Cours PHP

Versions 4.x et 5.x

Sébastien ROHAUT

Table des matières

1Introduction

1.1Définition

PHP est un langage de script HTML exécuté du côté du serveur. Il veut dire « PHP : Hypertext Preprocessor ». Sa syntaxe est largement inspirée du langage C, de Java et de Perl, avec des améliorations spécifiques. Le but du langage est d'écrire rapidement des pages HTML dynamiques.

1.2Historique

L'origine de PHP remonte à 1995 quand Rasmus Lerdorf a créé PHP/FI, une librairie de scripts Perl utilisés pour la publication de son CV sur son site personnel. Au fur et à mesure des évolutions, la librairie a été portée en C et agrémentée de nouvelles fonctionnalités pour créer des pages dynamiques simples pour le web et accéder à quelques sources de données. PHP/FI signifie Personal Home Page/Forms Interpreter.

PHP/FI 2.0 sort en 1997, toujours développé par une seule personne. C'est le succès : 50000 sites (1% des noms de domaines) disent l'utiliser ou le supporter. Les contributeurs affluent.

PHP 3.0 sort en juin 1998, c'est la première version développée conjointement par Rasmus Lerdorf, Andi Gutmans et Zeev Suraski et entièrement réécrite (les versions précédentes étaient trop lentes pour une application commerciale). Outre les performances, PHP 3 est modulaire et extensible, permettant de lui greffer des API supplémentaires, ce qui n'a pas manqué avec le support de nombreuses bases de données, de formats et de protocoles, une syntaxe plus cohérente et un support basique de l'objet. Il a été installé sur 10% du parc mondial des serveurs web avec des dizaines de illiers de développeurs et des centaines de milliers de sites.

Le moteur de PHP 4 a été repensé afin d'en améliorer les performances pour des applications complexes et d'augmenter encore la modularité. Un nouveau moteur a vu le jour, appelé Zend Engine (contraction de Zeev et Andi). PHP 4 sort officiellement en mai 2000 et apporte des performances accrues, le supports de plusieurs serveurs web, les sessions, une sécurité accrue. C'est actuellement la dernière version stable (nous ne comptons pas les 4.1, 4.2 et 4.3).

Et PHP 5 ? Actuellement en version beta, un cycle qui peut durer plusieurs mois, le but de PHP 5 est une amélioration des performances du moteur Zend (Zend Engine 2), un modèle objet étendu et très proche du C++, une compatibilité accrue avec de nouveaux standards et les anciens (RPC, XML, .NET, ...)

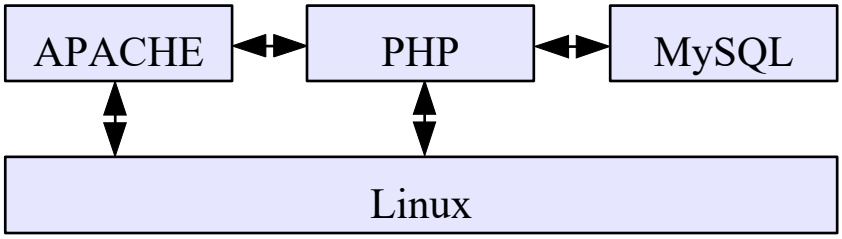
1.3Utilisation pratique

- Forums et Messageries
- Commerce électronique
- Banque / Comptes en ligne
- Publication en ligne
- Moteurs de recherche
- Tout ce que vous voulez, (sauf les jeux)

Résumons quelques sigles que nous allons utiliser par la suite.

- **HTML** : Hypertext Markup Language
- **PHP** : People Hate Perl... Non : PHP, Hypertext PreProcessor

- **SQL** : Structured Query Language
- **MySQL** : serveur de base de données et les outils pour y accéder
- **LAMP** : Linux – Apache – MySQL – PHP, le quatuor gagnant des serveurs web.



1.4Pages statiques vs pages dynamiques

Une page statique et une page écrite directement en HTML. Elle peut éventuellement incorporer du code Javascript lui donnant un semblant de 'dynamisme' mais uniquement du côté du navigateur et des données locales.

Pour des traitements plus lourds nécessitant l'accès à une base de données, un formatage de tableau en fonction de résultats, une recherche poussée, du graphisme, il faut passer par des pages dynamiques et par un langage qui sera exécuté du côté du serveur : ASP sur les serveurs Microsoft/IIS, Perl, PHP...

1.5Scripts CGI

PHP peut être utilisé comme langage CGI. Pour rappel, un script ou progra e CGI est un progra e comme un autre mais dont la sortie est dédiée à un navigateur, par exemple un progra e shell, C ou Perl qui sort du HTML. PHP étant un langage script comme un autre on peut l'utiliser dans ce cas, mais ce n'est pas forcément le mieux.

1.6Pages dynamiques et PHP

PHP est un langage Server Side ou côté serveur. Lors du chargement d'une page PHP, c'est le serveur qui va lire, interpréter et exécuter le code. Puis il renvoie le résultat, généralement sous la forme de code HTML au navigateur. Ainsi le navigateur et l'utilisateur ne voient jamais le véritable code PHP exécuté. De plus le résultat étant une page web classique en HTML, pas besoin d'installer sur le client des composants spécifiques (java, ...). Il n'y a donc pas de notion de vitesse d'exécution du côté du client, mais celle du serveur est prépondérante.

1.7Pourquoi préférer PHP à d'autres langages comme ASP ?

ASP n'est pas entièrement un langage puisque ASP veut dire Active Server Page. ASP dispose bien entendu de ses propres API mais qui peuvent être accessibles via un autre langage. Il s'agit d'utiliser un langage (quelconque) pour créer des pages dynamiques. Nativement disponible avec IIS sous Windows, il ne l'est pas par défaut avec les autres systèmes ou serveurs web comme Apache. Il existe cependant des modules de compatibilité ASP pour Apache comme Apache::ASP qui reprend les API ASP en les étendant et en progra ation PERL. Sous Windows, le code ASP se fait généralement en VBScript ou en Jscript. C'est aussi possible sous Unix/Linux et Apache, mais il faut alors se tourner vers des solutions commerciales (Sun One ASP, Instant ASP, ...). Sous

Windows ASP et VBScript sont utilisés pour l'accès aux ActiveX, composants bien pratiques mais véritables gruyères pour la sécurité. Il n'existe pas à ce jour de portage de ActiveX sur d'autres systèmes que Windows.

PHP est disponible pour toutes les plate-formes et permet de faire la même chose, sinon plus, que ASP et VBScript. N'oublions pas que PHP dispose de plus de 2700 APIs !

1.8Ce que vous devez déjà connaître

Vous devez connaître HTML ainsi qu'un langage de progra ation procédural (C par exemple). Une connaissance basique du SQL peut être nécessaire mais sera abordée.

1.9Le nécessaire serveur

PHP ne sert pas seulement à faire des pages dynamiques. C'est un langage interprété qui au même titre que Perl, Python ou TCL est capable de lancer des scripts intéractifs ou non. On peut même utiliser PHP pour créer des interfaces graphiques (extension GTK). Le minimum nécessaire et vital pour apprendre PHP est donc l'interpréteur PHP lui-même sur un environnement supporté (Unix, Windows, Mac, ...).

Si on parle de LAMP, on voit bien la référence. Un système Linux, un serveur Web Apache, un serveur de bases de données MySQL et un interpréteur PHP. La version 4 de PHP permet de fonctionner avec IIS, et MySQL existe aussi pour Windows. On peut donc utiliser une architecture WIMP (Windows, IIS, MySQL, PHP) ou WAMP (Windows, Apache, MySQL, PHP). Ajouter PHP à IIS, c'est laisser la porte ouverte à tous les vents, on préférera donc la seconde solution.

Les distributions Linux disposent déjà du nécessaire pour l'architecture LAMP et le paramétrage est assez simple. De plus le mode « user_mod » de Apache permet à chaque utilisateur de créer son site web dans un répertoire spécifique de son répertoire personnel.

Sous Windows, c'est déjà un peu plus complexe à mettre en oeuvre manuellement. On emploie donc des distributions toutes faites, la meilleure se nommant EasyPHP. Les versions 1.6 et 1.7 sont les eilleures pour développer. On peut ainsi développer son site sous Windows et le porter tel quel sur un serveur Unix.

1.10Le nécessaire client

Pour développer il suffit d'un simple éditeur mais il vaut mieux préférer un éditeur plus évolué supportant la coloration syntaxique et quelques fonctions évoluées. L'éditeur HAPEdit 3 est parfait pour développer en PHP. L'accès aux bases MySQL peut se faire soit depuis un navigateur (phpMyAdmin) soit depuis une interface MySQLCC ou encore DBDesigner pour la conception.

Pour les tests : un simple navigateur respectant les standards du web.

1.11Le respect des standards

Le W3C est l'organisme international faisant loi dans le monde des standards du web. Il a défini une série de normes dont le HTML, le XML, le XHTML, les CSS, etc. Pourquoi respecter un standard ? C'est la garantie d'un bon fonctionnement et du bon affichage de manière identique de vos pages sur

tous les navigateurs supportant ce standard.

Avec une bonne déclaration (entête HTML) les navigateurs IE6, Mozilla 1.5, Opera 7, Galeon, Konqueror, etc, sont compatibles avec ces standards.

L'utilisation de balises propriétaires à un navigateur (blink par exemple) ou de vbscript (sous IE) est à proscrire. La dernière norme HTML valide est HTML 4.01 STRICT. XHTML qui à terme remplacera définitivement HTML est aussi bien supporté. La dernière norme CSS et la 2. Cependant dans ce cas préférez CSS-1 car seul Opera supporte CSS-2.

Prenez l'habitude quand c'est possible de tester votre site sur deux ou trois navigateurs : IE6 car c'est le plus répandu, Opera car il est extrêmement respectueux des standards et Mozilla car c'est LE challenger et probablement remplaçant à terme de IE (qui n'aura plus d'évolution de version avant LongHorn).

2Bases :Présentation

PHP est langage très souple prenant ses sources dans divers langages comme le C, le Perl, le C++. Il est donc possible d'avoir plusieurs styles de scripts (progra ation classique dite procédurale ou progra ation objet, ou progra ation bordélique). Cette souplesse permet une très grande liberté, un peu comme en Perl. L'inconvénient est qu'on peut très vite obtenir du code illisible (bordélique), même si ça marche très bien. Prenez donc l'habitude de commenter votre code, de l'indenter et de ne placer qu'une instruction par ligne.

2.0.1Syntaxe de base

2.1Intégration à HTML

Une page php porte l'extension « `.php` ». Une page PHP peut être entièrement progra ée en PHP ou mélangée avec du code html. PHP est un langage « Embedded HTML », c'est à dire qu'il apparaît à n'importe quel endroit de la page HTML. Pour ça on le place dans des balises particulières : `<?php` et `?>`. On peut aussi utiliser les balises `<script language="php">` et `</script>`. La première forme est préférable pour plus de simplicité et une compatibilité XHTML. On écrit donc une page HTML dans laquelle on intègre du code PHP.

```
<html>
<head>
<title>Titre</title>
</head>
<body>
<?php
echo "Hello World !";
?>
</body>
</html>
```

Le code HTML généré sera le suivant

```
<html>
<head>
<title>Titre</title>
</head>
<body>
Hello World !
</body>
</html>
```

L'utilisation de balises pour l'intégration de code dans une page web est très souple et permet de jongler facilement avec du code PHP et du code HTML :

```
<?php
  if ( expression ) {
?>
  <strong>Ceci est vrai.</strong>
<?php
  } else {
?>
<strong>Ceci est faux.</strong>
<?php
  }
?>
```

2.2Séparateur d'instructions

Comme en C une instruction se termine par un point-virgule « ; ». Notez que la balise fermante ?> implique la fin d'une instruction.

```
<?php
    echo "Ceci est un test";
?>
<?php echo "Ceci est un test" ?>
```

2.3Bloc d'instructions

Un bloc d'instructions se place entre accolades { et }. Un bloc d'instructions peut contenir du code de n'importe quelle longueur et est considéré dans le reste du code comme une instruction unique. Si c'est une expression (qui a une valeur) on peut donc assigner le bloc, faire des calculs, ...

2.4Commentaires

Les commentaires s'utilisent comme en C et en C++ avec `/* .. */` et `//`. Notez qu'une balise fermante ferme le commentaire en cours.

```
<?php
/* echo "salut !" */
// Commentaire sur cette ligne
?>
```

3Travailler avec des variables

3.1Déclarer une variable

Une variable commence par un dollar « $ » suivi d'un nom de variable. Les variables ne sont pas typées au moment de leur création. Attention PHP est sensible à la casse : var et Var ne sont pas les êmes variables ! Voici les règles à respecter :

- Une variable peut commencer par une lettre
- Une variable peut commencer par un souligné (underscore) « _ »
- **Une variable ne doit pas commencer par un chiffre.**

```
// Déclaration et règles
$var=1; // $var est à 1
$Var=2; // $ Var est à 2
$_toto='Salut'; // Ok
$3petitscochons=5; // Invalide : commence par un chiffre
```

Leur type dépend de leur valeur et de leur contexte d'utilisation. Mais on peut forcer (cast) ponctuellement une variable à un type de données, ce qui s'appelle le transtypage. De même comme le type de variable peut changer en fonction de son utilisation ou du contexte, PHP effectue automatiquement un transtypage, ce qui peut parfois fournir des résultats surprenants. On affecte une valeur à une variable avec le signe égal « = » avec ou sans espace avant ou après.

```
// Déclaration et transtypage
$var='2'; // Une chaîne 2
$var+=1; // $var est maintenant un entier 3
$var=$var+0.3; // $var est maintenant un réel de type double 3.3
$var=5 + "3 petits cochons"; // $var est un entier qui vaut 8
```

Par défaut les variables sont assignées par valeur : la valeur assignée à la variable est recopiée dans la variable. PHP peut aussi travailler par référence. Une variable peut donc référencer une autre variable. On dit alors que la variable devient un alias, ou pointe sur une autre variable. On assigne par référence en utilisant le signe « & » devant la variable assignée

```
$var=2;
$ref=&$var; // $ref devient une référence de $var
echo $ref; // affiche 2
$var=3;
echo $ref; // affiche 3
$ref=4;
echo $var; // affiche 4
```

Attention à la valeur constante NULL insensible à la casse. Affecter une valeur NULL à une variable signifie ne pas puis affecter de valeur.

3.2Portée des variables

La portée d'une variable dépend du contexte. Une variable déclarée dans un script et hors d'une fonction est globale mais par défaut sa portée est limitée au script courant, ainsi qu'au code éventuellement inclus (include, require) et n'est pas accessible dans les fonctions ou d'autres scripts.

```
$a=1; // globale par défaut
 unction foo() {
 echo $a; // c'est une variable locale à la fonction : n'affiche rien
}
```

Pour accéder à une variable globale dans une fonction, il faut utiliser le mot-clé `global`.

```
$a=1; // globale par défaut
$b=2; // idem
 unction foo() {
 global $a,$b; // on récupère les variables globales
 $b=$a+$b;
}
echo $b; // affiche 3
```

PHP accepte les variables statiques. Comme en C une variable statique ne perd pas sa valeur quand on sort d'une fonction.

```
 unction test_static() {
 static $a=0;
 echo $a; // +1 à chaque passage dans la fonction
 $a++;
}
```

3.3Variables prédéfinies

PHP dispose d'un grand nombre de variables prédéfinies. Ces variables sont généralement de type scalaires ou des tableaux. Elles sont souvent de type superglobales, c'est à dire accessible depuis n'importe où sans notion de portée. Voici quelques tableaux prédéfinis (voir au point Tableaux pour comprendre leur utilisation).

- $_GLOBALS : tableau des variables globales. La clé est le nom de la variable.
- $_SERVER : variables fournies par le serveur web, par exemple 'SERVER_NAME'
- $_GET : variables fournies par HTTP par la méthode GET (formulaires)
- $_POST : idem mais pour la méthode POST
- $_COOKIE : les variables fournies par un cookie
- $_FILES : variables sur le téléchargement d'un fichier (upload)
- $_ENV : accès aux variables d'environnement du serveur
- $_SESSION : les variables de session (voir cours sur les sessions)

Note : avant la version 4.2.0 de PHP, les variables passées par la méthode GET, POST (formulaires et URL) et ainsi de suite étaient automatiquement créées dans le script de destination.

http://www.toto.fr/page.php?action=enr créait automatiquement la variable $action qui prenait la valeur 'enr'.

<input type="hidden" name="cache" value="toto"> créait dans la page cible (action du form) une variable $cache de valeur "toto".

Ce n'est plus automatique depuis PHP 4.2.0. Il faut passer par les tableaux superglobaux ci-dessus. On peut cependant passer outre en modifiant la valeur register_globals dans le php.ini.

3.4Variables dynamiques

Une variable dynamique utilise la valeur d'une variable comme nom d'une autre variable. On utilise les variables dynamiques en rajoutant un « $ » devant le nom de la première variable.

```
$a="var";
$$a=1; // $$a=1 equivaut en fait à $var=1
echo $a; // affiche var
echo $$a; // affiche 1
echo $var; // affiche 1
```

Attention avec les tableaux ! Pour éviter toute ambiguïté, il est préférable de placer la variable entre accolades.

3.5Types de variables

3.5.1booléens

Un booléen peut prendre deux valeurs `TRUE` ou `FALSE`. Les deux constantes `TRUE` et `FALSE` peuvent être utilisées sans aucune distinction de casse (pas de différences entre les majuscules et les inuscules).

```
$var=FALSE; // FALSE, False, false, ...
$var2=True; // TRUE, True, true, ...
```

Tous les types peuvent être convertis en booléens. Voici les cas où une variable retournera FALSE en booléen suivant le type :

- Booléen : FALSE
- Entier : 0 (zéro)
- Nombre flottant : 0.0 (zéro)
- Chaîne : chaîne vide "" ou "0" (zéro)
- Tableau : tableau vide sans aucun élément
- Objet : objet vide sans aucun élément
- Constante spéciale NULL

Dans tous les autres cas, la valeur retournée est TRUE. Attention : -1 est considéré comme vrai donc TRUE. Comme en C, les tests de conditions dans les structures de contrôles effectuent une convertion booléenne du la condition.

```
if($var==true) echo "ok";
if($var) echo "ok"; // Identique
```

3.5.2Entiers

Un entier est l'ensemble des nombres naturels, c'est à dire dans virgule, positifs ou négatifs. Les entiers sont généralement codés sur 32 bits mais cela dépend de l'architecture. Si on affecte un nombre entier qui dépasse la capacité de la variable, celle-ci sera transformée en réel (float). Enfin il n'y a pas de notion d'entier non signé.

Lors de la conversion d'un booléen en entier, FALSE devient 0 et TRUE devient 1. Lors de la conversion d'un nombre à virgule flottante, le nombre sera arrondi à la valeur inférieure s'il est

positif, ou supérieure s'il est négatif (conversion vers zéro). Pour la conversion depuis les chaînes, voir à ce type.

3.5.3Virgule flottante

On parte ici des nombres réels, double ou float, c'est à dire les nombres à virgules. La virgule est spécifiée par le point « . ». La puissance de 10 s'exprime avec le « e » ou le « E ».

```
$var=1.234;
$var2=1.1e4; // 1.1 * 10^4 : 11000
```

3.5.4Chaînes de caractères

Une chaîne est une séquence de caractères. PHP travaille en ASCII soit 256 caractères, mais ne supporte pas encore le format Unicode, prévu dans la version 5. Il n'y a pas de limite théorique pour la taille de la chaîne.

On distingue trois syntaxes pour utiliser un chaîne

- Les guillemets simples '...' (apostrophes) : Comme en shell, tous les caractères inclus dans la chaîne sont sortis tels quels sans interprétation. Si vous devez afficher un guillemet simple, il faudra l'échapper : \'
- Les guillemets doubles "..." : Certaines séquences de caractères sont interprétées et les variables sont substituées (remplacées par leur valeur).
- HereDoc : Identique aux HereScripts en Shell. Le texte saisi jusqu'à un délimiteur spécifié est placé dans la variable.

Voici le tableau issu de la documentation PHP des séquences pouvant être utilisés avec les guillemets doubles.

Séquence	**Valeur**
`\n`	Nouvelle ligne (linefeed, LF ou 0x0A (10) en ASCII)
`\r`	Retour à la ligne (`carriage return`, CR ou 0x0D (13) en ASCII)
`\t`	Tabulation horizontale (HT ou 0x09 (9) en ASCII)
`\\`	Antislash
`\$`	Caractère $
`\"`	Guillemets doubles
`\[0-7]{1,3}`	Une séquence de caractères qui permet de rechercher un nombre en notation octale.
`\x[0-9A-Fa-]{1,2}`	Une séquence de caractères qui permet de rechercher un nombre en notation hexadécimale.

```
echo 'Attention l\'abus d\'alcool est dangereux';
$var=2345;
echo "la valeur de \$var est $var\n";
$toto= <<<FIN
```

```
Salut les amis
Comment ça va ?
FIN;
echo $toto;
```

N'importe quelle variable peut être affichée dans une chaîne comme ci-dessus mais attention si deux variables ont un nom proche ! Il faut alors utiliser les accolades { } comme en shell. Enfin on peut accéder à n'importe quel caractère dans une chaîne en indiquant sa position (qui commence à zéro) entre deux accolades collées juste après le nom de la variable.

```
$fic='toto.fic';
$fics='riri fifi loulou';
echo "$fics ${fic}s"; // affiche riri fifi loulou toto.fics
echo "$fic{3}"; // affiche o
```

On peut facilement concaténer deux chaînes avec l'opérateur point « . ». On peut ajouter du texte à une chaîne avec l'opérateur point égal « .= ».

```
$str="Salut les Amis !\n";
$str.="Comment ça va ?"; // "Salut les Amis !\nComment ça va ?
$str2=$str."\n"; // "Salut les Amis !\nComment ça va ?\n
```

Si vous devez utiliser une chaîne de caractères comme valeur numérique (dans une addition par exemple, attention à son contenu. La chaîne sera de type double (réel) si elle contient un 'e' ou un 'E'. Sinon ce sera un entier. La valeur numérique est ensuite définie par le début de la chaîne. Si la chaîne commence par une valeur numérique, elle sera utilisée, sinon elle sera égale à 0 zéro. Si la première expression est une chaîne, le type de variable dépend de la seconde expression.

```
$val=10+"2.55"; // float, 12.55
$val=1+"toto2"; // 1 + 0 = 1
$val=2+"3 petits cochons"; // 2 + 3 = 5 (le 3 est en premier dans la chaîne)
```

3.5.5Les tableaux

Un tableau PHP est une association ordonnée. Une association fait correspondre des valeurs à des clés. Les tableaux sont très souples, ils peuvent avoir de multiples dimensions, être indexés par une clé numérique ou texte, être utilisés comme table de hachage, une pile, une queue, ... Une valeur de tableau peut être elle-même un tableau, pour créer des arbres par exemple.

Un tableau est créé avec la fonction `array()` qui prend comme arguments des paires « `key => value` » séparées par des virgules. La clé peut être soit un entier soit du texte. Attention, 8 est un entier, 08 une chaîne ! Si la clé est absente alors c'est la dernière clé entière plus 1 qui est choisie. Si c'est la première, c'est 0 zéro.

On accède aux éléments d'un tableau à l'aide des crochets « [et] ». On place entre ces crochets la clé entière ou la chaîne.

```
$var=array(10,15,17,23,9);
echo $var[0]; // 10
echo $var[3]; // 17
$tab=array("a"=>12,"nom"=>"toto","pipo",17,4=>5);
echo $tab[0]; // pipo
echo $tab[1]; // 17
echo $tab['a']; // 12
echo $tab['nom']; // toto
```

L'utilisation de la fonction array n'est pas obligatoire et on peut déclarer un tableau à la volée.

```
$tab2[1]=2;
$tab2[]=6; // equivaut $tab2[2]=6
$tab2['test']='Ma chaîne';
```

On peut aussi créer des tableaux multidimensionnels à l'aide des deux méthodes précédentes.

```
$tab=array("un"=>array("riri",1=>"fifi",2=>'loulou'),2=>array(1,2,3),array('un',
'deux','trois'));
echo $tab['un'][0]; // riri
echo $tab[2][1]; // 2
echo $tab[3][2]; // trois
$tab2['un']['deux']='test'; // créé un tableau à deux dimensions
```

Il existe une fonction très pratique pour lister le contenu d'un tableau, ou pour modifier les éléments : `oreach()`.

```
$tab=array(1=>'un',2=>'deux',3=>'trois');
 oreach($tab as $valeur) {
 echo "$valeur \n"; // affiche un deux trois
}
 oreach($tab as $cle => $valeur) {
 echo "$cle => $valeur\n"; // affiche 1 => un, 2 => deux, 3 => trois
}
```

3.5.6La variable objet

Les objets se créent avec l'instruction `class`. Pour créer une instace de l'objet il faut utiliser le mot clé `new`.

```
class test  {
 function affiche_hello() {
  echo "Hello !";
 }
}
$obj=new test;
$obj->affiche_hello();
```

3.6Les constantes

Les constantes est un nom qui prend une valeur ne pouvant pas être modifiée une fois fixée. Une constante n'est accessible qu'en lecture seule. Elles sont sensibles à la casse et doivent par convention être écrites en majuscules.

On définit une constante avec la fonction `define()` et doit respecter certaines règles :

- une constante ne commence pas par un $
- une constante est accessible depuis n'importe quel endroit du code
- une constante ne peut pas être redéfinie
- une constante ne peut contenir d'un scalaire (entier, booléen, chaîne, double).

```
define(CONSTANTE,"valeur");
echo CONSTANTE; // affiche "valeur"
```

3.7Obtenir le type d'une variable

Pour obtenir le type d'une variable, on utilise la fonction « **gettype** » qui retourne une chaîne de texte indiquant le type. **Mais attention rien ne garantit ue le résultat soit le même d'une version PHP à une autre.**

Les types retournés sont "boolean", "integer", "double" (pour des raisons historiques, "double" est retournée lorsqu'une valeur de type float est fournie, au lieu de la chaîne "float"), "string", "array", "object", "resource" (depuis PHP 4), "NULL" (depuis PHP 4), "unknown type"

Si vous souhaitez réellement tester le type d'une variable, il est préférable d'utiliser les fonctions de type « **is_*** » : **is_array, is_bool, is_double, is_float, is_int, is_integer, is_long, is_null, is_numeric, is_object, is_real, is_resource, is_string, is_callable (est-ce une fonction).**

3.8Définir et supprimer une variable

Si vous souhaitez savoir si une variable est définie, c'est à dire si elle est affectée, on utilise « **isset** ».

Enfin si vous souhaitez supprimer une variable, il faut utiliser « **unset** ».

4Les opérateurs

4.1La précédence des opérateurs

C'est l'ordre dans lequel les valeurs doivent être analysées. Ainsi l'expression 4 + 3 * 7 n'est pas lue de gauche à droite et ne retourne pas 49 mais 25. Voici le tableau des priorités des opérateurs par ordre croissant de priorité :

Associativité	**Opérateurs**
gauche	,
gauche	or
gauche	xor
gauche	and
droite	print
gauche	= += -= *= /= .= %= &= f= g= h= <<=>>=
gauche	? :
gauche	ff
gauche	&&
gauche	f
gauche	g
gauche	&
non-associative	== != === !==
non-associative	< <= > >=
gauche	<< >>
gauche	+ - .
gauche	* / %
droite	! h ++ -- (int) (double) (string) (array) (object) i
droite	[
non-associative	new

4.2Opérateurs arithmétiques

Les opérateurs +, -, *, / et %. Le « % » est le modulo : le reste de la division.

4.3Opérateurs d'assignation

Le principal est le = mais on a aussi comme en C des opérateurs combinés +=, -=, *=, /=, %=, .= ...

4.4Opérateurs sur les bits

Les opérateurs sont le & (AND), f (OR), g (XOR), h (NOT, h$a), >> ($a>>$b décalage de $b bits

sur la gauche) et << ($a << $b décalage de $b bits sur la droite). Un décalage de bits sur la gauche équivaut à une multiplication par deux, un décalage sur la droite à une division par deux.

4.5Opérateurs de comparaison

Les opérateurs sont == ($a==$b, même valeur), === ($a===$b, même valeur et même type), != ou <> (différent), <, >, <=, >=.

Il y a aussi l'opérateur ternaire « ?: » expr1?expr2:expr3 Si expr1 est vrai alors expr2 sinon expr3.

4.6Opérateur d'erreur

On dispose d'un opérateur spécial i qui appliqué à une expression empêche la sortie d'un message d'erreur en cas de problème. On peut toujours récupérer le message d'erreur éventuel à l'aide de la variable $php_errormsg mais uniquement si l'option « track_errors » est à « On » dans le php.ini.

```
$retour=@$tab['toto']; // ne retourne pas d'erreurs si l'index toto n'existe pas
```

4.7Opérateur d'exécution

On peut exécuter des commandes externes au PHP comme en Shell avec les opérateurs « guillemts inverses « j » (altgr+6). Attention l'option « safe_mode » doit être à « On » dans le php.ini. On peut aussi utiliser la fonction « shell_exec » qui fait exécuter une commande par le shell.

4.8Opérateurs d'incrémentation/décrémentation

On dispose comme en C des opérateurs ++ et --, à utiliser avant ou après le nom de variable.

```
$a++; // retourne $a puis l'incrémente de 1
++$a; // incrémente $a de 1 puis retourne $a
$a--; // retourne $a puis décrémente de 1
--$a; // décrémente $a de 1 puis retourne $a
```

Attention ! Les opérateurs réagissent aux types de variables. Le PHP réagit comme en PERL. Ainsi :

```
$a='Z';
$a++;
echo $a; // retourne AA
$a++;
echo $a; // retourne AB
```

4.9Opérateurs logiques

Les opérateurs logiques sont :

```
« and » ou « && » ($a and $b, $a && $b) vrai si $a et $b sont vrais
« or » ou « || » ($a or $b, $a || $b) vrai si $a ou $b sont vrais
« xor » ($a xor $b) vrai si $a ou $b sont vrais mais pas les deux en même temps
« ! » (!$a) vrai si $a est faux.
```

Attention, and et or n'ont pas la même priorité (priorité plus faible) que && et ff !

4.10Opérateurs de chaînes

Il y a deux opérateurs de chaînes : le « . » qui concatène deux chaînes entre elles et le « .= » déjà vu qui est l'opérateur d'assignation.

```
$a="Bonjour";
$b=$a." les amis"; // $b contient Bonjour les amis
$b.="! Salut."; // $b contient Bonjour les amis! Salut.
```

4.11Opérateur de tableaux

On peut « additionner » deux tableaux entre eux avec le « + » : le tableau de droite est ajouté au tableau de gauche.

La notion d'expression

En PHP, une expression peut être résumée en « tout ce qui a une valeur ». Ceci dit, on remarque vite que en PHP tout ou presque est une expression. Une variable ou une constante se voient affectés des valeurs. Cette valeur est donc l'expression de la variable ou de la constante.

Nous pouvons résumer en disant qu'une expression représente tout ce qui peut être évalué. On ne peut évaluer que les valeurs...

Une fonction peut aussi être une expression si elle retourne une valeur. On dit généralement qu'une fonction a retourné telle valeur. Une comparaison est aussi une expression : elle retourne une valeur booléenne. Un calcul est aussi une expression, puisqu'une valeur est retournée, affectée, calculée.

PHP est orienté expression ce qui fait que tout ou presque en PHP est une expression. PHP cherche donc à décomposer tout ce qui est possible en expressions à évaluer, pour optimiser ses traitements. Voici quelques exemples.

```
$a=2;
```

On distingue au premier coup d'oeil deux valeurs : la variable $a et la valeur 2, la variable $a étant ise à jour avec la valeur 2. Or en interne il y a une troisième valeur qui rentre en jeu: la valeur de l'assignation, ici elle-même assignée à 2. Cette ligne signifie en fait : « $a=2 est une expression qui vaut 2 ».

```
$b=$a=2;
```

Les assignations sont analysées de droite à gauche. Ici nous avons l'expression $a=2 qui vaut donc 2. $b vaut donc 2 aussi. C'est plus rapide à écrire et à exécuter que $a=2; $b=2;

```
$a=$a+3;
```

PHP évalue d'abord l'expression $a+3 soit la valeur de $a incrémentée de 3, et assigne la nouvelle valeur à $a.

```
$a+=3;
```

Le résultat est le même, mais il est plus rapide car $a n'est évalué qu'une seule fois. On peut en arriver à avoir des expressions surprenantes :

```
$a=2;
$b=$a+=3;
```

La variable $a est incrémentée de 3 et vaut donc 5. L'expression $a+=3 vaut 5. $b se voit assigné la valeur 5. L'expression vaut donc 5.

Dernière chose :

```
$a?$b:$c
```

si l'expression $a est vrai, l'expression $b est évaluée, sinon l'expression $c est évaluée.

```
$a=TRUE; $b=2;$c=3;
echo $a?$b:$c; // affiche 2
```

6 Les structures de contrôle

6.1if

```
if(expression) commande ou { bloc de commandes }
else commande ou { bloc de commandes }
```

Il y a aussi le « elseif », combinaison du if et du else. Le elseif en un mot peut aussi s'écrire en deux ots : le résultat est le même. On peut écrire des elseif en chaîne. Le premier dont l'expression est vrai est exécuté.

```
If(expression) commande ou { bloc de commandes }
elsif(expression) commande ou { bloc de commandes }
elsif(expression) commande ou { bloc de commandes }
...
```

On peut placer du HTML comme commande ou dans le bloc de commande.

```
<?php if ($a == 5) { ?>
A = 5
<?php } ?>
```

On peut aussi utiliser une syntaxe alternative : on ouvre le bloc (juste après le if, le else ou le elseif) avec les « : » deux points, et on ferme l'instruction avec « endif ».

```
<?php
if ($a == 5):
    print "a = 5";
    print "...";
elseif ($a == 6):
    print "a = 6";
    print "!!!";
else:
    print "a ne vaut ni 5 ni 6";
endif;
?>
```

6.2while

6.2.1while classique

C'est la boucle « tant que » simple : tant que la condition n'est pas vraie, on continue la boucle. L'expression est placée en début de boucle : si l'expression est fausse avant de rentrer dans la boucle, la boucle n'est pas exécutée.

```
While(expression) commande ou { bloc de commandes }
```

On peut aussi utiliser la syntaxe alternative :

```
while(expression): commande ou { bloc de commandes }
endwhile
```

6.2.2do ... while

C'est la seconde possibilité. Dans ce cas la commande ou le bloc de commande est exécutée au

oins une fois, car l'expression conditionnelle est testée en fin de boucle.

```
do { bloc de commandes } while(expression)
```

6.3for

Le « for » du PHP est identique au « for » du C.

```
 or(expr1;expr2;expr3) commande ou { bloc de commandes }
```

« expr1 » est exécutée à la première entrée dans la boucle. « expr2 » est exécutée à chaque début d'itération jusqu'à ce que l'expression soit fausse auquel cas on sort de la boucle. « expr3 » est exécutée à la fin de l'itération.

L'usage habituel d'une telle boucle est de placer l'état initial en expr1, la condition de sortie en expr2 et le calcul en expr3. Mais on peu effectuer toutes sortes de choses.

```
// de 1 à 10
 or ($i = 1; $i <= 10; print $i, $i++)
// infini
 or(;;)
// de 1 à 10
 or ($i = 1; $i <= 10; print $i, $i++) ;
```

On peut employer une syntaxe alternative avec le « : » et « endfor ».

```
 or(expr1;expr2;expr3): commande ou { bloc de commandes }
endfor
```

6.4foreach

La boucle « foreach » est peut-être l'une des plus intéressantes pour la manipulation de tableaux ou de résultats de requêtes SQL. Elle permet de lister les tableaux. Elle dispose de deux syntaxes.

```
 oreach(array_expression as $value) commandes
 oreach(array_expression as $key => $value) commandes
```

La première syntaxe récupère les éléments du tableau un par un, séquentiellement. La valeur de l'élément courant du tableau est placée dans $value.

La seconde syntaxe est presque identique, sauf qu'en plus la clé (l'index) de l'élément actuel est placée dans $key.

Attention : modifier la valeur de $value (ou de $key) ne modifie pas le tableau car cette boucle travaille sur une copie, pas une référence. Par contre dans le second cas, comme on dispose de la clé, rien n'empêche d'assigner quoi que ce soit à l'élément courant.

Remarque : un appel à foreach « rembobine » automatiquement le tableau à son premier élément. Mais pas dans les autres boucles, il faut alors utiliser « reset ».

```
reset($arr);
while (list(, $value) = each ($arr)) {
 echo "Valeur: $value<br>\n";
}
```

```
 oreach ($arr as $value) {
 echo "Valeur: $value<br>\n";
}
$a = array (
    "un" => 1,
    "deux" => 2,
    "trois" => 3,
    "dix-sept" => 17
);

 oreach($a as $k => $v) {
    print "\$a[$k] => $v.\n";
}
```

6.5break et continue

L'instruction « break » permet de sortir d'un for, while, foreach ou switch. On peut lui indiquer de combien de structures on souhaite sortir si elles sont emboîtées.

L'instruction « continue » permet de passer à l'itération suivante. Attention PHP conside le switch comme une boucle, et dans ce cas, réévalue le switch. On peut indiquer à continue combien de structures emboîtées relancer.

6.6switch

Le « switch » est équivalent à une série de if et permet de comparer avec un grand nombre de valeurs.

```
switch ($i) {
    case 0:
        print "i egale 0";
        break;
    case 1:
        print "i egale 1";
        break;
    case 2:
        print "i egale 2";
        break;
    default:
        print "i est inférieur à 0 ou supérieur à 2 »;
}
```

Le switch s'arrête à la première expression case vraie puis exécute le code suivant dans l'ordre indiqué, jusqu'à la première instruction break. S'il n'y a pas de break, tout le code jusqu'à la fin du switch est exécuté. Dans l'exemple suivant, si $i vaut 0, tous les print seront affichés !

```
switch ($i) {
    case 0:
        print "i egale 0";
    case 1:
        print "i egale 1";
    case 2:
        print "i egale 2";
}
```

Notez aussi que le default doit intervenir en dernier, sinon il n'a aucun intérêt.

Enfin on peut employer une syntaxe alternative avec « : » et « endswitch ».

```
switch ($i):
    case 0:
        print "i egale 0";
        break;
    case 1:
        print "i egale 1";
        break;
endswitch
```

6.7return

Contrairement à d'autres langages, « return » n'est pas une fonction mais une instruction. Dans une fonction, return sert à sortir de celle-ci et à retourner une valeur. Dans un script, elle sort de celui-ci. Attention cependant dans les scripts inclus (voir require et include) : le return dans ce type de code considère qu'il sort de la fonction « require » ou « include » et donc ne sort pas du script ou de la fonction dans lesquels ce code a été inclus !

Comme return est un élément du langage et pas une fonction il y a pas besoin d'utiliser les parenthèses.

6.8require et include (_once)

« require » et « include » incluent à l'endroit actuel et exécutent le fichier PHP. Ils sont identiques dans leur fonctionnement à une exception : le traitement des erreurs. Un include produit un « warning » (le code continue en principe à s'exécuter) tandis qu'un require produit une « erreur fatale » (l'exécution s'arrête).

Comme require et include sont des éléments du langage et pas des fonctions il y a pas besoin d'utiliser les parenthèses.

« require_once » et « include_once » ressemblent à leurs homologues avec cependant une différence. Quoi qu'il arrive, le fichier est inclus une seule fois. Si un second « require_once » apparaît avec le même fichier, l'instruction ne sera pas exécutée.

7 Les fonctions

7.1Syntaxe et portée

Voici la syntaxe d'une fonction.

```
 unction fonc($arg1, $arg2, $arg3, ..., $argn) {
    bloc de commandes
    return $valeur
}
```

Une fonction n'a pas besoin d'être déclarée avant d'être utilisée (sauf si vous voulez rester compatible avec PHP3), du moment qu'elle apparaît dans le script.

Il y a cependant deux exceptions : si la fonction est conditionnelle, c'est à dire déclarée dans un if, ou si la fonction est déclarée dans une autre fonction. Dans le premier cas elle sera inconnue du reste du code tant que la condition du if n'est pas vraie. Dans le second cas elle ne sera connue que quand la première fonction sera exécutée. Voici un exemple tiré du manuel.

Pour le premier cas :

```
$makefoo = true;

/* Impossible d'appeler foo() ici, car cette fonction n'existe pas.
   Mais nous pouvons utiliser bar() */
bar();

if ($makefoo) {
  function foo () {
    echo "Je n'existe pas tant que le programme n'est pas passé ici.\n";
  }
}

/* Maitenant, nous pouvons appeler foo() car $makefoo est maintenant vrai */
if ($makefoo) foo();

 unction bar() {
  echo "J'existe dès le début du programme.\n";
}
```

Pour le second cas :

```
 unction foo() {
  function bar() {
    echo "Je n'existe pas tant que foo() n'est pas appelé.\n";
  }
}

/* Impossible d'appeler bar() ici car il n'existe pas. */
 oo();

/* Maintenant, nous pouvons appeler bar(), car l'utilisation de foo() l'a rendu
   accessible. */
bar();
```

7.2Arguments

On peut passer autant d'arguments que souhaité à une fonction PHP, de tous types. Les arguments

sont séparés par des virgules. Par défaut ils sont passés par copie.

```
 unction affiche($msg) {
 echo $msg;
}
```

Il est possible de passer des arguments par référence, on peut donc en changer la valeur à l'intérieur de la fonction. Pour cela on rajoute un « & » devant l'argument. On peut passer ponctuellement une variable par référence en plaçant un « & » devant son nom lors de l'appel de la fonction.

```
 unction modif(&$msg) {
 $msg="coucou";
}
$msg="salut";
modif($msg);
echo $msg; // coucou
```

On peut aussi définir des valeurs par défaut mais ATTENTION, dans la liste des paramètres les valeurs par défaut doivent être en dernier.

```
 unction affiche($nom, $prenom="toto") {
 echo "$nom $prenom";
}
```

Enfin on peut utiliser un nombre variable d'arguments avec les fonctions func_num_args, func_get_arg et func_get_args.

```
 unction foo() {
 $numargs = func_num_args();
 echo "Nombre d'arguments: $numargs\n";
}
 oo (1, 2, 3);    // affiche 'Nombre d'arguments: 3'
```

7.3Valeur de retour

On retourne une valeur avec l'instruction « return ». On peut retourner tout type de valeurs, mais pas plusieurs en même temps. Cependant rien n'empêche dans ce cas de retourner un tableau. Une fonction peut aussi retourner une référence. Dans ce cas elle doit être déclarée avec un « & » devant son nom.

```
 unction carre ($num) {
   return $num * $num;
}
echo carre (4);  // affiche '16'
```

7.4Fonctions internes

En plus des fonctions définies par l'utilisateur PHP propose un grand nombre de fonctions internes couvrant une multitude de domaines. Pour accéder à ces fonctions il faut simplement les appeler par leur nom et que le module correspondant soit compilé avec PHP. Nous verrons dans les chapitres suivants comment en utiliser certaines.

8 Gestion des formulaires en PHP

8.1GET et POST

Le but est de récupérer le contenu des champs d'un formulaire HTML dans notre code PHP pour pouvoir le traiter. Lorsqu'un formulaire est envoyé à un script PHP, toutes les variables seront disponibles automatiquement dans le script.

Les formulaires peuvent être de type GET ou POST. Pour rappel, dans un formulaire de type GET, les informations sont passées directement par l'URL en clair, ce qui peut poser des problèmes de limitations suivant le serveur (de 256 à 8192 octets selon le cas). La méthode POST n'a pas ce genre de limitation, car les informations sont transmises par le conteneur de variables globales (dans l'entête) et sont de plus cachées. PHP peut gérer les deux méthodes de manière transparente.

8.2Récupération par tableau

Chaque champ de formulaire en PHP est défini par un nom et une valeur. Dans un script, PHP va récupérer ces noms et ces valeurs dans des tableaux spéciaux dit superglobaux (accessibles depuis partout). Pour la méthode GET, le tableau est $_GET, pour la méthode POST le tableau est $_POST. Si vous ne souhaitez pas vous soucier de la méthode, vous pouvez utiliser le tableau $_REQUEST. En index on aura le nom du champ de formulaire (ou de la variable passée en URL) et en valeur la valeur du champ. Par exemple :

```
<form action="foo.php" method="post">
Name: <input type="text" name="username"><br>
Email: <input type="text" name="email"><br>
<input type="submit" name="submit" value="Submit me!">
</form>
```

Dans la page PHP foo.php on aura :

```
<?php
echo $_POST['username'];
echo $_REQUEST['email'];
?>
```

Imaginons l'appel d'une page test.php par une URL comme ceci :

http://www.monsite.com/test.php?id=1

Ici on transmet une variable via une URL et donc la méthode implicite GET. Pour récupérer « id » dans un code PHP on peut donc faire :

```
<?php
echo $_GET['id'];
echo $_REQUEST['id'];
?>
```

$_GET ne contiendra que les variables de type GET. $_POST ne contiendra que les variables de type POST. $_REQUEST contient les variables de type POST et GET mais aussi les variables de cookies. Il est aussi possible de procéder autrement en récupérant le nom du champ directement en tant que variable sans passer par un tableau. **Pour cela il faut vérifier dans le fichier php.ini ue la valeur register_globals est à on**. Dans ce cas les noms des champs de formulaire seront les

noms des variables :

```
<?php
echo $username;
echo $email;
?>
```

8.3Récupération directe

Si cette valeur est à off et que pour diverses raisons (sécurité, pas le droit, etc) vous ne souhaitez/pouvez pas la modifier, il existe tout de même une méthode. La fonction « **import_reuest_variables** » permet d'effectuer explicitement la même chose. Cette fonction prend un ou deux paramètres, le second étant optionnel. Après exécution, on récupère les variables sans passer par un tableau. Le premier paramètre est le type de variables qu'on souhaite récupérer :

- P / p : variables de type POST
- G / g : variables de type GET
- C / c : variables de type cookie

On peut cumuler l'ensemble mais attention à l'ordre ! Le second paramètre est le préfixe qu'on souhaite donner au nom des variables qu'on va récupérer. Voici deux exemples :

```
<?php
import_request_variables("p");
echo $username;
?>

<?php
import_request_variables("P","Post_");
echo $Post_username;
?>
```

On peut aussi passer par la fonction « extract » qui permet d'exporter un tableau en variables. Ainsi pour le tableau $_POST :

```
<?php
extract($_POST);
echo $username;
?>
```

8.4Utiliser des tableaux

Les formulaires permettent d'utiliser des listes à choix multiples et PHP doit donc récupérer l'ensemble des choix de l'utilisateur. Pour ça il faut utiliser les tableaux.

```
<form action="<?php echo $HTTP_SERVER_VARS['PHP_SELF']; ?>" method="post">
Name: <input type="text" name="personal[name]"><br>
Email: <input type="text" name="personal[email]"><br>
Beer: <br>
<select multiple name="vin[]">
<option value="bordeaux">bordeaux</option>
<option value="beaujolais">beaujolais</option>
<option value="loire">loire</option>
</select><br>
```

```
<input type="hidden" name="action" value="submitted">
<input type="submit" name="submit" value="submit me!">
</form>
```

Dans l'exemple précédent si par exemple on sélectionne plusieurs vins (ex : bordeaux et beaujolais) on aura dans $vin[0] bordeaux et $vin[1] beaujolais.

9 Date et heure

9.1Localisation

Le format de date et d'heure (format et langue) est dépendant de l'environnement d'exécution du code PHP. Généralement la plupart des serveurs web incorporant PHP sont configurés en langue anglaise avec une gestion des informations de localisation en anglais. Il peut être préférable de passer en français pour un site francophone ce qui simplifie largement la tkche. On peut effectuer cela avec la fonction « **setlocale** ». Voici deux exemples : un sous NT/2000 et un autre sous Linux.

```
if (substr(php_uname(), 0, 7) != "Windows") {
    setlocale(LC_TIME,"fr_FR@euro");
} else {
    setlocale(LC_TIME,"fr");
}
```

Dans cet exemple, nous vérifions d'abord à quel système nous avons à faire (fonction php_uname) puis en fonction du résultat nous passons en localisation française selon deux méthodes. Après ce changement, les appels aux fonctions retournant des jours de la semaine ou des mois le feront en français.

9.2Récupérer la date et l'heure

9.2.1date()

Le plus simple est d'utiliser la fonction « **date** » qui retourne l'heure locale sous forme d'une chaîne de caractères à préciser en premier paramètre. En second paramètre on peut préciser un « **timestamp** » si on ne souhaite pas l'heure actuelle. Pour récupérer la date du jour :

```
echo date("d/m/Y"); // 06/01/2004
```

Voici les caractères utilisables (documentation PHP) :

- a - "am" (matin) ou "pm" (après-midi)
- A - "AM" (matin) ou "PM" (après-midi)
- B - Heure Internet Swatch
- d - Jour du mois, sur deux chiffres (éventuellement avec un zéro) : "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)
- g - Heure, au format 12h, sans les zéros initiaux i.e. "1" à "12"
- G - Heure, au format 24h, sans les zéros initiaux i.e. "0" à "23"
- h - Heure, au format 12h, "01" à "12"
- H - heure, au format 24h, "00" à "23"
- i - Minutes; "00" à "59"
- I (i majuscule) - "1" si l'heure d'été est activée, "0" si l'heure d'hiver .
- 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")
- - Mois; i.e. "01" à "12"
- M - Mois, en trois lettres (et en anglais) : par exemple "Apr" (pour Avril)
- n - Mois sans les zéros initiaux; i.e. "1" à "12"
- O - Différence d'heures avec l'heure de Greenwich, exprimée en heures; i.e. "+0200"
- r - Format de date RFC 822; i.e. "Thu, 21 Dec 2000 16:01:07 +0200" (ajouté en PHP 4.0.4)
- s - Secondes; i.e. "00" à "59"
- S - Suffixe ordinal d'un nombre, en anglais, sur deux lettres : i.e. "th", "nd"
- t - Nombre de jours dans le mois donné, i.e. "28" à "31"
- T - Fuseau horaire de la machine ; i.e. "MET"
- U - Secondes depuis une époque
- w - Jour de la semaine, numérique, i.e. "0" (Dimanche) to "6" (Samedi)
- W - Numéro de semaine dans l'année ISO-8601 : les semaines commencent le lundi (ajouté en PHP 4.1.0)
- Y - Année, 4 chiffres; i.e. "1999"
- y - Année, 2 chiffres; i.e. "99"
- z - Jour de l'année; i.e. "0" à "365"
- Z - Décalage horaire en secondes (i.e. "-43200" à "43200")

Tout autre caractère peut être employé et ne sera pas interprété. Si vous désirez employer l'un de ces caractères, il faut le verrouiller avec un antislash. **ATTENTION AVEC CETTE FONCTION LE RESULTAT EST EN ANGLAIS**.

On peut récupérer le timestamp actuel avec la fonction « **time** ». La commande suivante a la même effet que la précédente :

```
echo date("d/m/Y", time()); // 06/01/2004
```

9.2.2date locale strftime()

Pour formater le résultat en local (pour tenir compte de la localisation) il faut utiliser la fonction « **strftime** » qui tient compte de la langue. On aura donc un résultat en français. Le premier paramètre est la chaîne de formatage. Le second est un éventuel timestamp.

- %a - nom abrégé du jour de la semaine (local).
- %A - nom complet du jour de la semaine (local).
- %b - nom abrégé du mois (local).
- %B - nom complet du mois (local).
- %c - représentation préférée pour les dates et heures, en local.
- %C - numéro de siècle (l'année, divisée par 100 et arrondie entre 00 et 99)

- %d - jour du mois en numérique (intervalle 01 à 31)
- %D - identique à %m/%d/%y
- %e - numéro du jour du mois. Les chiffres sont précédés d'un espace (de ' 1' à '31')
- %h - identique à %b
- %H - heure de la journée en numérique, et sur 24-heures (intervalle de 00 à 23)
- %I - heure de la journée en numérique, et sur 12- heures (intervalle 01 à 12)
- %j - jour de l'année, en numérique (intervalle 001 à 366)
- %m - mois en numérique (intervalle 1 à 12)
- %M - minute en numérique
- %n - newline character
- %p - soit jam' ou jpm' en fonction de l'heure absolue, ou en fonction des valeurs enregistrées en local.
- %r - l'heure au format a.m. et p.m.
- %R - l'heure au format 24h
- %S - secondes en numérique
- %t - tabulation
- %T - l'heure actuelle (égal à %H:%M:%S)
- %u - le numéro de jour dans la semaine, de 1 à 7. (1 représente Lundi)
- %U - numéro de semaine dans l'année, en considérant le premier dimanche de l'année comme le premier jour de la première semaine.
- %V - le numéro de semaine comme défini dans l'ISO 8601:1988, sous forme décimale, de 01 à 53. La semaine 1 est la première semaine qui a plus de 4 jours dans l'année courante, et dont Lundi est le premier jour.
- %W - numéro de semaine dans l'année, en considérant le premier lundi de l'année comme le premier jour de la première semaine
- %w - jour de la semaine, numérique, avec Dimanche = 0
- %x - format préféré de représentation de la date sans l'heure
- %X - format préféré de représentation de l'heure sans la date
- %y - l'année, numérique, sur deux chiffres (de 00 à 99)
- %Y - l'année, numérique, sur quatre chiffres
- %Z - fuseau horaire, ou nom ou abréviation
- %% - un caractère j%' littéral

9.2.3Retour par tableau getdate()

La fonction « **getdate** » permet aussi de récupérer la date et l'heure actuels (avec éventuellement un timestamp en paramètre) mais sous forme de tableau indexé selon les valeurs suivantes :

- "seconds" - secondes
- "minutes" - minutes

- "hours" - heures
- "mday" - jour du mois
- "wday" - jour de la semaine, numérique. 0: dimanche jusqu'à 6: samedi
- "mon" - mois, numérique
- "year" - année, numérique
- "yday" - jour de l'année, numérique; i.e. "299"
- "weekday" - jour de la semaine, texte complet (en anglais); i.e. "Friday"
- "month" - mois, texte complet (en anglais); i.e. "January"

```
$aujourdhui = getdate();
$mois = $aujourdhui['month'];
$mjour = $aujourdhui['mday'];
$annee = $aujourdhui['year'];
echo "$mjour/$mois/$annee";
```

9.2.4L'heure localtime()

Pour l'heure on peut utiliser les fonctions « **localtime** » qui peut aussi prendre un paramètre timestamp et qui retourne un tableau indexé selon les valeurs suivantes ;

- "tm_sec" - secondes
- "tm_min" - minutes
- "tm_hour" - heure
- "tm_mday" - jour du mois
- "tm_mon" - mois de l'année
- "tm_year" - Année, incompatible an 2000
- "tm_wday" - Jour de la semaine
- "tm_yday" - Jour de l'année
- "tm_isdst" - Est-ce que l'heure d'hiver a pris effet?

9.3Récupérer n'importe quelle date

On peut récupérer n'importe quelle date en timestamp ou en chaîne. On peut pour cela utiliser la fonction « **strtotime** ». Elle prend comme paramètre une chaîne de texte en anglais qu'elle essaie de convertir en timestamp. Voici quelques exemples.

```
echo strtotime("now"); // timestamp actuel
echo strtotime("-1 day"); // hier
echo strtotime("+1 day"); // demain
echo strtotime("11 september 2001"); // Le 11 septembre 2001
echo strtotime("+1 year +2 weeks +3 days"); // dans 1 an 2 semaines 3 jours
```

10MySQL

10.1Présentation

MySQL est un SGBDR : « Système de Gestion de base de Données Relationnel » qui se définit lui-
même comme étant « La base de données Open Source la plus populaire au monde ». Rien que ça.
Et c'est vrai ! Edité par la société MySQL AB, MySQL est un produit Open Source libre d'utilisation sous licence GPL pour les projets libres. Cependant une licence commerciale est nécessaire dans les autres cas, nota ent si on souhaite redistribuer MySQL avec un produit non libre ou si on souhaite un support technique.

La version de production actuelle de MySQL est la version 4 (4.0.17 à l'écriture de ce support), mais la grande majorité des serveurs des hébergeurs sont encore dans les dernières versions de MySQL 3.23 (3.23.58). La future version actuellement en développement est la 5.0.0 et n'est surtout pas à utiliser en production.

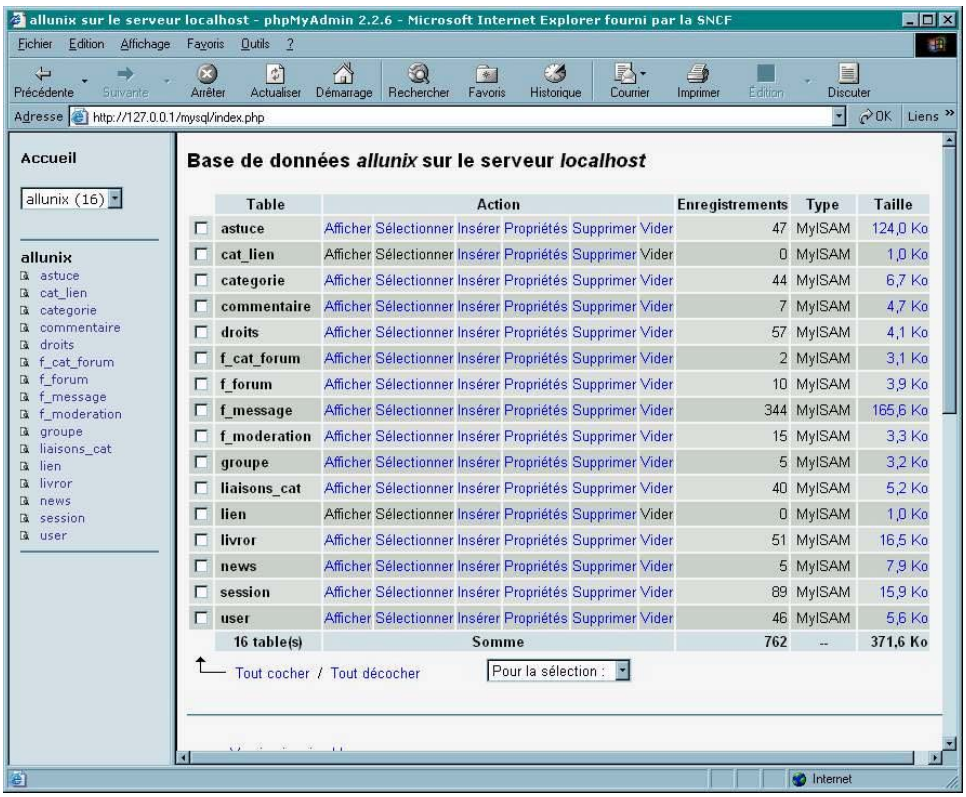
Les principales qualités de MySQL sont sa simplicité et sa rapidité. Son principale défaut est le
anque de fonctionnalités dites avancées (dans les versions précédentes) : clé étrangères,
procédures stockées, triggers et selects imbriqués nota ent. Mais cela ne doit pas occulter sa puissance avec l'ajout de fonctionnalités avancées comme une syntaxe SQL étendue (replace, limit, delete), les index de recherche « fulltext » permettant de créer des moteurs de recherche, ...

La prochaine version stable (5.0) comblera les lacunes des précédentes versions avec le support complet de la syntaxe SQL ANSI-99.

10.2Outils

10.2.1PhpMyAdmin

S'il reste bien entendu possible d'utiliser MySQL en ligne de commande, un peu comme « sqlplus » de Oracle, on est bien plus à l'aise avec un environnement plus intuitif. Ainsi, l'outil phpMyAdmin est une interface web à MySQL permettant d'effectuer la plupart des tkches de maintenance et d'utilisation. Cette solution fonctionne depuis n'importe quel navigateur et est indépendante de la
achine. On accède à phpMyAdmin généralement par l'URL http://server_name/mysql.



10.2.2MysqlCC

MysqlCC (MySQL Control Center) est le front-end graphique officiel de MySQL, développé par la me société. Basé sur e toolkit Qt, il est disponible sous Windows, Unix (linux) et bientôt sur MacOS. Il permet l'administration du serveur, la gestion des bases et tables, l'exécution de requêtes SQL interactives avec coloration syntaxique...

10.3Créer une base

A partir de l'écran d'accueil de phpMyAdmin, on saisit le nom de la base dans « Créer une base de données ». Il faut de préférence choisir un nom simple et intuitif. Puis on clique sur « Créer ». Après la création une nouvelle page s'affiche : c'est la page principale d'administration de la base. En haut seront toujours présents après l'exécution d'une commande les résultats de celle-ci. Cet écran permet nota ent l'exécution de commandes SQL, et le travail sur les tables.

10.4Utilisateurs et droits

Dans le menu de gauche, on choisit le « - » (tiret) qui correspond à la racine ou administration du serveur MySQL par lui-même. Dans la nouvelle page, on choisit « Utilisateurs et privilèges ». La page suivante permet d'ajouter un utilisateur. Il faut faire attention à une chose : le serveur. En effet, suivant la configuration initiale (sur Linux par exemple) les droits dépendent de la façon dont a été accédée la base. Ainsi si localhost et 127.0.0.1 sont identiques, si la base est appelée depuis une autre IP ou un autre nom, l'utilisateur n'aura pas tous les droits. En cas de doute, il suffit de choisir « Tout serveur ».

Pour créer un administrateur, le mieux est de donner tous les privilèges. Pour un utilisateur « standard » (requêtes de sélection, d'ajout, de mises à jour et de suppression d'enregistrements) on choisira les privilèges Select, Insert, Update et Delete.

Une fois l'utilisateur créé, et si plusieurs bases sont présentes, on peut affiner les droits pour chacune des bases. La liste des utilisateurs et de leurs droits sur une base est affiché en haut. En choisissant « modifier » on peut affiner la sélection nota ent en passant par « autres privilèges » : droits sur des bases, tables ou même colonnes.

10.5Créer et modifier des tables

On retourne par le menu de gauche dans une base existante. Dans l'écran d'administration on saisit (bas de la page) le nom de la table et le nombre de champs (colonnes). Il sera possible après coup d'en rajouter ou d'en supprimer. Puis on choisit « Créer ».

L'étape suivante consiste à définir la table. Intéressons-nous aux différents types de champs possibles.

10.5.1Types Numériques

10.5.1.1Types entiers

Pour chaque entier on peut ajouter le mot clé « UNSIGNED » : non signé. Attention au type BIGINT qui a des limitations (voir documentation MySQL). Entre parenthèses : les valeurs UNSIGNED. Il est possible de fournir une précision d'affichage (nombre de chiffres). Par exemple une taille de 2 pour un TINYINT équivaut à une valeur de 0 à 99.

- TINYINT : -128 à 127 (0 à 255)
- SMALLINT : -32768 à 32767 (0 à 65535)
- MEDIUMINT : -8388608 à 8388607 (0 à 16777215)
- INT ou INTEGER : -2147483648 à 2147483647 (0 à 4294967295)
- BIGINT : -9223372036854775808 à 9223372036854775807 (0 à 18446744073709551615)

10.5.1.2Types réels (flottants)

- FLOAT : nombre en virgule flottante
- DOUBLE, DOUBLE PRECISION, REAL : nombre en virgule flottante double-précision.
- DECIMAL, DEC, NUMERIC : spécial, les valeurs sont « unpacked » : c'est une chaîne texte

10.5.1.3Types booléens

- BIT, BOOL, BOOLEAN (depuis 4.1) : Un bit, équivaut à TINYINT(1)

10.5.2Types caractères

Ces types méritent un peu plus d'explications.

10.5.2.1Types CHAR

CHAR : une chaîne de caractère dont la taille peut être de 1 à 255 caractères. La taille est définie à la création de la table. Dans la table, le champ prend effectivement le nombre de caractères indiqué, ême si la taille réelle est inférieure.

VARCHAR : une chaîne de caractère dont la taille maximale peut être de 1 à 255 caractères mais dont la longueur peut être variable. MySQL stocke en plus un octet contenant la longueur de la chaîne. C'est une optimisation, une chaîne de 100 caractères va donc prendre 100+1 octets en place dans la table.

On peut rajouter la définition BINARY. Par défaut les champs CHAR et VARCHAR sont triés sans tenir compte de la casse. L'attribut BINARY permet de trier en tenant compte de la casse selon la achine.

Avec ces champs, les espaces finaux sont supprimés.

10.5.2.2Types TEXT

Les types TEXT et BLOB permettent de stocker des quantités variables de texte selon leurs définitions. Ici « L » est la longueur du texte :

- TINY(BLOB/TEXT) : L+1 octets avec L<2g8
- BLOB/TEXT : L+2 octets avec L<2g16
- MEDIUM(BLOB/TEXT) : L+3 octets avec L<2g24
- LONG(BLOB/TEXT) : L+4 octets avec L<2g32

TEXT et BLOB s'utilisent de manière identique mais il faut respecter les rêgles suivantes :

- Lors de tris, un champ BLOB tient compte de la casse, pas un champ TEXT.
- On ne peut pas faire de recherche FULLTEXT sur un champ de type BLOB. Il faudra utiliser les champs TEXT.
- Les champs de type TEXT et BLOB ne prennent pas de valeurs par défaut.
- Les espaces finaux ne sont pas supprimés.

10.5.2.3Type ENUM

Un champ de type ENUM ne peut contenir qu'une chaîne de texte par plusieurs prédéfinies à la création de la table. Voici un exemple :

```
CREATE TABLE test (
  id int NOT NULL auto_increment,
  c_enum enum('oui','non'),
  PRIMARY KEY  (id)
);
```

Le champ c_enum ne pourra prendre que deux valeurs : oui ou non. Si on tente d'y mettre autre chose :

```
insert into text (c_enum) values('toto');
```

Dans ce cas une chaîne vide est insérée à la place. Si le champ est déclaré NOT NULL, la valeur par défaut devient la première de la liste d'énumération.

10.5.2.4Type SET

Un type set ressemble un peu au type ENUM sauf que le champ peut avoir une ou plusieurs valeurs parmi une liste prédéfinie à la création de la table. Dans le type ENUM une seule valeur est autorisée dans le champ. Avec le type SET, plusieurs valeurs séparées par des espaces sont autorisées.

```
CREATE TABLE test (
  id int(11) NOT NULL auto_increment,
  c_set set('oui','non'),
  PRIMARY KEY  (id)
);
insert into test (c_set) values ('oui,non');
insert into test (c_set) values ('non,oui');
insert into test (c_set) values ('oui);
```

on effectue une recherche d'un élément SET soit avec LIKE, soit avec la fonction FIND_IN_SET :

```
select * from test where c_set LIKE '%oui%';
select * from test where FIND_IN_SET('oui',c_set)>0;
```

10.5.3Types date

Attention avec les dates : la logique de MySQL est de laisser aux développeurs le soin de contrôler la validité de leurs dates. Ainsi MySQL autorisera l'enregistrement d'une date de type 2004-02-30 ! Les seuls tests effectués sont la validité du mois (entre 0 et 12) et la validité du jour (entre 0 et 31). Le zéro (0) est autorisé pour des raisons pratiques: stockage de l'année, ou de l'année/mois seulement par exemple. Voici quelques autres informations :

- MySQL, dans les comparaisons nota ent, essaie « d'interpréter » le format de date qu'on lui passe. C'est le cas pour les comparaisons entre des champs de type date et des champs texte. Dans la plupart des cas le résultat sera celui attendu mais attention tout de même au formalisme.

- Les dates doivent être données de préférence au format année-mois-jour. L'année doit toujours apparaître en premier. Dans le cas d'un année-jour-mois attention : MySQL ne peut pas deviner si 2004-02-05 est le 5 février ou le 2 mai...

- Si une date est utilisée dans un calcul numérique, elle est convertie en nombre (timestamp).

- Si une valeur illégale est transmise, c'est la valeur zéro (0) qui est retenue. La valeur zéro dépend du type de champ.

- La gestion de l'an 2000 se fait ainsi :

    - Dans le cas d'années à 4 chiffres, aucun souci. Il vaut mieux TOUJOURS utiliser des

années sur 4 chiffres.

- Dans le cas d'une année à deux chiffes, 00-69 sont converties en 2000-2069, 70-99 sont converties en 1970-1999.

10.5.3.1Formats supportés

- AAAA-MM-JJ HH:MM:SS ou AA-MM-JJ HH:MM:SS. Les séparateurs peuvent être n'importe quoi : AAAA/MM/JJ HH.MM.SS est valable
- AAAA-MM-JJ ou AA-MM-JJ, même remarque pour les séparateurs.
- AAAAMMJJHHMMSS ou AAMMJJHHMMSS si la chaîne ou le nombre ont une signification en tant que date
- AAAAMMJJ ou AAMMJJ, même remarque

10.5.3.2Type DATETIME

Le type DATETIME permet de stocker la date et l'heure sous le format AAAA-MM-JJ HH:MM:SS. Son zéro est donc 0000-00-00 00:00:00.

10.5.3.3Type DATE

Avec ce type seule la date est stockée au format AAAA-MM-JJ. Son zéro est 0000-00-00.

10.5.3.4Type TIMESTAMP

Le type TIMESTAMP permet de stocker implicitement et automatiquement un timestamp (heure courante) lors d'un enregistrement de données avec INSERT ou UPDATE. Si une table dispose d'un champ TIMESTAMP, lors de l'ajout ou de la mise à jour d'un celui-ci sera automatiquement mis à jour pour cette ligne d'enregistrement. La colonne n'a pas besoin d'être explicitement spécifiée. Si on la spécifie, alors on peut lui mettre n'importe quelle valeur.

```
CREATE TABLE test (
  id int(11) NOT NULL auto_increment,
  c_ts timestamp(14) NOT NULL,
  c_txt varchar(10) NOT NULL default '',
  PRIMARY KEY  (id)
)
insert into test (c_txt) values('toto');

select c_ts from test;

20040112143212

update test set c_txt='truc';

select c_ts from test;

20040112143339
```

On remarque que le résultat retourné par MySQL sur la colonne TIMESTAMP n'est pas une valeur timestamp Unix (bien que telle en interne) mais une valeur formatée. Par défaut un TIMESTAMP

retourne une valeur (numérique) sur 14 caractères. On peut choisir cette taille à la création de la table :

- TIMESTAMP ou TIMESTAMP(14) : AAAAMMJJHHMMSS
- TIMESTAMP(12) : AAMMJJHHMMSS
- TIMESTAMP(10) : AAMMJJHHMM
- TIMESTAMP(8) : AAAAMMJJ
- TIMESTAMP(6) : AAMMJJ
- TIMESTAMP(4) : AAMM
- TIMESTAMP(2) : AA

Avec tous ces formats, spécifiez toujours l'année le mois et le jour même avec un timestamp de 4 ou de 2 sinon la date sera invalide.

10.5.3.5Type TIME

Le type TIME peut être utilisé pour stocker une information horaire. Son format est HH:MM:SS ou HHH:MM:SS et la valeur peut être négative. Les valeurs maximales possibles vont de -838:59:59 à 838:59:59. TIME ne sert pas qu'à stocker une heure de la journée mais un intervalle ou une durée ce qui explique la plage horaire. On y accède comme ceci : D HH:MM:SS.fraction : D est le jour. La fraction n'est pas stockée. Ou encore : HH:MM:SS, HH:MM, D HH:MM:SS, D HH:MM, D HH, D SS, HHMMSS, .SS, MMSS, ...

10.5.3.6Type YEAR

Le type YEAR stocke uniquement l'année, sur une plage allant de 1901 à 2155. L'interprétation des valeurs à deux chiffres et soumise à l'interprétation de la gestion de l'an 2000.

10.5.4Création : Syntaxe SQL

On utilise la commande SQL CREATE.

```
CREATE TABLE [IF NOT EXIST] nom_table
(
champ1 TYPE [NOT NULL| NULL] [DEFAULT valeur_defaut] [AUTO_INCREMENT] [PRIMARY
KEY],
...,
PRIMARY KEY(nom_champ),
KEY [nom] (col1, col2, ...),
INDEX [nom] (col1, col2, ...),
UNIQUE [INDEX] [nom] (col1, col2, ...),
FULTEXT [INDEX] [nom] (col1, col2, ...)
) TYPE=type_table;
```

Notez que

- KEY et INDEX sont des synonymes.
- Un PRIMARY KEY est un index UNIQUE dont toutes les colonnes doivent être NOT NULL
- Un PRIMARY KEY peut contenir plusieurs colonnes. Dans ce cas, PRIMARY KEY ne doit pas apparaître au niveau de la définition d'une colonne mais la syntaxe PRIMARY KEY(cols,...) doit être utilisée
- Il ne peut y avoir qu'un seul index PRIMARY KEY, mais plusieurs index UNIQUE.
- Si une application veut obtenir le PRIMARY KEY d'une table et qu'il n'existe pas, alors MySQL retournera le premier index UNIQUE trouvé.
- Si on ne donne pas de noms à un index, celui-ci est généré automatiquement en prenant le nom de la première colonne de l'index suivi d'un suffixe numérique (col_1, col_2, ...).
- Bien que cela soit déconseillé, on peut établir un index sur une colonne TEXT ou BLOB. Mais dans ce cas il faut obligatoirement spécifier une longueur d'index.

Par exemple :

```
CREATE TABLE f_message (
  id_message int(10) unsigned NOT NULL auto_increment,
  id_msg_initial int(11) unsigned NOT NULL default '0',
  id_msg_prec int(10) unsigned NOT NULL default '0',
  id_forum int(10) NOT NULL default '0',
  login varchar(20) NOT NULL default '',
  date datetime NOT NULL default '0000-00-00 00:00:00',
  sujet varchar(200) NOT NULL default '',
  texte text,
  PRIMARY KEY  (id_message),
  KEY precedent (id_msg_prec),
  KEY login (login),
  KEY forum (id_forum),
  KEY initial (id_msg_initial),
  FULLTEXT KEY ft1 (sujet,texte)
) TYPE=MyISAM;
```

10.5.5Types de tables (MyISAM et InnoDB)

Le Type de table est particulier à MySQL. Le type par défaut est « MyISAM », le gestionnaire de tables originel de MySQL, simple et rapide, correspondant à la majorité des besoins.

Le second type principal est « InnoDB » et permet une gestion des transactions et verrouillage des données. Les « commit », « rollback », restauration après crash, et surtout les contraintes d'intégrité avec clés étrangères sont possibles.

```
CREATE TABLE parent(id INT NOT NULL, PRIMARY KEY (id)) TYPE=INNODB;
CREATE TABLE child(id INT, parent_id INT, INDEX par_ind (parent_id),
             FOREIGN KEY (parent_id) REFERENCES parent(id)
             ON DELETE SET NULL
) TYPE=INNODB;
```

On peut aussi utiliser les clauses « ON DELETE CASCADE » et « ON DELETE SET NULL ». Quelques règles sont à respecter pour l'utilisation :

- Avec un ON DELETE CASCADE, si une ligne de la table parente est supprimée, les lignes correspondantes à la clé étrangère sont automatiquement supprimées dans la table fille. Dans l'exemple, une suppression d'un enregistrement de la table « parent » supprime tous les enregistrements de la table « child » dont le « parent_id » est égal au « id » de l'enregistrement de la table « parent » supprimé.

- Avec un ON DELETE SET NULL, les enregistrements fils ne sont pas supprimés, mais les colonnes de la clé étrangère (ici parent_id) prennent la valeur NULL.

- Un index doit exister pour la clé parente et la clé fille.

10.5.6Index FULLTEXT

10.5.6.1Création

Dans ses dernières versions (3.23.23 et supérieures) MySQL permet l'indexation et la recherche sur un ou plusieurs champs de type TEXT. On peut ainsi créer des moteurs de recherches très rapidement. Pour ça il faut créer un index « **FULLTEXT** » ou en Texte Entier, où chaque mot du champ texte est en fait indexé.

Les index FULLTEXT ne fonctionnent qu'avec des champs de type TEXT (pas BLOB) ou VARCHAR et sur des tables de type MyISAM uniquement. Voici comment créer un index FULLTEXT. Si la table existe déjà :

```
CREATE FULLTEXT INDEX ft1 on f_message (sujet,texte);
```

A la création de la table, on rajoute une ligne de création d'index :

```
FULLTEXT KEY ft1 (sujet,texte)
```

On peut préciser une ou plusieurs colonnes dans ce genre d'index. Mais **ATTENTION** : lors de la recherche si on a créé un index sur deux colonnes, on ne peut pas rechercher sur une seule des deux colonnes en même temps... Dans ce cas il faudra créer plusieurs index FULLTEXT.

10.5.6.2Recherche

On effectue une recherche avec les fonctions **MATCH()** et **AGAINST()**. La fonction MATCH permet de préciser sur quelles colonnes on effectue la recherche, et la fonction AGAINST les mots recherchés.

```
SELECT id_message FROM f_message WHERE MATCH(sujet,texte) AGAINST('linux
nvidia');
```

Cette requête va effectuer une recherche sur les mots 'linux' et 'nvidia' dans les colonnes 'sujet' et 'texte'. Elle va retourner les id des messages correspondants avec les plus pertinents en premier (ceux qui correspondent le mieux à la recherche). Cela est dl au fait que MATCH est dans une clause WHERE.

La pertinence est déterminée par MySQL selon des critères (un peu opaques tout de même) comme le nombre de mots dans la ligne, le nombre de mots uniques dans la ligne, la longueur de la ligne. C'est la fonction MATCH qui retourne la pertinence sous forme d'un flottant supérieur ou égale à

zéro. Le zéro signifie une pertinence nulle (pas de correspondance). Plus la valeur est élevée plus le résultat est pertinent.

```
SELECT id_message, MATCH(sujet,texte) AGAINST('linux nvidia') FROM f_message;
```

Cette requête récupère en seconde colonne la valeur de la pertinence. Celle-ci n'est pas classée car il n'y a pas de WHERE ou de ORDER BY. Voici une dernière méthode permettant de récupérer les champs et la pertinence triés par ordre de pertinence.

```
SELECT id_message,sujet MATCH (sujet,texte) AGAINST ('linux nvidia')
FROM f_messages WHERE MATCH (sujet,texte) AGAINST ('linux nvidia');
```

10.5.6.3Recherche booléenne

Les versions 4.x de MySQL ont introduit la notion de recherche booléenne qui permet d'affiner ses recherches.

```
SELECT id_message FROM f_message WHERE MATCH(sujet,texte) AGAINST ('+linux
+nvidia +ati' IN BOOLEAN MODE);
```

Voici la signification des caractères supplémentaires

- + : le mot doit obligatoirement être présent dans chaque enregistrement retourné
- - : le mot ne doit pas être présent dans les enregistrements retournés
- > et < : valorisation du mot. Un >toto <titi valorise plus toto que titi dans la recherche
- (et) : grouper une série de mots
- h : négation d'un mot. Contrairement au -, le mot est tout de même pris en compte avec une pertinence négative.
- * : tronquage d'un mot : bon* => bonjour, bonsoir ...
- les guillemets « " » : une chaîne recherchée telle quelle.

10.6MySQL et PHP

10.6.1Connexion à une base de données

Deux étapes sont généralement nécessaires pour plus de simplicité. La seconde n'est pas obligatoire ais bien pratique dans le cas où on travaille sur une seule base de données dans toute la page.

10.6.1.1Connexion au serveur

On utilise la fonction **mysl_connect()**. Cette fonction prend (au minimum) trois paramètres : le serveur (hostname) l'utilisateur et son mot de passe.

```
$b_host="sql.tf-data.net"
$b_user="toto";
$b_pass="CreT1";
```

```
$cnx=mysql_connect($b_host, $b_user, $b_pass);
if(!$cnx) die ("erreur de connexion à MySQL");
```

10.6.1.2Choix d'une base

On choisit une base de données par défaut avec la fonction **mysl_select_db()**. Elle prend au inimum un paramètre, le nom de la base. Le second paramètre optionnel est une ressource retournée par mysql_connect, en cas de connexion sur plusieurs serveurs MySQL. Par défaut, la fonction prend la dernière connexion ouverte (celle du dernier mysql_connect exécuté).

```
$b_base="slyunix";
$db=msql_select_db($b_base);
if(!$db) die("Erreur de connexion à la base $b_base");
```

10.6.1.3Fermeture d'une connexion

On ferme une connexion à MySQL avec la fonction **mysl_close()**. Cependant dans le cas de connexion non persistantes (voir à ce point) cette fonction n'est pas obligatoire car PHP ferme automatiquement les connexions à la fin d'un script.

```
mysql_close($cnx);
```

10.6.1.4Séquence complète pour une base

```
$b_host="sql.tf-data.net"
$b_user="toto";
$b_pass="CreT1";
$b_base="slyunix";

mysql_connect($b_host, $b_user, $b_pass) or die("erreur de connexion à MySQL");

mysql_select_db($b_base) or die("erreur à la selection de $b_base);

mysql_close();
```

10.6.2Les requêtes

10.6.2.1Exécuter une requête

On exécute une requête SQL avec la fonction **mysl_query()**. Cette fonction prend au moins un paramètre : une requête SQL sous forme de chaîne. La fonction retourne FALSE en cas d'échec (colonne ou table invalide, droits insuffisants, pas de connexion, etc).

ATTENTION : Ce n'est pas parce que l'appel à mysql_query() n'a pas retourné d'erreur que la fonction retourne des lignes de résultats dans le cas d'un SELECT par exemple. Enfin, la requête SQL ne doit pas finir par un point-virgule.

La requête peut être de n'importe quel type (selection, mise à jour, destruction, etc). Dans le cas d'un SELECT, SHOW, EXPLAIN ou DESCRIBE, mysql_query() retourne une ressource qui sera ensuite utilisée pour lire le résultat.

```
$result=mysql_query("select id_message, sujet from f_message");
```

10.6.2.2Nombre de lignes affectées

Dans le cas d'un DELETE, INSERT, REPLACE ou UPDATE, on peut connaître le nombre de lignes affectées (modifiées) par la requête à l'aide de la fonction **mysl_affected_rows()**.

```
$result=mysql_query("delete from f_message where login='toto'");
if($result) echo mysql_affected_rows()." Enregistrements supprimés";
```

10.6.2.3Nombre de lignes retournées

Dans le cas d'un SELECT, le nombre d'enregistrements (lignes) retourné est obtenu avec la fonction **mysql_num_rows()**.

```
$result=mysql_query("select * from f_message where id_message > 10");
if($result) echo mysql_num_rows().' enregistrements retournés');
```

10.6.2.4Récupérer les résultats

La fonction la plus sympathique pour récupérer les enregistrements après l'exécution d'une sélection est **mysl_fetch_array()**. Elle prend au minimum un paramètre : une ressource résultat (résultat de ysql_query). Elle retourne une ligne de résultat sous forme d'un tableau associatif, d'un tableau indexé ou des deux. Par défaut, le tableau retourné est à la fois associatif et indexé.

Dans un tableau associatif, l'index du tableau est le nom du champ correspondant à la colonne. Dans un tableau indexé, les colonnes sont numérotées à partir de zéro.

Notez que mysql_fetch_array() ne retourne qu'une seule ligne de résultat. Pour passer à la suivante, il faut exécuter la fonction à nouveau. Elle retournera FALSE quand il n'y aura plus de lignes à lire.

```
$result=mysql_query("select sujet,texte from f_message where login='toto'");
$tab=mysql_fetch_array($result); // Première ligne du résultat
echo $tab['sujet'].', '.$tab['texte']; // affiche les champs sujet et texte
echo $tab[0].', '.$tab[1]; // idem
```

Si plusieurs colonnes portent le même nom, la dernière colonne sera prioritaire. Dans une requête affichant des noms de colonnes identiques, le mieux est de les renommer :

```
SELECT t1.nom as col1, t2_nom as col2 FROM t1, t2 ...
```

Le second paramètre de mysql_fetch_array() peut être :

- MYSQL_ASSOC : le résultat est uniquement un tableau associatif (index=nom de colonne)
- MYSQL_NUM : le résultat est uniquement un tableau indexé numériquement.
- MYSQL_BOTH (par défaut) : les deux

Voici comment récupérer tous les résultats :

```
$result=mysql_query("select sujet,texte from f_message where login='toto'");
while ($result && $tab=mysql_fetch_array($result)) {
 echo $tab['sujet'].', '.$tab['texte']; // affiche les champs sujet et texte
}
```

Notez l'expression du while. Pour y rentrer, il faut que le résultat de la requête ($result) ne soit pas FALSE (donc que la requête ait fonctionné) et qu'il reste des lignes à lire. Autrement dit si la requête ne retourne rien, on ne rentre pas dans la boucle.

Notez enfin qu'il existe deux autres fonctions possibles :

- **mysql_fetch_row()** : équivaut à mysql_fetch_array($var,MYSQL_NUM)
- **mysql_fetch_assoc()** : équivaut à mysql_fetch_array($var, MYSQL_ASSOC)

10.6.2.5Insertion avec auto-incrément

L'identifiant unique d'une table est souvent un entier auto-incrémenté. L'avantage est qu'il n'y a pas besoin de gérer cet identifiant, c'est MySQL qui le détermine tout seul. Mais dans certains cas, il peut être nécessaire de récupérer la valeur de ce champ auto-incrémenté après une insertion pour, par exemple, mettre à jour un autre enregistrement (liste chaînée, jointure, etc). On utilise pour ça la fonction **mysl_insert_id()**.

```
$result=mysql_query("insertf fintof     _messagef f(sujet,f ftexte)f fvalues
('test','test')");

$id=mysql_insert_id();
$result=mysql_query("select sujet, texte from f_message where id_message=$id");
$tab=mysql_fetch_array($result);
echo $tab['sujet'].', '.$tab['texte']; // test, test
```

10.6.3Récupération des erreurs

En cas d'erreur lors d'une manipulation sur MySQL, on peut récupérer le numéro de l'erreur MySQL par la fonction **mysl_errno()**. De même on peut récupérer le message d'erreur par la fonction **mysql_error()**.

On peut trouver la liste des codes d'erreurs ici :

http://www.mysql . com/doc/en/Error-returns.html

11Le système de fichiers

11.1Travail sur les fichiers

11.1.1Ouverture

La fonction **fopen()** permet d'ouvrir ou de créer un fichier selon divers modes. Elle prend en premier pramètre le nom du fichier avec son chemin. Ce nom de fichier peut être une URL (il faut pour cela que la directive **allow_url_fopen** soit activée dans le php.ini). Le second paramètre est le mode d'ouverture :

- **r** :lecture seule, en début de fichier
- **r+**: lecture/écriture, en début de fichier
- **w** : écriture seule , taille du fichier à 0 (ancien contenu effacé), créé s'il n'existe pas
- **w+** : lecture/écriture, taille du fichier à 0, créé s'il n'existe pas
- **a** : écriture seule, en fin de fichier, créé s'il n'existe pas
- **a+** : lecture/écriture, en fin de fichier, créé s'il n'existe pas.

On peut sous Windows rajouter la lettre 'b' pour le type binaire, inutile sous Unix.

Enfin, fopen() admet un troisième paramètre : 1. Dans ce cas le fichier à ouvrir sera recherché dans le **include_path** (voir php.ini).

La valeur retournée est une ressource de fichier (identifiant de fichier) ou la valeur FALSE en cas d'échec.

```
$file=fopen('server.log','a+',1);
```

11.1.2Lecture

11.1.2.1fgets()

La fonction **fgets()** permet de lire une ligne d'un fichier en mode texte. La fonction s'arrête lorsqu'elle arrive à la fin du fichier ou à un retour chariot. On peut préciser en second paramètre une longueur de texte en octet qui est par défaut 1024 et ce paramètre est obligatoire avant les versions 4.2. Le premier paramètre est l'identifiant du fichier. FALSE sera retourné en cas d'erreur.
ATTENTION : fgets() ne retourne pas FALSE à la fin du fichier.

```
$ligne=fgets($file);
```

11.1.2.2fread();

La fonction **fread()** permet de lire un fichier en mode binaire. Le second paramètre est la longueur souhaitée. Sous Windows il faut penser à placer 'b' dans fopen(). La lecture s'arrête lorsque les n octets (longueur) ont été lus, la fin du fichier a été atteinte ou qu'une erreur est survenue.

```
$file=fopen('monimage.gif','rb');
$img=fread($file, $filesize('nomimage.gif'));
 close($file);
```

11.1.2.3fscanf()

La fonction **fscanf()** lit des lignes d'un fichier en fonction d'un formatage particulier.

```
$file=fopen('users.txt','r');
while($ligne=fscanf($file,"%s\t%s\t%s\n")) {
 echo $ligne;
}
```

11.1.2.4fgetc()

La fonction **fgetc()** lit un seul caractère dans le fichier. Contrairement à fgets(), elle retourne FALSE à la fin du fichier. Le premier paramètre est l'identifiant du fichier.

11.1.2.5fgetss()

La fonction **fgetss()** fonction est identique à fgets() mais la ligne retournée est débarrassée de ses éventuelles balises PHP ou HTML. On peut préciser en troisième paramètre une liste de balises autorisées.

11.1.2.6fgetcsv()

La fonction fgetcsv() est une fonction très puissante car elle permet de lire un fichier de type CSV, c'est à dire dont les champs sont délimités, pour récupérer les champs dans un tableau. Lesond paramètre est la longueur maximale d'une ligne du fichier, le troisième paramètre le séparateur, qui est la virgule par défaut.

Voici un exemple permettant de lire un fichier CSV et d'en afficher le résultat sous forme d'un tableau HTML :

```
$filename = "book1.csv";
$id = fopen($filename, "r");
while ($data = fgetcsv($id, filesize($filename))) $table[] = $data;
 close($id);

echo "<table>\n";

 oreach($table as $row) {
 echo "<tr>";
 foreach($row as $data) echo "<td>$data</td>";
 echo "</tr>\n";
}

echo "</table>\n";
```

11.1.2.7Renvoyer un fichier par la sortie standard

La fonction **readfile()** prend comme paramètre un nom de fichier et renvoie son contenu par la sortie standard. La fonction **readgzfile()** effectue la même chose mais le fichier est au format compressé gzip (zlib).

11.1.3Ecriture

Les deux fonctions **fwrite()** et **fputs()** sont identiques 100% car **fputs()** est un alias de **fwrite()**. Elles écrivent une chaîne dans un fichier. On peut limiter la longueur de la chaîne en troisième paramètre mais par défaut toute la chaîne est écrite. La valeur retournée est le nombre d'octets écrits, ou FALSE en cas d'erreur.

```
$file=fopen('monfic.txt','a);
$ret=fwrite($file,'Une ligne dans mon fichier');
 close($file);
```

En cas de travail sur un fichier binaire, il ne faut pas oublier sous Windows de rajouter l'option 'b'.

11.1.4Déplacement

11.1.4.1fseek()

La fonction **fseek()** permet de se déplacer dans un fichier. Le second paramètre (offset) est le nombre d'octets de déplacement. Il peut être négatif. Par défaut le déplacement est calculé à partir du début du fichier mais un troisième paramètre peut modifier cet état. La valeur retournée est 0 (zéro) en cas de réussite, -1 sinon. Un positionnement après la fin du fichier n'est pas une erreur.

- SEEK_SET : Calcul à partir du début du fichier. Position finale : Début+offset
- SEEK_CUR : Calcul à partir de l'emplacement actuel. Position finale : Actuelle+offset
- SEEK_END : Calcul à partir de la fin du fichier. Position finale : Fin+Offset.

```
$ret=fseek($file,0, SEEK_END); // Fin du fichier
```

11.1.4.2ftell()

La fonction **fseek()** renvoie la position actuelle du pointeur dans le fichier.

```
$pos=ftell($file);
```

11.1.4.3rewind()

La fonction **rewind()** permet de retourner au début du fichier. Elle retourne TRUE en cas de réussite, FALSE sinon.

11.1.5Fin de fichier

La fonction **feof()** indique si on est à la fin du fichier ou non. Elle retourne TRUE si la fin de fichier a été atteinte, FALSE sinon.

```
$fd = fopen ("/tmp/inputfile.txt", "r");
while (!feof ($fd)) {
 $buffer = fgets($fd, 4096);
 echo $buffer;
}
 close ($fd);
```

11.1.6Fermeture

On ferme un fichier avec la fonction **fclose()**.Elle retourne TRUE en cas de réussite, FALSE sinon.

11.2Travail sur les répertoires

11.2.1Répertoire actuel

La fonction **getcwd()** retourne le nom du dossier de travail courant, c'est à dire là où on est.
Attention sous Windows : cette fonction ne fait pas de distinctions entre les majuscules et les inuscules.

```
$chemin=getcwd();
echo $ chemin ;  :: /srv/www/site
```

11.2.2Se déplacer

On se déplace avec la fonction **chdir()**. L'argument peut être absolu ou relatif. En cas d'erreur, FALSE est retourné.

```
if(chdir('./rep'))
 getcwd();
else
 die('Impossible de se déplacer');
```

11.2.3Ouvrir, lister, fermer

On ouvre un répertoire en lecture avec la fonction **opendir()** qui prend comme unique paramètre un nom ou chemin de répertoire. En cas d'échec, la fonction retourne FALSE sinon un identifiant (handler) de répertoire.

On lit une entrée avec la fonction **readdir()** qui prend comme paramètre l'identifiant de répertoire. Quand il n'y a plus rien à lire, la fonction retourne FALSE sinon elle retourne le nom de l'entrée actuellement lue. ATTENTION, les entrées « . » et « .. » sont lues comme les autres.

On ferme un répertoire en cours de lecture avec la fonction **closedir()** qui prend comme paramètre l'identifiant de répertoire.

```
$handle=opendir('.');
while ($file = readdir($handle)) {
   if ($file != "." && $file != "..") {
       echo "$file\n";
   }
}
closedir($handle);
```

On peut retourner au début du répertoire avec la fonction **rewinddir()** qui prend comme paramètre un identifiant de répertoire. La prochaine lecture sera la première du répertoire.

11.2.4Récupérer toutes les entrées

Même s'il est possible de récupérer l'ensemble des entrées d'un répertoire avec les fonctions

précédentes et une boucle, la fonction **scandir()** propose la même chose, en une ligne et en plaçant tous les noms de fichiers du chemin indiqué dans un tableau. Le premier paramètre est un chemin de répertoire. Le second est optionnel et définit l'ordre du tri qui est alphabétique par défaut. Un « 1 » indique un tri alphabétique décroissant.

```
$files=scandir('./rep');
print_r($files);
```

11.3Travail sur le système de fichiers

11.3.1Informations sur le fichier

On peut obtenir diverses informations sur un fichier à l'aide des commandes de type **is_x()** :

- **is_dir()** : TRUE si le nom de fichier est un répertoire
- **is_file()** : TRUE si le nom de fichier est un véritable fichier normal
- **is_executable()** : TRUE si le fichier est exécutable. Sous Windows il faut PHP 5.x.
- **is_readable()** : TRUE sir le fichier est lisible
- **is_writeable()**, **is_writable()** : TRUE si le fichier est accessible en écriture
- **is_link()** : TRUE si le fichier est un lien.

De même, on peut obtenir des détails sur un fichier avec les fonctions **filexxx()** :

- **file_exists()** : TRUE si le fichier existe
- **fileatime()** : retourne sous forme de TIMESTAMP la dernière date d'accès. Sous Windows, ce sera la dernière date de modification.
- **filectime()** : retourne la date de création
- **filemtime()** : retourne la date de modification
- **filegroup()** : retourne le GID du fichier. On récupère le nom réel du groupe avec la fonction **posix_getgrgid()**.
- **fileowner()** : retourne le UID du fichier. On récupère le nom réel de l'utilisateur avec la fonction **posix_getpwuid()**.
- **fileinode()** : retourne l'inode du fichier
- **fileperms()** : retourne les permissions du fichier. ATTENTION le résultat est en base 10. Il faudra convertir en base 8 avec la fonction **base_convert()**.
- **filesize()** : retourne la taille du fichier en octets.

- **filetype()** : retourne le type du fichier. ATTENTION il s'agit des types de fichiers sur disque, pas de leur contenu (image, etc) : fifo, char, dir, block, link, file, unknown.

Si un fichier est déjà ouvert on peut obtenir des informations avec la fonction **fstat()** qui prend comme paramètre un identifiant de fichier retourné par fopen() et qui retourne un tableau indexé numériquement ou associatif. La fonction **stat()** retourne la même chose mais on lui précise un chemin ou nom de fichier. La fonction **lstat()** est quasi-identique sauf que dans le cadre du lien, ce sont les informations sur le lien lui-même (et pas le fichier pointé) qui sont retournées.

0.volume (ou encore **dev**)

1.inode (ou encore **ino**)

2.droits d'accès au fichier (mode de protection du inode). A convertir en octal. Voir aussi fileperms(). (ou encore **mode**)

3.nombre de liens (ou encore **nlink**)

4.id de l'utilisateur propriétaire (ou encore **uid**)

5.id du groupe propriétaire (ou encore **gid**)

6.type du volume de l'inode * (ou encore **rdev**)

7.taille en octets (ou encore **size**)

8.date du dernier accès (ou encore **atime**)

9. date de la dernière modification (ou encore **mtime**)

10.date du dernier changement (ou encore **ctime**)

11.taille de bloc du système pour les entrées/sorties * (ou encore **blksize**)

12.nombre de blocs alloués (ou encore **blocks**)

11.3.2Actions sur les fichiers et répertoires

On retrouve ici des fonctions qui portent souvent le même nom que leurs équivalents sous le shell Unix :

- **dirname()**, **basemane()** : isolent le chemin et le nom d'un fichier

- **chown()**, **chgrp()** : changent le propriétaire et le groupe d'un fichier. Le premier paramètre est le fichier, le second le groupe ou le propriétaire (UID, GID, ou en toutes lettres)

- **chmod()** : modifie les droits. Le premier paramètre est le nom du fichier, le second les droits en octal. ATTENTION sous Windows, les résultats peuvent surprendre...

- **copy()** : copie d'un fichier : source et destination
- **link()** : créé un lien hard link: cible, nom du lien
- **symlink()** : créé un lien symbolique : cible, nom du lien
- **mkdir()** : créé un répertoire, le second paramètre est le mode (droits par défaut) AVANT application du masque.
- **rename()** : renomme et/ou déplace un fichier : orgine, destination.
- **rmdir()** : supprime un répertoire. Il doit être vide.
- **unlink()** : supprime un fichier.
- **umask()** : modifie le umask par défaut

11.4Upload de fichiers

L'upload de fichiers permet d'envoyer depuis un poste client un fichier sur le serveur. Cette opération nécessite un navigateur sachant prendre en charge une balise spéciale et l'appel d'un script PHP.

Côté Page HTML, il faut modifier la définition d'un formulaire, placer le champ formulaire de type **FILE** et indiquer la taille maximale du fichier téléchargé avec un nom **HIDDEN** de bom **MAX_FILE_SIZE**. Cette taille est précisée en octets. Ce champ doit être placé AVANT l'apparition du champ de type FILE.

Le formulaire doit être de type POST **(**bien que GET soit possible, voir documentation PHP) et le type d'encodage **ENCTYPE "/multipart/form-data"** doit être précisé.

```
<form ENCTYPE="multipart/form-data" action="/page.php" method="post">
 <INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="35840">
 <INPUT NAME="avatar" TYPE="file" SIZE=50>
 <input type="submit" value="Enregistrer"/>
</form>
```

Notes : Certains navigateurs se fichent totalement du poid (taille) des fichiers envoyés en ne réagissant pas à la variable MAX_FILE_SIZE. Dans ce cas la vérification doit être faite du côté du serveur. Au pire on peut limiter côté serveur la taille maximale à l'aide du paramètre **upload_max_filesize** dans le php.ini.

On choisit son fichier via la boite de dialogue du gestionnaire de fichiers quand on clique sur le bouton Parcourir. Une fois le formulaire validé, les données du formulaire et le fichier sont envoyés à la page PHP.

Côté PHP, on récupère les informations sur le fichier uploadé via un tableau **$_FILE** dont les index sont les suivants **$_FILE[nom][propiete]** le nom étant le nom du champ du formulaire FILE et la propriété :

- $_FILES['userfile']['name'] : Le nom original du fichier, tel que sur la machine du client web.

- $_FILES['userfile']['type'] : Le type MIME du fichier, si le navigateur a fourni cette information. Par exemple, cela pourra être "image/gif"

- $_FILES['userfile']['size'] : La taille, en octets, du fichier téléchargé.

- $_FILES['userfile']['tmp_name'] : Le nom temporaire du fichier qui sera chargé sur la machine serveur.

- $_FILES['userfile']['error'] : Le code d'erreur associé au téléchargement de fichier (PHP 4.2)

Par défaut le fichier téléchargé et en attente de déplacement est placé dans le répertoire temporaire du système (variable système TEMP ou TMPDIR) ou dans le répertoire précisé par le paramètre de configuration **upload_tmp_dir** du fichier php.ini. Il est conseillé d'initialiser et d'utiliser ce paramètre. Le chemin du fichier temporaire uploadé est donc :

```
$tmp_file=ini_get("upload_tmp_dir").'/'.$_FILE['userfile']['tmp_name'];
```

Pour savoir si un fichier a été correctement uploadé par la méthode POST, on peut utiliser la fonction **is_uploaded_file()** qui prend comme paramètre le nom du fichier uploadé.

```
if(is_uploaded_file($_FILE['userfile']['name'])) {// ou tmp_name
 echo $_FILES['userfile']['size'];
}
```

Sachant la position du fichier temporaire uploadé et s'il a été correctement uploadé, on peut utiliser la fonction copy. Mais le mieux est d'utiliser la fonction **move_uploaded_file()** qui teste la réussite de l'upload et qui déplace le fichier. Le premier paramètre est le nom du fichier uploadé. Ce peut être le nom du champ FILE, le nom réel du fichier ou le nom du fichier temporaire.

```
if(move_uploaded_file($_FILES['avatar']['tmp_name'],$_SERVER['DOCUMENT_ROOT']."/
forum/avatars/$login".$_FILES['avatar']['name'])) {
 $avatar=$login.$_FILES['avatar']['name'];
 chmod($_SERVER['DOCUMENT_ROOT']."/forum/avatars/$avatar",0644);
}
```

12Travail sur chaînes

12.1Fonctions pour code HTML et MySQL

12.1.1Gestion des caractères verrouillés

La fonction **addslashes()** rajoute des slashes (backslashes \) devant tous les caractères qui doivent être verrouillés. Ces caractères sont le « " », le « ' », le « \ » et le caractère nul (NULL). Ainsi la chaîne résultante peut être utilisée pour être placée en base de données via des commandes SQL.

La fonction **stripslashes()** effectue l'opération inverse : elle supprime les anti-slash sauf dans le cas des doubles anti-slash \\, elle n'en supprime qu'un.

```
$chaine="L'oreille cassée";
$c1=addslashes($chaine);
echo $c1; // L\'oreille cassée
echo stripslashes($c1); // L'oreille cassée
```

12.1.2balises et caractères spéciaux HTML

La fonction **strip_tags()** permet de filtrer une chaîne contenant des balises (tags) HTML en autorisant certains et interdisant d'autres. Le premier paramètre est la chaîne HTML. Le second est la liste des tags autorisés. **ATTENTION** : la fonction ne modifie pas le contenu des balises autorisées, laissant ainsi les attributs CLASS, STYLE et appels JavaScript.

```
$chaine="<u><i><b>Bonjour les amis</b></i></u>";
echo strip_tags($chaine,'<u><i><b>'); // Bonjour les amis
echo strip_tags($chaine,'<b>'); // <b>Bonjour les amis</b> en gras
```

La fonction **htmlspecialchars()** convertit certains caractères spéciaux en entités HTML. Ces caractères sont :

- « & » :&
- « " » : " si ENT_NOQUOTES est utilisé
- « ' » : &m039; si ENT_QUOTES est utilisé
- « < » : <
- « > » : >

Le second paramètre (ENT_xxx) définit le fonctionnement par rapport aux quotes.

La fonction **htmlentities()** va plus loin en remplaçant TOUS les caractères qui ont des équivalents en HTML. Pour décoder une telle chaîne, il faut alors utiliser la fonction **html_entity_decode()**.

La fonction **nl2br()** remplace tous les retour-chariots d'une chaîne en retours à la ligne HTML :
.

La fonction **urlencode()** formate une chaîne pour son utilisation dans une URL. Les espaces sont

remplacés par des « + » et tous les caractères non-alphanumériques (sauf « - », « . » et « _ ») sont remplacés par leur équivalent hexadécimal %XX. Une telle chaîne sera décodée par la fonction **urldecode()**.

12.2Fonctions texte générales

12.2.1Découper une chaîne

La fonction **explode()** permet de découper une chaîne selon un séparateur. Elle retourne un tableau contenant les divers éléments découpés. Le premier paramètre est le séparateur, le second la chaîne. Un troisième paramètre permet de limiter le nombre de valeurs retournées.

```
$chaine='login:pwd:uid:gid';
$tab=explode(':',$chaine);
 oreach($tab as $value) echo $value; // login pwd uid gid
```

La fonction inverse se nomme **implode()**.Le premier paramètre est la « colle » : le ou les caractères de séparation qui seront placés entre les divers éléments de la chaîne reconstituée. Le second paramètre est un tableau.

```
// suite de l'exemple précédent
$newchaine=implode(',',$tab);
echo $chaine; // login,pwd,uid,gid
```

La fonction **join()** est un alias de implode().

On peut aussi effectuer un découpage tous les n caractères avec la fonction **chunk_split()**. Elle découpe le premier paramètre en morceaux de longueur du second paramètre (par défaut 76 caractères) avec comme séparateur le troisième paramètre (\n\r par défaut).

```
$chaine='FF5EB649C4';
$n_chaine=chunk_split($chaine,2,':');
echo n_chaine; // FF:5E:B6:49:C4
```

La fonction **wordwrap()** effectue la césure d'une chaîne. La chaîne résultante sera le texte découpé en tenant compte des mots. Par défaut la longueur (second paramètre) est de 75 caractères. On peut préciser la chaîne de coupure de ligne en troisième paramètre, et le dernier paramètre s'il est à 1 force la coupure même au milieu d'un mot. Pour un affichage en HTML, il faut encapsuler le résultat dans un nl2br().

```
$chaine="Bonjour les amis et bienvenus dans le merveilleux monde du langage
PHP !";
echo nl2br(wordwrap($chaine,20));
```

retourne

```
Bonjour les amis et
bienvenus dans le
merveilleux monde
du langage PHP !
```

La fonction **str_split()** découpe une chaîne de caractère en tableau. Par défaut la longueur est de 1 ais on peut la choisir en second paramètre.

On peut découper une partie de chaîne avec la fonction **substr()**. Le premier paramètre est la chaîne, le second le début (depuis 0) et le dernier la longueur. Le début de la chaîne peut être une valeur négative et dans ce cas on compte en partant de la fin de la chaîne.

```
$chaine='abcdef';
echo substr($chaine,0); // abcde
echo substr($chaine,2,3); // cde
echo substr($chaine,-1); //
echo substr($chaine,-3,2); // de
```

Les fonctions **trim()**, **rtrim()** et **ltrim()** suppriment les caractères invisibles à droite (trim, rtrim) et à gauche (trim, ltrim) de la chaîne.

12.2.2Rechercher et comparer

La fonction **strcmp()** effectue une comparaison de deux chaînes. Retourne <0 si la chaine1 est inférieure à chaine2, 0 si égale, >0 sinon. La fonction **strncmp()** à le même effetmais sur une longueur donnée, en troisième paramètre.

Les fonctions **strcasecmp()** et **strncasecmp()** sont des versions insensibles à la casse.

La fonction **strstr()** recherche la première occurrence du second paramètre dans le premier et retourne la chaîne depuis cette position jusqu'à la fin. La fonction **strchr()** est un alias de strstr(). La fonction **stristr()** est la même mais insensible à la casse.

La fonction **strrchr()** effectue quasiment la même chose sauf qu'elle retourne la fin de la chaîne depuis le dernier caractère trouvé.

La fonction **strpos()** recherche dans la chaîne du premier paramètre la chaîne en second paramètre et retourne sa position. On peut placer en troisième paramètre la position de départ de la recherche.

13Sessions et cookies

13.1Les cookies

13.1.1Création

Un cookie s'envoie avec la fonction **setcookie()**. Les arguments sont les suivants :

- No
- Valeur
- date d'expiration (un timestamp unix)
- chemin (validité du cookie suivant le chemin dans l'url)
- domaine (pare défaut, le vôtre)
- secure : flag de sécurité : le cookie n'est accessible que via une connexion sécurisée.

Seul le premier argument est obligatoire. Pour ne pas spécifier une valeur, il suffit de ne rien mettre. Attention à quelques limitations :

- Un cookie étant envoyé avec les entêtes HTTP, il ne peut pas être envoyé si une sortie a déjà eu lieu (html, texte, ...)
- Un cookie n'est pas disponible dans la page qui l'a créé. Il faut soit recharger la page, soit pointer sur une autre.
- Un cookie ne peut être supprimé qu'avec les mêmes paramètres qui ont servi à sa création. C'est le navigateur qui supprime le cookie.

```
setcookie("testcookie",session_id(),time()+3600);
if(isset($_COOKIE['testcookie'])) echo $_COOKIE['testcookie'];
```

Dans l'exemple précédent, au premier chargement rien ne sera affiché car le contenu du cookie n'est pas encore accessible. Au deuxième chargement la valeur sera réellement affichée, mais attention au décalage entre la mise à jour et l'accès.

```
if(!isset($_COOKIE['testcookie'])) {
 setcookie("testcookie",'toto',time()+3600);
 header("Location: http://".$_SERVER['HTTP_HOST'].$_SERVER['REQUEST_URI']);
}
```

Dans cet exemple, on regarde si le cookie existe, sinon on le créé et on rappelle la page.

13.1.2Accès

On accède au cookie grkce à la variable globale $_COOKIE qui est un tableau. L'index du tableau est le nom du cookie.

```
$valeur=$_COOKIE['testcookie'];
echo $valeur; // par rapport à l'exemple précédent : toto
```

Note : on peut placer les tableaux avec les cookies. Il suffit de nommer son cookie avec une notation par crochets.

```
setcookie("testcookie[1]",'toto',time()+3600);
setcookie("testcookie[2]",'titi',time()+3600);
setcookie("testcookie[3]",'tata',time()+3600);

... (page suivante)

 oreach($_COOKIE as $tab) {
 if(is_array($tab)) foreach($tab as $key => $value) echo "$key => $value";
}
```

13.1.3Suppression

Pour supprimer un cookie, il suffit de donner une date antérieure à la date actuelle à celui-ci.

```
if(!isset($_COOKIE['testcookie'])) {
 setcookie("testcookie",'toto',time()+3600);
} else {
 setcookie("testcookie",'',time()-3600);
}
if(isset($_COOKIE['testcookie'])) echo "OK";
else echo "Pas de cookie";
```

Dans l'exemple précédent, les appels au script vont provoquer successivement l'affichage de « pas de cookie » et « OK ».

13.2Identification HTTP

13.2.1Connexion

Cette possibilité n'est disponible que si PHP est utilisé comme module Apache, et pas en tant que script CGI. L'identification HTTP permet via l'envoi d'un HEADER HTTP de demander une identification au client. Dans ce cas le navigateur va ouvrir une fenêtre de connexion demandant une saisie de login et de mot de passe. PHP récupère bien entendu ces valeurs. Voici l'exemple issu de la documentation PHP

```
if (!isset($_SERVER['PHP_AUTH_USER'])) {
 header('WWW-Authenticate: Basic realm="Slyunix"');
 header('HTTP/1.0 401 Unauthorized');
 echo 'Texte utilisé si le visiteur utilise le bouton d\'annulation';
 exit;
} else {
 echo "<p>Bonjour, {$_SERVER['PHP_AUTH_USER']}.</p>";
 echo "<p>Votre mot de passe est {$_SERVER['PHP_AUTH_PW']}.</p>";
}
```

Dans cet exemple, on passe en entête une demande d'authentification. En cas d'échec (bouton annuler) un message d'annulation est affiché. Sinon on peut récupérer les valeurs via les variables :

- $_SERVER['PHP_AUTH_USER'] : login

- $_SERVER['PHP_AUTH_PW'] : mot de passe

13.2.2Déconnexion

La déconnexion n'est pas si évidente que ça car une fois connecté, les identifiants sont placés dans la session. La méthode la plus simple consiste à fermer le navigateur et à le relancer.

13.3Sessions PHP

13.3.1Principe

Les sessions permettent de préserver des données lors de la visite d'un site. Chaque personne se voit attribué un identifiant unique appelé identifiant de session, ou SID. Ce SID est soit stocké dans un cookie, soit passé par URL. On peut ainsi définir un nombre infini de variables qui seront accessibles durant toute la durée de la session.

Notez que si vous fermez et relancez votre navigateur, vous changez d'identifiant, et donc la précédente session est perdue, même si elle était nommée.

On peut connaître son SID grkce à la constante de même nom, ou par la fonction **session_id()**.

13.3.2Utilisation

13.3.2.1Ouverture

Si dans PHP.INI la valeur « session.auto_start » est activée, les sessions sont démarrées de manière automatique lors de l'accès à une page. Sinon il faut utiliser explicitement la fonction **session_start()**. Son appel créé une session ou restaure une session précéde ent stockée sur le serveur.

ATTENTION : L'ouverture d'une session doit obligatoirement apparaître avant toute sortie/affichage du script PHP ou de balises HTML, même une ligne vide !

On peut créer ou récupérer une session nommée en précisant un **session_name()** qui prend comme paramètre le nom de la session qui doit être créée (premier appel) ou accédée (appels suivants). La session par défaut se nomme « **PHPSESSID** » et c'est celle-là qui est chargée si session_name() n'est pas appelée. Il faut donc appeler session_name() AVANT session_start() pour accéder à une session nommée.

Une session peut aussi être indirectement créée ou ouverte avec l'utilisation de la fonction **session_register()**.

13.3.2.2Variables de session

Il y a deux moyens de créer des variables de session, qui seront accessibles depuis n'importe quelle page visitée à partir du moment ou session_start() a été appelée. Le premier moyen ne fonctionne ne fonctionne pas si **register_globals** est à **off** dans PHP.INI :

Méthode 1 (déconseillée) :

La fonction **session_register()** prend comme paramètre un nom de variable et la définit comme variable de session. Cette variable peut ensuite être accédée par son nom.

```
$txt='Bonjour les amis';
session_register('txt'); // txt devient une variable de session.
```

Pour savoir si une variable est enregistrée dans la session courante, on utilise **session_is_registered()**.

```
if(session_is_registered('txt')) echo $txt;
```

Pour supprimer une variable de session, il faut utiliser **session_unregister()**. Par contre il ne faut pas oublier d'utiliser **unset()** pour supprimer la variable globale, car la précédente fonction ne la supprime pas, elle empêche seulement sa sauvegarde à la fin du script.

```
if(session_is_registered('txt')) {
 session_unregister('txt');
 unset($txt);
}
```

Méthode 2 (La meilleure à utiliser) :

On utilise tout simplement le tableau global **$_SESSION**. Ainsi :

```
$_SESSION['txt']="Bonjour les amis";
```

crée la variable de session txt,

```
echo $_SESSION['txt']
```

l'affiche et

```
unset($_SESSION['txt']);
```

l'efface. Pour effacer toutes les variables de sessions, il suffit de faire

```
$_SESSION=array();
```

13.3.3Expiration de la session

La durée de vie d'une session PHP dépend de trois paramètres de configuration du fichier PHP.INI : **session.gc_maxlifetime, session.cookie_lifetime** et **session.cache_expire**. Le premier concerne la durée de vie des données sur le serveur, en nombre de secondes. Il est par défaut de 1440 secondes, soit 24 minutes. Au delà, les données sont supprimées et elles seront rechargées via le cookie de session lors du prochain appel.

Le second est la durée de vie du cookie de session, exprimé en secondes. Par défaut la valeur est zéro (0) ce qui signifie : jusqu'à ce que le navigateur soit éteind.

Le troisième est la durée de vie des données de session, en minutes. Sa valeur par défaut est de 180 inutes, soit trois heures. Ce qui veut dire que dans une même session, on peut rester trois heures sur la même page, les valeurs ne sont pas perdues. Au delà, il faut rouvrir une nouvelle session et

restaurer de nouvelles valeurs. A chaque chargement de page (ouverture de session), on recommence le décompte.

On peut modifier le délai d'expiration de la session grkce à la fonction **session_cache_expire()** qui prend comme valeur la nouvelle durée en minutes. Sans paramètre elle retourne la valeur actuelle.

13.3.4Changer l'accès par défaut

PHP sait très bien se débrouiller tout seul mais on peut remplacer la prise en charge par défaut des sessions par ses propres fonctions, dans le but de gérer l'ouverture et la fermeture ou encore la sauvegarde des données de session dans un fichier ou une base de données.

Une autre utilisation peut être le traçage via PHP de toutes les ouvertures/fermetures des sessions.

Pour ça on utilise la fonction **session_set_save_handler()**. Son étude dépasse le cadre du cours ais voici un exemple tiré de la documentation PHP.

```
if(!mysql_table_exists("sessions",$DB)) {
 $query = 'CREATE TABLE sessions
 (
  SessionID    char(255)  not null,
  LastUpdated  datetime    not null,
  DataValue    text,
  PRIMARY KEY ( SessionID ),
  INDEX ( LastUpdated )
 )';
 mysql_query($query);
}


 unction sessao_open($aSavaPath, $aSessionName){
 global $aTime;
 sessao_gc( $aTime );
 return True;
}

 unction sessao_close(){
 return True;
}

 unction sessao_read( $aKey ) {
 $query = "SELECT DataValue FROM sessions WHERE SessionID='$aKey'";
 $busca = mysql_query($query);
 if(mysql_num_rows($busca) == 1) {
  $r = mysql_fetch_array($busca);
  return $r['DataValue'];
 } ELSE {
  $query = "INSERT INTO sessions (SessionID, LastUpdated, DataValue)
            VALUES ('$aKey', NOW(), '')";
  mysql_query($query);
  return "";
 }
}

 unction sessao_write( $aKey, $aVal ) {
 $aVal = addslashes( $aVal );
 $query = "UPDATE sessions SET DataValue = '$aVal', LastUpdated = NOW()
      WHERE SessionID = '$aKey'";
 mysql_query($query);
 return True;
```

```
}

 unction sessao_destroy( $aKey ) {
 $query = "DELETE FROM sessions WHERE SessionID = '$aKey'";
 mysql_query($query);
 return True;
}

 unction sessao_gc( $aMaxLifeTime )
{
 $query = "DELETE FROM sessions WHERE UNIX_TIMESTAMP(NOW()) - UNIX_TIMESTAMP
      (LastUpdated) > $aMaxLifeTime";
 mysql_query($query);
 return True;
}

session_set_save_handler("sessao_open", "sessao_close", "sessao_read",
 "sessao_write", "sessao_destroy", "sessao_gc");
```

14Images

14.1Pré-requis importants

Ceci concerne les versions de PHP avant la 4.2.3 et EasyPHP 1.6: certaines fonctions comme gd_info() ou encore imagecreatetruetype() ne sont pas toujours disponibles car PHP fait appel à une bibliothèque externe appelée GD2. Pour les version 4.2.3 et au-delà on trouve des patches à l'adresse http://www.boutell.com/gd/. Avant ces versions, la version GD linkée avec PHP est la 1.8.4.

Ceci concerne les utilisateurs de PHP 4.2.3 et + et EasyPHP 1.7 sur Windows : la configuration par défaut de EasyPHP 1.7 n'inclut pas le support des images. Pour cela il faut modifier le fichier PHP.INI en retirant le commentaire de :

```
extension=php_gd2.dll
```

14.2Créer une image

14.2.1Types possibles

Il faut tout d'abord choisir le type de l'image que vous souhaitez créer. Pour cela on peut tester quels types PHP peut générer avec la fonction **imagetypes()** et ses constantes associées :

- **IMG_PNG** : format PNG
- **IMG_JPG** : format JPEG
- **IMG_GIF** : format GIF
- **IMG_WBMP** : format Windows Bitmap BMP

```
if (imagetypes() & IMG_PNG) {
   echo "Le type PNG est supporté";
```

Puis au début du script, on envoie un header spécifiant le type d'image :

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

14.2.2Création

14.2.2.1Nouvelle image dynamique

Et enfin on créé une image avec la fonction **imagecreate()** (basée sur une table de couleurs indexée) ou **imagecreatetruecolor()** (en truecolor) avec comme paramètres la largeur et la hauteur. La seconde est conseillée par la documentation PHP. Les fonctions retournent une ressource identifiant l'image. **Par défaut l'image est noire.**

```
$img=@createimage(100,100) or die("Votre PHP ne supporte pas les images");
```

14.2.2.2Récupération d'une image existante

PHP peut charger une image existante au lieu de la créer de toutes pièces. Dans ce cas l'image est chargée depuis un fichier et on peut ensuite la retravailler comme on le souhaite. On peut utiliser les fonctions :

- **imagecreatefromgif()** : charge une image GIF
- **imagecreatefromjpeg()** : charge une image JPEG
- **imagecreatefrompng()** : charge une image PNG
- **imagecreatefromstring()** : charge une image depuis une chaîne de texte (par exemple un champ)

14.2.3Publication

Enfin, on publie l'image avec les fonctions **imagexxx()**. Si cette fonction est utilisée sans paramètre, l'image créée est redirigée vers la sortie standard. On peut toutefois préciser comme paramètre un nom de fichier. Dans ce cas, l'image est sauvée sur disque au format spécifié.

- **imagepng()** : créé une image au format PNG
- **imagegif()** : créé une image au format GIF
- **imagejpeg()** : créé une image au format JPEG

Une fois l'image crée, on peut la supprimer avec **imagedestroy()** ce qui la décharge de la mémoire.

Voici donc le code d'une page PHP image.php retournant une image :

```
header ("Content-type: image/png");
$img=@imagecreatetruecolor (100, 50) or die ("Impossible de crée un flux d'image
GD");

/* ICI ON PLACE LES COMMANDES DE DESSIN OU MANIPULATION */
imagepng ($img);
imagedestroy ($img);
```

14.2.4Chargement via une autre page

Pour appeler cette image dynamique depuis une page HTML ou PHP il suffit de la placer dans une balise IMG :

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

14.3Travailler sur les Images

14.3.1Récupérer des informations

La fonction **getimagesize()** permet de récupérer des informations sur une image si on lui passe un nom de fichier (pas de ressource image). Elle retourne un tableau indexé.

```
list($width,$height,$type,$attr)=getimagesize("toto.gif");
```

- width : largeur x
- height : hauteur y
- type : 1 = GIF, 2 = JPG, 3 = PNG, 5 = PSD, 6 = BMP, 7 = TIFF (Ordre des octets Intel), 8 = TIFF (Ordre des octets Motorola), 9 = JPC, 10 = JP2, 11 = JPX, 12 = JB2, 13 = SWC, 14 = IFF
- attr: chaine width/height pour HTML

Si on travaille sur une image qu'on a créé nous-même :

- **imagesx()** : largeur x de l'image
- **imagesy()** : hauteur de l'image

14.3.2Les couleurs

Il n'est pas possible de voir toutes les fonctions cependant une chose est à retenir : la notion de palette indexée (createimage()) et la notion de truetype. Dans le premier cas les couleurs sont indexées via un numéro. Dans le second cas, la couleur est une composante RVB.

La fonction **imagecolorallocate()** permet de définir une couleur qui sera utilisée pour une image. Le premier paramètre est le ressource de l'image, les trois autres les niveaux de rouge, vert et bleu de la couleur.

ATTENTION : Avec une image créée avec createimage(), la palette est indexée et le premier index est la couleur du fond. Donc : un premier appel à imagecolorallocate() dans ce cas modifie la couleur du fond.

```
$white=imagecolorallocate($img,255,255,255);
```

La fonction **imagecolorat()** retourne la valeur exacte (soit l'index de la palette, soit un entier RVB) du point aux coordonnées x/y de l'image.

```
$color=imagecolorextact($img,50,50);
```

La fonction **imagecolortransparent()** permet de définir la couleur de transparence d'une image dont la palette est indexée.m **Le premier paramètre est la ressource de l'image OBLIGATOIREMENT créée avec imagecreate()**, le second une valeur retournée par imagecolorallocate().

La fonction **imagecolorstotal()** retourne le nombre total de couleurs utilisées pour l'image donnée.

14.3.3Dessin

14.3.3.1Remplissage

On utilise la fonction **imagefill()** :

```
$background=imagecolorallocate ($img,255,255,255);
ImageFill($img,0,0,$background);
```

La fonction **imagefilltoborder()** permet un remplissage d'une zone délimitée par une couleur donnée. Dans ce cas les deux derniers paramètres sont la couleur de la bordure et la couleur de remplissage.

14.3.3.2Point

On utilise **imagesetpixel()** :

```
$green=imagecolorallocate ($img,0,255,0);
Imagesetpixel($img,2,2,$green);
```

14.3.3.3Ligne

On utilise **imageline()** :

```
imageline($img, 10,10,30,40,$blue);
```

14.3.3.4Rectangle

On utilise **imagerectangle()** pour un rectangle simple, **imagefilledrectangle()** pour un rectangle rempli.

```
$blue=ImageColorAllocate($img,0,0,255);
imagerectangle($img, 10,10,30,40,$blue);
imagefilledrectangle($img,12,12,27,37,$green);
```

14.3.3.5Cercle, ellipse

On utilise **imageellipse()** ou **imagefilledellipse()**. Attention les deux premiers paramètres sont les coordonnées du centre de l'ellipse, les deux autres la largeur et la hauteur de l'ellipse.

```
imageellipse ($img, 50, 50,30, 20, $red);
```

On peut aussi faire des arcs de cercles avec les fonctions **imagearc()** et **imagefilledarc()** : Juste

avant la couleur on précise le début et la fin de l'arc en degrés (de 0 à 360).

14.3.3.6Polygones

On utilise les fonctions **imagepolygone()** et **imagefilledpolygone()**. Les sommets du polygone sont définis dans un tableau : Tab[0]=x0, Tab[1]=y0, Tab[1]=x1, Tab[2]=y1, ...

On doit aussi préciser le nombre de sommets.

```
imagepolygon($img, array (0, 0,100, 200, 300, 200),3, $blue);
```

14.3.3.7Texte simple

On utilise **imagestring()** :

```
imagestring($img, 1, 5, 5,  "Une simple chaîne de texte", $text_color);
```

Le second paramètre est le numéro de police par défaut de la page actuelle ou du navigateur.

15La programmation objet

15.1Structure

Une classe est une collection de variables (attributs) et de fonctions (méthodes) qui fonctionnent avec ces variables. L'ensemble forme les membres ou les propriétés de l'objet. Une classe est définie en utilisant le mot-clé « **class** » :

```
class Panier {
      // Eléments de notre panier
      var $contenu;

      // Ajout de $qte articles de type $code au panier
      function ajout_article ($code, $qte) {
               if(isset($this->contenu[$code])) $this->contenu[$code] += $qte;
               else $this->contenu[$code] = $qte;
      }

      // Suppression de $num articles du type $artnr du panier
      function suppr_article ($code, $qte) {
               if ($this->contenu[$code] > $qte) {
                  $this->contenu[$code] -= $qte;
                  return TRUE;
               } else {
                 return FALSE;
               }
      }
}
```

15.2Création d'un objet

On utilise le mot-clé « **new** ».

```
$mon_caddie=new Panier;
```

Si le constructeur existe (voir plus loin) on peut bien entendu passer des valeurs à la création :

```
$o1=new Obj("riri");
```

15.3A èsaux membres

15.3.1PHP 4 et

On utilise les caractères « **->** ». On accède aux membres avec la syntaxe suivante. Notez qu'on utilise pas le $ devant un nom d'attribut.

```
$mon_caddie->contenu=array('ref01' => 2, 'ref02' => 1);
```

Au sein d'un objet, on accède aux attributs de la classe en la précisant avec la variable « **$this** ». Si on ne le précise pas, on utilise en fait la variable localz à la méthode.

```
$this->contenu['ref01']+=1;
```

15.3.2PHP

Le PHP 5 étend les possibilités d'accès aux membres (variables et fonctions) via les variables dynamiques. Ex :

```
class Panier {
      // Eléments de notre panier
      public $etat=false; // false : vide
      ...
}

 unction bidon() {
      return 'etat';
}

$mon_caddie=new Panier;
$pipo='etat';
echo $mon_caddie->$pipo; // affiche $this->etat
$fonction='affiche_panier';
$mon_caddie->$fonction();
echo $nom_caddie->{bidon()};
```

15.4L'opérateur ::

L'opérateur « **::** » permet de faire référence aux membres d'une classe de base (voir héritage), ou d'utiliser des méthodes alors que la classe n'a pas été instanciée.

```
class A {
  function example() {
   echo "Je suis la fonction originale.<br>\n";
  }
}

// Il n'y a pas d'objets de classe A.
// L'affichage est : Je suis la fonction originale.
A::example();
```

Dans le cas d'un appel à une méthode avec :: d'une classe non instanciée (pas d'objets), la méthode ne doit pas appeler d'attributs de la classe, puisqu'ils n'existent pas. De même pour la même raison il ne faut pas employer la variable $this dans les méthodes appelées.

15.5Les variables, protection

15.5.1PHP 4.x

On déclare les attributs avec le mot-clé « **var** ». Les attributs sont considérés comme étant tous publics. PHP 4 ne possède aucun mécanisme de protection des variables. Ainsi un attribut est accessible depuis toute les classes dérivées et depuis l'extérieur de la classe.

```
var $mavariable;
```

Il est interdit d'initialiser une variable lors de sa déclaration.

```
var $toto="salut !"; // Ne marche pas !
```

15.5.2PHP 5.x

15.5.2.1Protection des variables et fonctions

Le PHP 5 rajoute la notion de protection des données qui manquait à PHP 4. On appelle ça **les trois « P »** : « **public** », « **protected** », « **private** », avec les mots-clés correspondants. Voici un tableau récapitulatif de leur effet:

Accès	Public	Protected	Private
Dans la classe elle-même	✓	✓	✓
Depuis une classe dérivée	✓	✓	×
Depuis l'extérieur	✓	×	×

Note : Les mots-clés `public`, `private` et `protected` peuvent bien entendu être utilisés pour les méthodes (fonctions) auquel cas la protection s'étend de la même manière.

Le mot-clé « **var** » est toujours utilisable en PHP 5 et dans ce cas les attributs sont considérés comme publics.

Il est possible d'initialiser les variables à leur création avec n'importe quelle valeur.

```
class Panier {
      // Eléments de notre panier, privé
      private $contenu=array();
      ...
}
```

15.5.2.2attributs et méthodes statiques

On peut aussi définir des attributs et des méthodes statiques qui seront accessibles en dehors de la classe, l'objet n'étant pas forcément déclaré. On rajoute le mot-clé « **static** » après la définition de la visibilité de l'attribut. Un « **static** » n'ayant pas de portée définie est considéré comme public. Par exemple, voici un compteur permettant de savoir combien de fois une classe a été instanciée. Notez l'utilisation de « **self** » à la place de « **$this** ». « **$this** » ne peut pas être utilisé ici il fait référence à un objet et un attribut statique fait référence à la classe elle-même.

```
class A {
    public static $compteur=0;
        function A()
        {
            self::$compteur++; // On incrémente
        }

        function affiche()
        {
            echo self::$compteur."<br />"; // self !
        }
}

$o1=new A;
$o1->affiche(); // 1
$o2=new A;
$o2->affiche(); // 2
$o3= new A;
$o3->affiche(); // 3

echo A::$compteur; // 3
```

15.5.2.3constantes de classe

On peut créer des constantes dans une classe à l'aide du mot-clé « **const** ». La déclaration d'une constante ne doit pas être précédée d'un indicateur de visibilité. Une constante est toujours publique. On ne peut pas non plus assigner à une constante une valeur elle-même dynamique, comme le retour d'une fonction. Comme pour les variables statiques, on utilise « **self** » pour accéder à sa valeur.

```
class A {
    const classname="A";
        function __construct()
        {
            echo self::classname;
        }
}
$o1=new A; // Affiche A
```

15.6Héritage et polymorphisme

L'héritage permet de définir des classes génériques réutilisables. On peut alors faire des extensions à une classe, faire dériver une classe d'une autre. La classe dérivée hérite de toutes les méthodes et variables de la classe de base et peut redéfinir ses propres fonctions et variables.

Une classe ne peut hériter que d'une seule autre classe. L'héritage multiple n'est pas supporté, ni par PHP 4, ni par PHP 5. Par contre rien n'empêche de faire des héritages en cascade.

Le « **polymorphisme** » permet de redéfinir une méthode de même nom dans une classe dérivée, par d'autres paramètres éventuels.

NOTE : Le « polymorphisme » n'est pas une surcharge. Une surcharge consiste à définir plusieurs méthodes de même nom mais de prototypes différents. PHP 5 ne supporte pas cette forme de surcharge. Le polymorphisme remplace une méthode par une autre dans une classe dérivée.

Pour créer un héritage on utilise le mot-clé « **extends** ».

```
class Panier_nomme extends Panier {
   var $proprietaire;
   function nom_proprietaire ($nom) {
       $this->proprietaire = $nom;
   }
}
```

On créé une classe « Panier_nomme » qui hérite de « Panier ». Elle possède donc les mêmes propriétés que la classe « Panier » à laquelle on rajoute un nouvel attribut et une nouvelle méthode.

```
$ncart = new Panier_nomme;
$ncart->nom_proprietaire ("Slyce");
print $ncart->proprietaire;
$ncart->ajout_article("ref03", 1);
```

Par le polymorphisme on peut redéfinir dans la classe dérivée les méthodes de la classe héritée.

```
class A {
      function affiche() {
               echo "Classe A<br />";
      }
}

class B extends A {
      function affiche() {
               echo "Classe B<br />";
      }
}

$test=new B;
$test->affiche(); // Affiche Classe B
```

15.7Les mots-clés parent et self

Le mot-clé « **parent** » représente la classe de base (celle présente après un « **extends** ») et permet de faire référence aux variables et fonctions d'une classe de base.

```
class A {
  function example() {
   echo "Je suis A::example(), fonctionnalité de base.<br>\n";
  }
}
class B extends A {
  function example() {
   echo "Je suis B::example(), fonctionnalité supplémentaire.<br>\n";
   parent::example();
  }
}
$b = new B;
// appelle B::example(), qui, à sont tour, appelle A::example().
$b->example();
```

Le mot-clé « **self** » représente la classe elle-même et peut donc être utilisée pour préciser l'accès à un membre précis, comme un attribut statique.

15.8Constructeurs

15.8.1PHP 4 et

Le constructeur est la méthode qui est automatiquement appelée lors de la création d'un objet par « **new** ». Une méthode devient un constructeur si elle porte le même nom que la classe dans laquelle elle est définie.

```
class auto_panier extends Panier_nomme {
	function auto_panier() {
		$this->proprietaire=session_id();
	}
}
```

Cette classe hérite de Panier_nomme. Le constructeur initialise le nom du propriétaire comme étant l'identifiant de session. Le constructeur peut prendre comme n'importe quelle méthode des paramètres optionnels.

```
class auto_panier extends Panier_nomme {
	function auto_panier($nom="anonyme") {
		$this->proprietaire=$nom;
	}
}

$panier=new auto_panier; // le constructeur est appelé avec sa valeur par défaut
$panier2=new auto_panier(session_id()); // valeur passée
```

15.8.2PHP

Outre le nom de la classe elle-même, PHP 5 propose une méthode générique appelée « **__construct** » (avec deux underscores «**_** » au début) en lieu et place de la première méthode vue ci-dessus. Ainsi :

```
 unction __construct() {
		$this->proprietaire=session_id();
}
```

revient exactement au même.

15.9Constructeurs et classes dérivées

15.9.1PHP 4 et

Soit les classes A et B suivantes :

```
class A {
	function A () {
		echo "Je suis la classe A<br />";
	}
}

class B extends A {
	function B () {
		echo "Je suis la classe B<br />";
	}
}
```

La création d'un nouvel objet de type B :

```
$b=new B;
```

Affichera

```
Je suis la classe B
```

Si une classe n'a pas de constructeur, c'est le constructeur de la classe de base qui est appelé :

```
class A {
      function A () {
                echo "Je suis la classe A<br />";
      }
}

class B extends A {
      var $var;
}

$b=new B; // Je suis la classe A
```

On remarque que les constructeurs ne sont pas appelés en cascade. Ainsi dans le cas la classe B, le constructeur de la classe A n'est pas appelé, ce qui est logique car PHP ne peut pas savoir par défaut comment appeler le constructeur (quels paramètres). On doit donc l'appeler explicitement à l'aide du ot-clé « **parent** »:

```
class A {
      function A () {
                echo "Je suis la classe A<br />";
      }
}

class B extends A {
  var $var;
    function B()
    {
        echo "Je suis la classe B<br />";
        parent::A();
    }
}

$b=new B; // Affiche : je suis la classe B, Je suis la classe A
```

15.9.2PHP

Le PHP 5 réagit de manière presque identique que PHP 4, sauf qu'il est possible d'appeler un constructeur en cascade, toujours avec le mot-clé « **parent** », mais sans connaître le nom du constructeur de la classe de base, en utilisant « **__construct** » :

```
class A {
      function A () {
                echo "Je suis la classe A<br />";
      }
}

class B extends A {
  var $var;
    function __construct()
```

```
    {
        echo "Je suis la classe B<br />";
        parent::__construct(); // Appelle le constructeur de A : A()
    }
}

$b=new B; // Affiche : je suis la classe B, Je suis la classe A
```

15.10Destructeur

15.10.1PHP 4

Il n'y a pas de destructeur en PHP 4. On fera confiance dans ce cas à la gestion de la mémoire (ramasse-miettes) de PHP, ou appeler une méthode particulière de la classe à la fin du script, chargée de faire le ménage, ce qui peut être important par exemple lorsqu'on a alloué une ressource pour une image dynamique. Pour détruire un objet, il suffit de réaffecter sa variable, ou de la détruire avec « **unset** ». L'objet sera détruit de toute façon implicitement à la fin de l'exécution du script.

15.10.2PHP

Le destructeur est présent en PHP 5. Il se nomme « **__destruct** » et sera automatiquement appelé lors de la destruction de l'objet. Tout comme le constructeur, il n'est pas appelé en cascade et doit donc être appelé explicitement lors d'un héritage.

```
class A {
      function __destruct() {
            echo "Au revoir la classe A<br />";
        }
}

class B extends A {

      function __destruct() {
            echo "Au revoir la classe B<br />";
            parent::__destruct();
    }
}

$b=new B;

unset($b); // destruction de l'objet, affiche Au revoir la classe B et A ...
```

15.11Affectation d'objets

15.11.1Code commun

Le fonctionnement des affectations des objets en PHP 4 et PHP 5 n'est pas le même. En, effet sur PHP 4, l'affectation se fait par copie, et par référence en PHP 5 ce qui peut induire de grandes différences de fonctionnement lorsque du code objet en PHP 4 est utilisé en PHP 5. Nous allons voir un exemple à l'aide d'une classe très simple.

```
class A {
    var $a;
       unction A()
        {
            $this->a="Salut";
```

```
        }

        function affiche()
        {
            echo "$this->a<br />";
        }
}
 unction modif($objet)
{
      $objet->a='coucou';
}
```

15.11.2PHP 4

En PHP 4, les objets sont affectés par copie, un nouvel objet est créé reprenant toutes les propriétés du premier, mais est totalement indépendant. Ainsi voici le résultat du code suivant :

```
$obj1=new A;
$obj2=$obj1; // affectation par copie

$obj1->affiche(); // affiche salut
$obj2->affiche(); // affiche Salut

$obj1->a="Bonjour";
$obj1->affiche(); // affiche Bonjour
$obj2->affiche(); // affiche Salut

modif($obj1); // Par copie !
$obj1->affiche(); // affiche Bonjour
$obj2->affiche(); // affiche Salut
```

De même si on créé une fonction qui doit modifier un objet, comme il est passé par défaut en copie, on ne modifie en fait que la copie locale de l'objet. Dans ce cas on le passe par référence.

```
 unction modif(&$objet)
{
      $objet->a='coucou';
}
$obj2=&$obj1; // Par référence
modif($obj1); // Par référence
$obj1->affiche(); // affiche coucou car on a modifié par référence
$obj2->affiche(); // affiche coucou
```

Donc, si on souhaite créer un code objet en PHP 4 (un peu) compatible avec les objets de PHP5, alors on doit effectuer des copies par référence, et passer les objets aux fonctions par référence. Le code modifié ci-dessus marche dans les deux cas.

15.11.3PHP

En PHP 5, les objets sont affectés par référence par défaut et de manière implicite. Ainsi on a

```
$obj1=new A;
$obj2=$obj1; // affectation par référence

$obj1->affiche(); // affiche salut
$obj2->affiche(); // affiche Salut

$obj1->a="Bonjour";
$obj1->affiche(); // affiche Bonjour
```

```
$obj2->affiche(); // affiche Bonjour

modif($obj1); // Par référence
$obj1->affiche(); // affiche coucou
$obj2->affiche(); // affiche coucou
```

Ce qui simplifie énormément la tkche des développeurs. Cependant, on peut explicitement demander la copie d'un objet à l'aide d'une méthode spéciale « **__clone** » et de l'instruction « **clone** ». Le clonage consiste à recopier tous les attributs d'une classe dans une autre. Le clonage peut être implicite (copie directe par un mécanisme interne de PHP) ou explicite (via la méthode __clone).

```
$obj3=clone $obj1; // on copie implicitement $obj1 dans $obj3
$obj1->a='Bonjour';
$obj3->a='Au revoir';
$obj1->affiche(); // Affiche bonjour
$obj3->affiche(); // Affiche Au revoir
```

Cette méthode fonctionne bien tant qu'on n'utilise pas des informations complexes comme des ressources ou des descripteurs de fichiers. Par exemple si dans $obj1 on avait ouvert une connexion sur une base de données, on a copié une ressource Base de données, mais elle serait inutilisable, car pas ouverte. Il faudrait la rouvrir avec « **__clone** » :

```
class Base {
      private $db;
       unction __construct() {
            $this->db=mysql_connect('localhost');
      }
       unction __clone() {
            $this->db=mysql_connect('localhost');
      }
}
$obj1=new Base;
$obj2=clone $obj1; // copie : __clone rouvre la connexion pour la copie
```

Note : la méthode « __clone » est appelée implicitement si le mot-clé « clone » est utilisé. On ne peut pas appeler explicitement cette méthode.

15.12Comparaison d'objets

15.12.1PHP 4

On peut effectuer des comparaisons d'objets en PHP4. Deux instances d'objets sont égales si et seulement si :

- elles proviennent de la même classe,
- elles sont les mêmes attributs,
- les attributs sont de même valeur.

On peut employer indifféré ent la comparaison avec « **==** » ou «**===** ».

```
 unction bool2str($bool) {
   if ($bool === false) {
           return 'FALSE';
   } else {
```

```
            return 'TRUE';
    }
}

 unction compareObjects(&$o1, &$o2) {
    echo 'o1 == o2 : '.bool2str($o1 == $o2)."\n";
    echo 'o1 != o2 : '.bool2str($o1 != $o2)."\n";
    echo 'o1 === o2 : '.bool2str($o1 === $o2)."\n";
    echo 'o1 !== o2 : '.bool2str($o1 !== $o2)."\n";
}

class Flag {
    var $flag;

    function Flag($flag=true) {
            $this->flag = $flag;
    }
}

class SwitchableFlag extends Flag {

    function turnOn() {
        $this->flag = true;
    }

    function turnOff() {
        $this->flag = false;
    }
}

$o = new Flag();
$p = new Flag(false);
$q = new Flag();

$r = new SwitchableFlag();

echo "Compare des instances créées avec les mêmes paramètres\n";
compareObjects($o, $q);

echo "\nCompare des instances créées avec différents paramètres\n";
compareObjects($o, $p);

echo "\nCompare une instance d'un parent avec celle d'une sous-classe\n";
compareObjects($o, $r);
```

Cet exemple va afficher :

```
Compare des instances créées avec les mêmes paramètres
o1 == o2 : TRUE
o1 != o2 : FALSE
o1 === o2 : TRUE
o1 !== o2 : FALSE

Compare des instances créées avec différents paramètres
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE

Compare une instance d'un parent avec celle d'une sous-classe
o1 == o2 : FALSE
o1 != o2 : TRUE
o1 === o2 : FALSE
o1 !== o2 : TRUE
```

15.12.2PHP

PHP 5 fait une différence entre la comparaison simple « **==** » et complète «**===** ».

Avec « **==** », PHP retournera vrai si les deux objets ont les mêmes attributs et valeurs, et s'ils sont des instances de la même classe, comme pour PHP 4.

Avec « **===** », PHP retournera vrai si les deux objets font référence au même objet de la même classe.

```
$o1=new A;
$o2=$o1; // référence
$o3= new A;

if($o1==$o2) echo "o1==o2<br />";
else echo "o1!==o2<br />";
if($o1===$o2) echo "o1===o2<br />";
else echo "o1!===o2<br />";
if($o1==$o3) echo "o1==o3<br />";
else echo "o1!==o3<br />";
if($o1===$o3) echo "o1===o3<br />";
else echo "o1!===o3<br />";
```

Va afficher

```
o1==o2 // o1 et o2 sont identiques (même classe, mêmes attributs et props.)
o1===o2 // o1 et o2 sont des références du même objet
o1==o3 // o1 et o3 sont identiques (même classe, mêmes attributs et props.)
o1!===o3 // o1 et o3 ne sont pas des références sur le même objet)
```

16 Nouveautés en PHP

16.1Les objets

16.1.1La fonction __autoload

La fonction « **__autoload** » indique à PHP comment réagir quand on tente de créer un objet depuis une classe qui n'a pas été définie. C'est une facilité offerte qui ne se substitue pas forcément aux require et include. Elle peut être utile par exemple pour éviter d'inclure l'ensemble des classes utilisées dans un projet. On préférera inclure la définition de la classe au moment de la création de son instance. La fonction « **__autoload** » ne prend qu'un paramètre, le nom de la classe. A vous de la progra er ensuite pour inclure le code nécessaire.

```
 unction __autoload($class) {
      require_once("/inc/{$class]}.php");
}
```

16.1.2Les classes abstraites

Les classes abstraites sont une nouveauté de PHP 5. Une classe abstraite est une classe sans implémentation (qui ne peut être instanciée), représentée par au moins une fonction abstraite (virtuelle pure, sans code), l'implémentation étant effectuée dans les classes dérivées.

Les classes abstraites servent essentiellement à définir une classe de base normalisée dont la structure pourra être reprise pour le développement d'autres classes effectives.

Toute classe qui contient au moins une méthode abstraite doit être définie comme abstraite.

Si la classe dérivée ne redéfinit pas une méthode abstraite de la classe parente (de base), elle devra être déclarée abstraite à son tour et ne pourra être instanciée.

La classe qui implémente la méthode abstraite doit être définie avec la même visibilité ou plus faible. Si la méthode abstraite est définie en tant que protégée, la fonction l'implémentant doit être définie en tant que protégée ou publique.

L'instruction pour définir des classes et méthodes est « **abstract** ».

```
// Classe abstraite : elle contient une méthode abstraite
abstract class AbstractClass {

   // Méthode abstraite qui devra être redéfinie dans la classe dérivée
   abstract protected function getValue();
   public function prt() {
     print $this->getValue();
   }
}

class ConcreteClass1 extends AbstractClass {
   // Redéfinition de la méthode abstraite de la classe de base
   protected function getValue() {
     return "ConcreteClass1";
   }
}
```

```
class ConcreteClass2 extends AbstractClass {
   // Redéfinition de la méthode abstraite de la classe de base
   protected function getValue() {
     return "ConcreteClass2";
   }
}

$class1 = new ConcreteClass1;
$class1->prt(); // affiche ConcreteClass1

$class2 = new ConcreteClass2;
$class2->prt(); // affiche ConcreteClass2
```

16.1.3Classes, méthodes et variables finales

PHP inclut un nouveau mécanisme de protection interdisant aux classes dérivées de réimplémenter une méthode ou une variable, ou à une classe d'être elle-même dérivée. Avec le mot-clé « **`final`** », on indique que la classe, la méthode ou la variable sont définitives et qu'elles ne pourront plus être implémentées. Voici un cas d'erreur :

```
class A {
    protected $val;
    function __construct() {
        $this->val=1;
    }

    final function affiche() {
        print $this->val;
    }
}

class B extends A {
    function __construct() {
        parent::_construct();
    }

    function affiche() { // ERREUR ! ON NE PEUT PAS REDEFINIR !
        print $this->val;
    }
}

$o1=new B;
$o1->affiche();
```

Dans la classe B nous avons tenté de récrire la méthode affiche déclarée en « **`final`** », et PHP nous retournera une erreur.

16.1.4Les interfaces objet

Les interfaces permettent de créer du code qui spécifie quelles méthodes et attributs une classe peut implémenter sans avoir à définir comment elles seront gérées. Toutes les méthodes spécifiées dans l'interface doivent être implémentées dans la classe qui la définit. L'interface est donc un moyen de définir les pré-requis obligatoires à la construction d'un modèle de classe en en donnant la liste des prototypes.

On créé une interface avec le mot-clé « **`interface`** ». Dans notre exemple, nous allons définir que toutes les classes qui implémentent notre interface doivent obligatoirement définir les fonctions affiche et incrémente.

```
interface Template
{
    public function affiche();
    public function incremente($cpt);
}
```

Attention : la déclaration des méthodes (notamment des paramètres) doit être rigoureusement identique dans la classe à celle de l'interface.

On indique qu'une classe implémente une interface avec le mot-clé « **implements** ».

```
class A implements Template {
    protected $val;
    function __construct() {
        $this->val=1;
    }

    function affiche() {
        print $this->val;
    }

    function incremente($cpt) {
        $this->val+=$cpt;
    }
}

$o1=new A;
$o1->incremente(2);
$o1->affiche();
```

Une classe peut implémenter plusieurs interfaces en même temps. Dans ce cas la liste des interfaces doit être précisée après le mot-clé « **implements** », et chaque interface séparée par des virgules.

```
interface Template  {
    public function affiche();
    public function incremente($cpt);
}

interface Template2 {
    public function decremente($cpt);
}

class A implements Template, Template2 {
    protected $val;
    function __construct() {
        $this->val=1;
    }

    function affiche() {
        print $this->val;
    }

    function incremente($cpt) {
        $this->val+=$cpt;
    }

    function decremente($cpt) {
        $this->val-=$cpt;
    }
}
```

16.1.5Gestion dynamique des méthodes et attributs

On ne peut toujours pas surcharger directement ses méthodes, attributs et opérateurs en PHP 5 comme en C++. Cependant de nouvelles fonctions permettent d'intercepter des appels à des éthodes ou attributs non prévus.

Note : Ces méthodes peuvent uniquement être déclenchées lorsque votre objet, hérité ou non, ne contient pas l'attribut ou la méthode que vous tentez d'accéder. Dans le cas contraire, la méthode ou l'attribut sont utilisés.

Pour une méthode, on utilise la méthode « **__call** » qui prend deux paramètres. Le premier est le nom de la méthode, le second est un tableau des paramètres passés à la méthode. Imaginons que nous souhaitons appeler la méthode « test » de notre objet, mais que nous n'avions pas prévu ce cas. Il suffit de gérer ce cas dans la méthode « **__call** » :

```
class A {
  function __call($m, $a) { // Gère les appels aux méthodes non définies
        if($m=="test") {
            echo "fonction test appelée avec les valeurs $a";
            return $a;
        }
  }
}

$o1 = new A();
$a = $o1->test(1, "2", 3.4, true); // test n'existe pas et pourtant...
print_r($a); // ça marche et ça retourne les paramètres de test.
```

Pour un attribut, on utilise les méthodes « **__get** » et « **__set** ». La méthode « **__get** » est appelée implicitement lorsque vous tentez d'accéder à la valeur d'une variable qui n'est pas définie dans votre classe. Elle prend comme unique paramètre le nom de l'attribut auquel vous tentez d'accéder. La méthode « **__set** » est appelée lorsque vous tentez d'affecter une valeur à un attribut qui n'est pas défini, et prend deux paramètres : le nom de la variable et la valeur affectée.

```
class A {
    private $tab=array();

    function __get($v) {
        if(!isset($this->tab[$v])) return false;
        else return $this->tab[$v];
    }

    function __set($v,$c) {
        $this->tab[$v]=$c;
    }
}

$o1 = new A();
$o1->var=1; // Le membre var n'existe pas et pourtant ...
echo $o1->var; // ça marche et ça affiche 1 !
```

16.1.6Les itérations d'objets

PHP 5 indexe les attributs de ses objets. Il est possible d'accéder à la liste des attributs (si la visibilité le permet) à l'aide du simple boucle itérative comme « **foreach** ».

```
class A {
    public $v1="salut";
    public $v2="les";
    public $v3="amis !";
    private $v4="privé";
}

$o1 = new A();
 oreach($o1 as $cle => $valeur) {
    echo "$cle = $valeur <br />\n";
}
```

Ce code affiche

```
v1 = salut
v2 = les
v3 = amis !
```

Et les attributs qui ne sont pas visibles ne sont pas affichés.

Il existe des interfaces prédéfinies permettant de créer sa propre classe d'itération. La classe « **`Iterator`** » implémente les cinq méthodes suivantes :

- `current`
- `next`
- `key`
- `valid`
- `rewind`

On peut donc créer une classe implémentant cette interface pour, par exemple, naviguer dans le résultat d'une requête de base de données, ou dans un tableau.

16.1.7Type hinting

PHP 5 permet de contrôler le type de valeur passé en paramètre à une fonction. Prenons le code suivant qui retourne les infos d'un utilisateur de classe « `utilisateur` » :

```
 unction detail($user) {
      return $user->infos();
}
```

La fonction attend un objet particulier de classe utilisateur. Tout fonctionnera à merveille si votre code est blindé. Cependant si on lui passe un entier, ça ne marchera pas, mais c'est seulement lors du « `return $user->infos()` » que PHP générera une erreur. Le « **`type hinting`** » indique à PHP ce qu'attend exactement la fonction.

```
 unction detail(utilisateur $user) {
      return $user->infos();
}
```

Cette fois PHP 5 retournera une erreur dès l'appel à la fonction si elle ne reçoit pas un objet de classe utilisateur.

16.2Les exceptions

16.2.1try ... catch

PHP 5 inclut enfin le support des exceptions avec le bloc « **try ... catch** ». Ainsi on peut placer du code « à risque » ou un code « exceptionnel » qui peut produire une erreur dans un bloc « **try** », et gérer l'erreur, selon son type, dans des blocs « **catch** ». On put utiliser plusieurs blocs « **catch** » suivant le type d'erreur généré. Attention, quand c'est possible on gérera les erreurs avec du code normal.

```
try {
       ... // Code à risque d'erreur
}
catch (Exception $e) {
      ... // Code réglant la situation exceptionnelle
}
```

Au sein du bloc « **try** », l'exécution d'une instruction erronée va déclencher l'émission d'un signal : une exception. Ce signal interrompt l'exécution du reste de tout le bloc. En fonction du signal, PHP exécutera le bloc « **catch** » approprié.

On remarque que dans le bloc « **catch** », PHP gère les exceptions à l'aide d'une classe spéciale appelée « **Exception** ». Une exception est en fait définie par un objet de cette classe ou d'une classe dérivée. On peut donc créer ses propres gestionnaires d'exceptions. Comme on peut écrire plusieurs blocs « **catch** » pour gérer les exceptions suivant le type d'objet.

16.2.2Déclenchement avec throw

On peut déclencher manuellement une exception à l'aide de l'instruction « **throw** ». Voici un code qui provoque toujours une exception :

```
try {
    $erreur="Message d'erreur";
    throw new Exception($erreur);
    echo 'Jamais exécuté';
} catch (Exception $e) {
   echo 'Exception: '.$e->getMessage().' à la ligne '.$e->getLine()."\n";
}
```

A l'aide de l'instruction « **throw** » nous avons provoqué l'émission d'une exception qui sera gérée par la classe « Exception ». Dans ce cas, PHP va rechercher le bloc « **catch** » correspondant au type d'objet créé pour gérer l'exception, ici «**catch(Exception $e)** ».

16.2.3classe Exception et classes personnalisées

La classe « **exception** » est imposée par PHP comme classe de base et parente de toute classe utilisable avec « **catch** ». La classe de base dispose des méthodes suivantes:

- **__construct($erreur,$code)** : le constructeur avec le message d'erreur et le code
- **getMessage()** : retourne le message d'erreur généré

- **getCode()** : Code erreur de l'exception
- **getLine()** : numéro de la ligne du script ayant provoqué l'exception
- **getFile()** : nom du script PHP ayant provoqué l'exception
- **getTrace()** et **getTraceAstring()** : gestion de la pile d'appels, un peu comme Java.
- **__toString()** : chaîne formatée à afficher.

Les méthodes « **getXXXX()** » sont finales. On ne peut donc pas les étendre. Mais le constructeur et la méthode « **__toString** » peuvent être étendus. On peut donc créer sa propre classe dérivée.

```
class MonException extends Exception {
   public function __construct($message="Erreur par défaut", $code = 0) {
       parent::__construct($message, $code);
   }
}
$a=1;
try {
    switch($a) {
        case 1: throw new MonException("Ouille !",$a);
            break;
        case 2: throw new Exception("Aie !",$a);
            break;
    }
}
catch(MonException $e) {
    echo $e->getMessage();
}
catch (Exception $e) {
   echo 'Exception: '.$e->getMessage().' ligne '.$e->getLine()."\n";
}
```

Si $a vaut 1, « **MonException** » sera envoyé, si $a vaut 2, ce sera « **Exception** ».

16.2.4PHP : Erreur ou Exception ?

Il se peut que dans certains cas, PHP déclenche naturellement une exception suivant l'exécution d'une fonction. Dans ce cas, ce sera indiqué dans la documentation PHP (voir site PHP) en ligne. Mais dans la plupart des cas, c'est l'ancien système de gestion des erreurs qui est utilisé. Ainsi le code suivant

```
try {
      $a=fopen('toto','r');
}
catch(Exception $e) {
         echo 'Exception: '.$e->getMessage();
}
```

ne donnera pas du tout le résultat escompté. L'exécution de « fopen » ne provoquera pas l'émission d'une exception mais une erreur PHP de type WARNING :

```
Warning: fopen(toto) [function.fopen]: failed to open stream: No such file or
```

```
directory in C:\apachefriends\xampp\htdocs\objet\panier.php on line 21
```

PHP gère plusieurs niveau d'erreurs : **WARNING**, **NOTICE**, etc. Par exemple, un problème d'ouverture de fichier émet une erreur de type WARNING, une erreur de syntaxe PARSE, l'utilisation d'une variable non déclarée un NOTICE. On peut régler et modifier les niveaux d'erreurs dans le php.ini, mais aussi dynamiquement via des fonctions.

Parmi ces fonctions, deux peuvent nous intéresser. « **set_error_handler** » permet de modifier le comportement par défaut de PHP lorsqu'il rencontre une erreur dans votre script. Au lieu d'exécuter son propre gestionnaire, il exécutera le votre. Attention, ce gestionnaire ne peut pas gérer certains types d'erreurs (syntaxe, etc, voir http://www.php.net/manual/en/function.set-error-handler.php), mais dans le cas d'un simple **WARNING** ou **NOTICE** il n'y a pas de problème. On lui passe en paramètre le nom de sa propre fonction qui doit gérer l'erreur. Les paramètres sont assez parlants. Dans notre fonction, nous allons déclencher une exception à la réception de n'importe quelle erreur.

```
 unction gest_erreur($errno, $errstr, $errfile, $errline)
{
    throw new Exception("$errstr line $errline",$errno);
}
set_error_handler("gest_erreur");
```

A l'exécution de notre bloc **« try...catch** », nous aurons cette fois

```
Exception: fopen(toto) [function.fopen]: failed to open stream: No such file or
directory line 21
```

ce que nous voulions : nous avons bien déclenché une exception.

16.2.5Exception par défaut

Si certaines instructions PHP génèrent des exceptions et que ces instructions ne sont pas au sein d'un bloc « **try...catch** », PHP générera une exception par défaut avec son propre gestionnaire d'exceptions. On peut modifier cet état avec la fonction « **set_exception_handler** ».

```
 unction exception_handler($exception) {
  echo "Exception imprévue : " , $exception->getMessage(), "\n";
}

set_exception_handler('exception_handler');
```

17Sauvegarder ses objets

17.1Cas d'une session

Les informations indiquées ici fonctionnent aussi bien en PHP 4 qu'en PHP 5. Il faut distinguer le cas des sessions où il faut passer un objet d'une page à l'autre et le cas d'une sauvegarde d'un objet sur disque.

Dans le premier cas, si les sessions sont utilisées les objets peuvent être passés directement d'une page à une autre en respectant quelques règles :

- La session doit être ouverte sur chaque page par la fonction « **session_start()** » avant l'utilisation de l'objet

- Les classes doivent être définies avant l'ouverture de session par session_start() et l'appel à l'objet, ce qui implique soit de répéter cette déclaration dans chaque page, soit de la placer dans un fichier à inclure avec « **reuire()** » ou « **reuire_once()** » en début de script.

- L'objet doit être une variable de session stockée soit par la fonction « **session_register()** », soit à l'aide du tableau global « **$_SESSION[]** ».

Voici un exemple :

objet.inc :

```
<?
class Panier {
      // Eléments de notre panier
      var $contenu;

      // Ajout de $qte articles de type $code au panier
      function ajout_article ($code, $qte) {
               if(isset($this->contenu[$code])) $this->contenu[$code] += $qte;
               else $this->contenu[$code] = $qte;
      }

      // Suppression de $num articles du type $artnr du panier
      function suppr_article ($code, $qte) {
               if ($this->contenu[$code] > $qte) {
                  $this->contenu[$code] -= $qte;
                  return TRUE;
               } else {
                 return FALSE;
               }
      }
      function liste_article() {
               foreach($this->contenu as $ref => $qte) echo "$ref=$qte <br />";
      }
}

class Panier_nomme extends Panier {
   var $proprietaire;

   function nom_proprietaire ($nom) {
       $this->proprietaire = $nom;
   }
}
```

```
?>
```

objet.php :

```
<?php
require_once("objet.inc");
session_start();
?>
<html>
<head>
<title>Page 1</title>
</head>
<body>
<?
$_SESSION['objet']=new Panier_nomme;
$_SESSION['objet']->nom_proprietaire("seb");
$_SESSION['objet']->ajout_article("ref01",3);
$_SESSION['objet']->suppr_article("ref01",1);
$_SESSION['objet']->ajout_article("ref02",3);
echo "{$_SESSION['objet']->proprietaire}<br />";
?>
<a href="objet2.php">Page 2</a>
</body>
</html>
```

objet2.php :

```
<?php
require_once("objet.inc");
session_start();
?>
<html>
<head>
<title>Page 2</title>
</head>
<body>
<?
echo $_SESSION['objet']->proprietaire."<br />";
$_SESSION['objet']->liste_article();
?>
</body>
</html>
```

17.2Autres cas

Rien n'empêche de conserver un objet pour le récupérer plus tard, même après avoir fermé une session. Pour ça on peut le stocker dans un fichier ou en base de données. Pour récupérer un format correct de données pour stockage, il faut utiliser les fonctions « **serialize() »** et « **unserialize() »**.

La première fonction retourne une chaîne de texte représentant la variable (pas forcément un objet) passée en paramètre. Ce format de donnée peut ainsi être stocké dans la manière qui vous convient.

Le seconde fonction prend comme paramètre la chaîne de texte issue de serialize et retourne la variable du type originel, dans notre cas un objet.

classa.inc:

```
<?php
  class A {
   var $one = 1;
   function show_one() {
     echo $this->one;
```

```
    }
  }
?>
```

page1.php:

```
<?php
  include("classa.inc");
  $a = new A;
  $s = serialize($a);
  // enregistrez $s où la page2.php pourra le trouver.
  $fp = fopen("store", "w");
  fputs($fp, $s);
  fclose($fp);
?>
```

page2.php:

```
<?php
  // Ceci est nécessaire pour que unserialize() fonctionne correctement
  include("classa.inc");
  $s = implode("", @file("store"));
  unserialize($s);
  // maintenant, utilisez la méthode show_one de l'objet $a.
  $a->show_one();
?>
```