoins par l'héritage) seront absolument nécessaires.

8)Les mises au point

Cette section est là pour appuyer sur certains points importants qu'il est bon de connaître pour bien s'aventurer dans ce merveilleux langage qu'est **PHP**.

Headers already sent : j'ai perdu la tête

Les messages d'erreur **"Cannot modify header information - headers already sent by ..."** et **"Cannot send session cookie - headers already sent by ..."** sont des grands classiques du **développement PHP**, mais beaucoup ne comprennent pas exactement toutes les raisons de ce type d'anomalie.

Qu'est qu'un en-tête ?

En fait, dans le monde du **développement Web**, quand on parle d'en-têtes, on désigne les en-têtes du **protocole HTTP. HTTP,** c'est le langage pratiqué par votre navigateur et le serveur Web. Dans leurs échanges, le navigateur et le serveur Web vont faire précéder leur contenu d'une série d'informations. C'est une sorte d'avant propos que l'on appelle l'en-tête. C'est d'ailleurs un abus de langage que de parler d'en-tête au pluriel car l'en-tête est juste la partie qui précède le contenu et il n'y en a pas plusieurs.

Que contient l'en-tête ?

Eh bien, cela dépend s'il provient du navigateur ou du serveur Web.

Dans le cas du navigateur, il y a quelques informations qui sont obligatoires telles que la page demandée, ce qui est bien utile pour que le serveur sache quel contenu retourner, mais on peut retrouver d'autres informations telles que le type de navigateur, l'url de la page précédente si elle existe, le type d'encodage des caractères supportés, etc.

Si le navigateur dispose de cookies associés au site Web visité, c'est également au travers de l'en-tête qu'il va communiquer cette information.

En ce qui concerne le **serveur Web**, celui-ci va également communiquer différentes informations telles que le type de contenu envoyé, c'est-à-dire **html, texte, image GIF**, animation **Flash, document PDF**, etc. C'est également dans l'en-tête que le serveur Web indique au navigateur le contenu des cookies à enregistrer ou la page à laquelle il doit désormais se rendre.

Comment peut-on voir ces informations d'en-tête ?

Les informations envoyées par le navigateur se retrouvent dans la variable super-globale **$_SERVER** sous des noms préfixés par **HTTP_**, mais vous pouvez aussi consulter ces

données au moyen de la fonction **"apache_request_headers()"** si PHP est utilisé comme **module Apache**.Pour les informations envoyées par le serveur Web, vous disposez de la fonction **"apache_response_headers()",** mais j'ai constaté que celle-ci ne retournait pas toujours toutes les informations d'en-tête. Ainsi, le "content-type" ne figure pas dans cette liste. Si PHP n'est pas installé comme module Apache, cette fonction n'est pas utilisable et il n'est donc pas possible d'obtenir ces informations.

Si vous exécutez le code suivant :

```
<?php
error_reporting(E_ALL);
header('content-type: text/plain', true);
header('Last-Modified: ' . gmdate('D, d M Y H:i:s') . ' GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('X-Mon-En-Tete: ma valeur');

setcookie( 'membre', 'charlie', time() + 100 );

print_r( apache_response_headers() );
echo "\n------------\n";
print_r( apache_request_headers() );
?>
```

Vous obtiendrez une réponse dans le genre :

```
Array
(
   [X-Powered-By] => PHP/5.0.3
   [Last-Modified] => Thu, 31 Mar 2005 16:43:58 GMT
   [Cache-Control] => no-store, no-cache, must-revalidate
   [X-Mon-En-Tete] => ma valeur
   [Set-Cookie] => membre=charlie; expires=Thu, 31-Mar-2005 16:45:38 GMT
)

------------
Array
(
   [Host] => test1
   [User-Agent] => Mozilla/5.0 (Windows; U; Windows NT 5.0; fr-FR; rv:1.7.6)
Gecko/20050223 Firefox/1.0.1
   [Accept] =>
text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/
png,*/*;q=0.5
   [Accept-Language] => fr,fr-fr;q=0.8,en-us;q=0.5,en;q=0.3
   [Accept-Encoding] => gzip,deflate
   [Accept-Charset] => ISO-8859-1,utf-8;q=0.7,*;q=0.7
   [Keep-Alive] => 300
   [Connection] => keep-alive
   [Cache-Control] => max-age=0
)
```

Remarque : Toujours pour les heureux utilisateurs de Firefox, il existe une excellente extension nommée **LiveHTTPHeaders** permettant d'afficher tous les en-têtes échangés entre votre butineur et le site Web.

Et l'erreur headers already sent?

Comme je vous l'ai dit, l'en-tête est envoyé avant le contenu, or le contenu, c'est le texte placé en dehors des balises **PHP** ou généré par des fonctions comme **"echo", "printf", "print", "print_r", "var_dump"** etc.

Vous comprendrez bien que le code suivant pose un problème à **PHP**:

```
<?php
echo 'Mon texte' ;
setcookie('membre', 'charlie', time() + 100 );
?>
```

En effet, l'information sur le cookie est envoyée dans l'en-tête, or du contenu a déjà été généré et vous obtenez la fameuse erreur **"Cannot modify header information - headers already sent"**.

Quand envoyer des informations d'en-tête

Soit en utilisant directement la **fonction header** prévue à cet effet, soit comme nous l'avons vu plus haut, lorsque nous envoyons **des cookies**. Mais il y a un autre cas, c'est au moment de l'appel à la fonction **session_start**. En effet, dans sa configuration par défaut, l'utilisation des sessions entraîne la sauvegarde du **SID** dans un cookie et nous pouvons constater le message d'erreur un peu plus explicite : **« Cannot send session cookie - headers already sent ».**

Si on veut pouvoir utiliser le **système de session**, il est donc impératif de placer la fonction **session_start** avant de générer le moindre contenu.

Cependant, il n'est pas rare de voir des gens s'interroger sur le fait qu'ils n'utilisent pas d' **echo/print**, mais qu'ils obtiennent malgré tout cette erreur. Pourtant, ce message est sans équivoque, ils envoient obligatoirement du contenu et d'ailleurs, s'ils lisaient les messages d'erreur en entier, ils constateraient que **PHP** leur indique l'emplacement exact du contenu perturbateur.

Généralement, ce contenu correspond à du texte placé en dehors des balises PHP, comme ceci :

```
<?php
session_start() ;
?>
```

Comme vous pouvez le constater, il y a un retour à la ligne juste avant la **balise PHP,** ce qui implique l'envoi d'un contenu sous la forme d'un caractère invisible.
Le code le plus pernicieux est sans conteste celui-ci :

```
<?php
require 'config.inc.php' ;
```

```
session_start() ;
?>
```

Il ne semble pas y avoir de caractères invisibles en dehors des **balises PHP**, mais en regardant le message d'erreur avec plus d'attention comme nous devrions le faire plus souvent, nous pouvons lire : **"output started at /www/config.inc.php:27".** Ce message nous indique donc, que l'envoi de contenu a commencé dans le script **config.inc.php** à la **ligne 27** et il y a de fortes chances pour que cette ligne corresponde à la **balise PHP** de fermeture. En observant de plus près, nous allons constater un simple espace juste après cette balise, et cet insignifiant caractère invisible est le grain de sable dans l'engrenage. Dans les fichiers inclus, il ne doit donc jamais y avoir d'espaces ou de retours à la ligne avant et après les balises PHP.

Que faire pour sortir vraiment du texte avant

Il y a des possibilités comme rediriger temporairement la sortie vers la mémoire au moyen de la **fonction ob_start** ou en modifiant la directive de configuration **output_buffering** pour temporiser cette sortie. Mais il faut se poser une autre question : Pourquoi vouloir absolument générer du contenu avant d'envoyer les en-têtes ? La réponse est sans appel car vous n'avez aucune raison de faire une telle chose car cela relève d'une erreur de conception.

Le plus souvent, cela vient de l'utilisation de concepts tel que **les pseudo-frames**, dont je déteste l'appellation :

```
<html>
<head>
<title>Mon super site</title>
<body>
<?php require 'menu.php' ?>
<div id="contenu">
<?php
$page = isset( $_GET['p'] ) ? basename($_GET['p']) : 'accueil';
$page .= '.php';
if( is_readable( $page ) ) require $page;
else require '404.php';
?>
</div>
<?php require 'bas_de_page.php' ?>
</body>
</html>
```

Avec ce genre de fonctionnement, les apprentis développeurs se concentrent sur le script de contenu avec la certitude de toujours avoir un menu et un bas de page.

Seulement, si on veut utiliser la **fonction header,** mettre en place un cookie ou démarrer la session, vous comprendrez vite qu'il y a un problème. On a déjà sorti beaucoup de texte et on cherche à envoyer des informations d'en-tête.

Pourtant ce **type de concept est une formidable opportunité** pour séparer le code de traitement du code d'affichage. Pourquoi ne pas utiliser le code suivant :

```
<?php
// Nom de la page
$page = isset( $_GET['p'] ) ? basename($_GET['p']) : 'defaut';
$page .= '.php';
//On en profite pour adapter le titre qui peut être
// modifié par le traitement
$titre = 'Mon super site';
// Y-a-t-il un traitement à effectuer ?
if( is_readable( 'traite/' . $page ) ) require 'traite/' . $page;
else require 'traite/defaut.php';
?>
<html>
<head>
<title><?php echo $titre ?></title>
<body>
<?php require 'frame/menu.php' ?>
<div id="contenu">
<?php
// on affiche le contenu
if( is_readable( 'frame/' . $page ) ) require 'frame/' . $page;
else require 'frame/404.php';
?>
</div>
<?php require 'frame/bas_de_page.php' ?>
</body>
</html>
```

Certes, il y a désormais deux scripts à éditer par page, mais vous avez une bonne séparation du traitement et de l'affichage. De plus, l'édition d'un script de traitement n'est pas obligatoire, et vous pouvez utiliser le script par défaut pour démarrer par exemple votre session.

En résumé, **headers already sent** indique obligatoirement que vous avez généré un contenu avant l'utilisation d'une fonction générant une information d'en-tête, que ce soit volontairement ou involontairement. De plus, il ne faut pas se dire je n'ai pas le choix, car il y a toujours une alternative, meilleure de surcroît.

Le mot-clef : echo.

Lorsque l'on veut envoyer des données sur le **flux de sortie avec PHP** et que l'on utilise le **mot-clef echo**, plusieurs possibilités s'offrent à nous. Mais quelle forme est la plus performante ?

Quelles sont les différentes formes ?

Il existe en gros quatre formes d'utilisation de **echo** :

1. La forme **brute** :

```
<?php
echo 'la valeur de x est ';
```

```
echo $x;
?>
```

2. La forme **chaîne dynamique** :

```
<?php
echo "la valeur de x est $x";
?>
```

3. La forme **concaténation** (le point) :

```
<?php
echo 'la valeur de x est ' . $x;
?>
```

4. la forme **multi-paramètres** (la virgule) :

```
<?php
echo 'la valeur de x est ', $x;
?>
```

Quelle est la plus performante ?

Certains pourraient être tentés de dire que la forme **chaîne dynamique** est la plus performante, mais ce n'est malheureusement pas le cas. Observons la décomposition faite par la fonction **token_get_all**:

Valeur	Type
<?php\n	T_OPEN_TAG
echo	T_ECHO
	T_WHITESPACE
"	
la	T_STRING
	T_ENCAPSED_AND_WHITESPACE
valeur	T_STRING
	T_ENCAPSED_AND_WHITESPACE
de	T_STRING
	T_ENCAPSED_AND_WHITESPACE
x	T_STRING
	T_ENCAPSED_AND_WHITESPACE
est	T_STRING
	T_ENCAPSED_AND_WHITESPACE
$x	T_VARIABLE
"	
;	
\n	T_WHITESPACE

Valeur	Type
?>	T_CLOSE_TAG

Vous remarquez que tous les mots et espaces de la chaîne de caractères sont séparés.

Si nous décomposons maintenant le code:

Valeur	Type
<?php\n	T_OPEN_TAG
echo	T_ECHO
	T_WHITESPACE
'la valeur de x est '	T_CONSTANT_ENCAPSED_STRING
,	
	T_WHITESPACE
$x	T_VARIABLE
;	
\n	T_WHITESPACE
?>	T_CLOSE_TAG

Nous constatons désormais que la chaîne de caractères est entière ce qui conduit **PHP** à effectuer moins de traitements.

C'est l'une des trois autres, mais laquelle ?

En fait, les **formes brutes et multi-paramètres** sont sémantiquement identiques, mais le code étant un peu plus long pour la forme brute, nous pouvons considérer qu'il y a une très légère différence en faveur de la **forme multi-paramètres.**

Comparons donc les **formes concaténation** et **multi-paramètres**.

La **forme concaténation** consiste à réserver de l'espace mémoire pour accueillir le résultat de la concaténation avant de l'envoyer sur le flux de sortie alors que la forme multi-pramètres envoie chaque chaîne de caractères successivement sur le flux de sortie sans allouer d'espace mémoire inutilement. On pourrait donc s'attendre à ce que la forme multi-pramètres, en plus de consommer moins de mémoire, soit plus performante, mais ce serait aller bien vite en conclusion.

En effet, un autre phénomène entre en jeu : **l'envoi sur le flux de sortie**. En réalité, cette opération est gourmande en ressources et de multiples appels engendrent un ralentissement non négligeable. De ce fait, la concaténation consomme plus de mémoire, mais elle est **beaucoup** plus performante que les multi-paramètres.

Utilisons donc systématiquement la concaténation.

Nous faisons tous une erreur grave depuis le début : enchaîner les **echo** dans nos scripts. Effectivement, il y a fort à parier que vos scripts ne contiennent pas qu'un seul **echo** et ces appels multiples engendrent le ralentissement dont nous avons parlé précédemment.

Alors quelle est la solution ?

La première solution qui viendrait à l'esprit serait de concaténer toutes les chaînes de caractères dans une variable globale que l'on envoie sur le flux de sortie à la fin du script, mais ce n'est pas nécessaire. En effet, le résultat peut être obtenu de façon identique en utilisant la fonction **ob_start** qui place les données de sortie dans un tampon et les envoie lors de l'appel à **ob_end_flush** ou lorsque le script se termine. Si vous reprenez vos scripts et que vous ajoutez cette fonction avant le premier **echo**, vous allez constater un gain de performance **fort appréciable**.

Mais dans ce cas, la forme multi-paramètres devient plus performante.

C'est exact, et le gain est d'environ 10% par rapport à la version "concaténation".

Pour conclure correctement cette fois-ci, je vous conseille d'utiliser **ob_start** au début de vos scripts et d'utiliser la forme **multi-paramètres** de **echo.**

Remarque : Vous pouvez combiner **ob_start** avec **ob_gzhandler** afin de faire d'une pierre deux coups : Accélérer le temps de génération et compresser le flux de sortie pour diminuer le temps de chargement de votre page par l'utilisateur.

Require ou include ?

Pour inclure un script dans un autre script, on peut utiliser les mots clefs **require** ou **include.** Lequel doit-on utiliser ? Peut-être connaissez-vous déjà la réponse, mais je constate souvent que certains les utilisent indifféremment sans trop vraiment se poser de questions.

Petit retour en arrière.

Avant la version 4.0.2, la différence entre les deux était très importante. En effet, le mot clef **require** pratiquait l'inclusion du script **au moment de l'analyse** et non de l'exécution.

Regardons le **script suivant** de plus près :

```
<?php
function affiche_index( $chemin ) {
require $chemin . 'index.html';
}
echo 'début';
affiche_index('cache/');
?>
```

Ce code n'affiche pas **début** et génère une erreur fatale indiquant l'impossibilité d'ouvrir **index.html.** En fait, au moment de l'analyse, **PHP** a tenté d'ouvrir immédiatement le fichier, mais comme le contenu de la variable **$chemin** n'existe pas encore, il l'a considéré comme vide. Ainsi, en utilisant le mot clef **include** à la place de **require**, **PHP** n'effectue cette opération qu'au moment même de son utilisation. **L'avantage du require** est donc la performance car cette inclusion n'est faite qu'une seule fois, même si elle est placée dans une boucle de plusieurs millions d'itérations, mais en contrepartie, vous ne pouvez pas l'utiliser si le chemin n'est pas complètement défini.

Le code suivant sera correct :

```
<?php
$chemin = 'cache/';
require $chemin . 'index.html'
?>
```

En effet, le contenu de **$chemin** est fixé au moment de l'analyse car l'affectation est une valeur constante. En fait, cette différence **a disparu** à la sortie de la version 4.0.2 et l'inclusion s'effectue désormais **au moment** de l'exécution.

Mais une petite nuance existe et elle fait toute la différence. Tandis que **require** génère une erreur fatale en interrompant brutalement le script, **include** ne fait que générer un avertissement puis poursuit l'exécution.

Remarque : En fait, **PHP** génère deux erreurs, la première (avertissement) indiquant que le fichier n'existe pas et la seconde (fatale ou avertissement) indiquant que **l'inclusion n'a pas pu être effectuée.** Il est donc recommandé **dans la très grande majorité des cas** d'utiliser **require** afin d'interrompre le script lorsqu'un fichier est nécessaire au fonctionnement de l'application et d'utiliser **include** uniquement dans les cas où le script inclus peut-être absent. Bien évidemment dans le cas d'**include**, il faut gérer le cas où le script n'existe pas.

Valeur de retour

require et **include** retourne la valeur entière **1** lorsqu'on les appelle, et retourne le booléen **false** en cas d'erreur. En fait, il n'y a que **include** qui peut retourner **false** car **require** arrête le script avant même de faire le retour.

Remarque : Vous pouvez utiliser le mot clef return dans le script inclus comme pour une fonction et dans ce cas, la valeur de retour de **require** ou de **include** sera modifiée en conséquence. Vous pouvez d'ailleurs retourner **false** pour faire croire que l'inclusion n'a pas fonctionnée correctement.

Tester l'existence ou gérer l'erreur ? On pourrait être tenté de gérer les inclusions de deux manières différentes :

En testant l'existence du fichier à inclure :

```
<?php
if( file_exists( 'premier.txt' ) ) require 'premier.txt';
else require 'deuxième.txt';
?>
```

En **exploitant l'optimisation** du **ou** :

```
<?php
@include 'premier.txt' or require 'deuxième.txt';
?>
```

Si on compare les **performances des deux solutions**, on s'aperçoit que la première est bien plus performante que la seconde, ce qui n'est pas une mauvaise chose car la première solution est bien plus claire en termes de logique. L'avantage de la deuxième forme (outre sa concision), c'est sa gestion d'une valeur de retour **false** générée par le fichier inclus. Dans les deux solutions, j'ai bien pris soin d'utiliser **include** uniquement lorsqu'il était **vraiment** nécessaire.