Collège Technique **Aumôniers du Travail**
Enseignement de **promotion sociale**
185 Grand'rue 6000 Charleroi
Tel. 071.285.905

Projet de développement Web

Code de l'unité de formation : ***753430D32D1***

Partie(s) concernée(s) : ***Php***

Domaine de formation : 701

- ***Bachelier en informatique de gestion***

Nom du chargé de cours : ***Mr De Guglielmo***

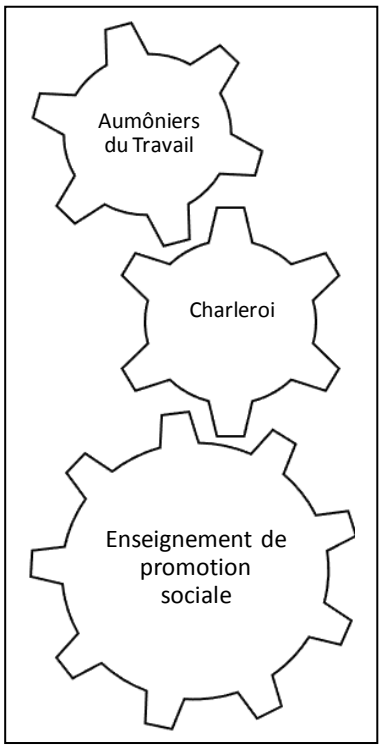

Table des matières

1 Introduction 4
1.1 Les commentaires 4

2 Algorithmique 4
2.1 Les variables 4
2.2 Les séquences 5
2.3 Les répétitives 5
2.4 Les alternatives 5

3 Les variables 7
3.1 Portée des variables 7
3.2 Le mot-clé global 8
3.3 Utilisation des variables static 8
3.4 Les constantes 10

4 L'instruction echo 11

5 Guillemets ou apostrophes 12
5.1 Ce qui n'est pas conseillé 12
5.2 Ce qui est conseillé 12

6 Les formulaires HTML 13
6.1 Les formulaires et php 15
6.2 Un formulaire et son traitement 16

7 Les tableaux 17
7.1 Les tableaux simples 17
7.2 Les tableaux associatifs 18
7.3 La boucle foreach 19
7.4 Boucle foreach et variables POST 20

8 Les opérateurs, les fonctions, for et switch 23
8.1 Les opérateurs 23
8.2 Les fonctions 24
8.2.1 Les fonctions dans la page 24
8.2.2 Plusieurs paramètres passés à la fonction 26

8.2.3 Les fonctions dans un fichier séparé 26
8.2.4 Fonction qui renvoie une valeur de retour 27
8.3 La boucle for 27
8.4 L'instruction switch 28
9 Gestion des sessions 30
9.1 Passer l'identifiant de session (session ID) 30
9.2 Cookies 31
10 Les bases de données 32
10.1 Introduction 32
10.2 Alimenter sa base de données via php 33
10.2.1 Présentation du formulaire 33
10.2.2 Connexion à la base de données en php 33
10.2.3 Pour travailler proprement 34
10.2.4 Insérer des données dans notre base de données via php 34
10.2.5 Synthèse 35
10.3 Récupérer les données de sa base de données via php 36
10.3.1 Pour travailler proprement 36
10.3.2 Le code pour 'récupérer toutes les filles' 36
11 PHP Data Objects 38
11.1 Connexions et gestionnaire de connexion 38
11.2 Transactions et validation automatique (autocommit) 39
11.3 Requêtes préparées et procédures stockées 40
12 Fonctions de gestion des variables 42
12.1 Sommaire 42
13 Fonctions sur les tableaux 43
13.1 Sommaire 43
14 Fonctions sur les chaînes de caractères 45
14.1 Sommaire 45
15 Fonctions diverses 47
15.1 Sommaire 47

16 Les Superglobales.....48
16.1 Description 48

1 Introduction

Vous avez de bonnes notions de **HTML** et **CSS**, vous savez donc que ces deux langages ne sont pas des langages de programmation, mais des langages de simple affichage statique.

Facile d'afficher pour vous une page web qui dit : "bonjour, on est lundi." Le seul souci, c'est que l'on sera toujours lundi sur votre page... Pas très dynamique tout ça.

C'est là qu'intervient PHP qui est un langage de programmation web. Il produit du code HTML.

En quoi est-il utile ?

Parce que le code HTML que produit PHP change en fonction des circonstances que vous avez programmées. On dira qu'il introduit du dynamisme dans la page web.

Le code PHP que vous avez inséré dans vos pages agit à chaque chargement (et donc rafraîchissement) de page web. Ceci pour relativiser son "dynamisme".

L'interactivité avec l'utilisateur se limite à certaines actions de l'utilisateur.

Un rafraîchissement de page, c'est par exemple le clic de l'utilisateur sur le bouton 'submit' inclut dans une balise form, ou bien une première arrivée sur une page web.

1.1 Les commentaires

Au sein de votre code, il est bon de poser quelques commentaires, quelques lignes d'explication qui ne seront bien entendu pas considérées par la machine comme des lignes de programmation à exécuter mais des lignes qu'elle pourra ignorer, et qui ne sont destinées qu'au programmeur, qui remet le nez dans son programme parfois dix ans après et ne sait plus pourquoi ou comment il a conçu son code.

Parfois, c'est une autre personne qui hérite du programme. Dans tous les cas, il faut donc commenter son code.

Deux sortes de syntaxe pour des commentaires en PHP

- Petit commentaire sur une seule ligne // en début de ligne
- Commentaire sur plusieurs lignes /* au début et */ en dernier

Comment rédiger des commentaires dans son code

```
<?php

/*

Commentaire

sur plusieurs

lignes

*/


// Commentaire sur une ligne

// Autre commentaire sur une autre ligne

?>
```

2 Algorithmique

2.1 Les variables

Un exemple suffira pour introduire le sujet. Tout le monde a déjà eu entre les mains une publicité faussement personnalisée du type :

”Bonjour Madame Michu, vous avez gagné notre canapé extra-cuir lors du tirage du 20/12/2008.”

Il est clair que l’entreprise qui envoie cette pub possède une base de données avec des tas de noms. Et que c’est un programme automatique qui stocke les valeurs de Madame, de Michu, et la date dans des variables. Le reste est du texte figé. On va dire que civilité, nom et date sont des variables.

2.2 Les séquences

Programmer, c’est donner une suite d’instructions à la machine. Exactement comme si l’on construisait une recette de cuisine. Sauf que l’on écrit la recette à l’usage d’une machine qui ne comprend que le binaire et qu’une instruction à la fois. On avance, instruction par instruction, de façon linéaire. Nous dirons de manière séquentielle.

Dans la structure d’un programme, deux types de construction peuvent s’éloigner de cette linéarité :

2.3 Les répétitives

Dans notre recette de cuisine, nous avons une action affreusement répétitive découpez les trois plaques de chocolat en carrés individuels. En programmation, on ne va pas dire 150 fois, détachez le carré de chocolat, détachez le carré de chocolat, détachez le carré de chocolat... Alors nous allons faire ce que l’on nomme une boucle (il y en a de plusieurs sortes).

```
tant qu’il y a une plaquette de chocolat{
détache le carré de chocolat
}
```

Le terme ”boucle” est utilisé parce que le programme lit la condition, entre dans le bloc d’instructions après la première accolade (si la condition est vraie) et ”boucle” tant que la condition est vraie...

2.4 Les alternatives

Nous avons une condition. Restons sur notre recette de mousse au chocolat. Si j’ai 6 gourmands, je casse 8 œufs, si j’ai 6 personnes au régime, je casse 4 œufs, mais par défaut, une bonne mousse, c’est 6 œufs pour 6...

Ces conditions s’excluent l’une l’autre

```
Sors le plat.
si gourmand{
casse 8 œufs
}
sinon si régime{
casse 4 œufs
}
sinon{
```

```
casse 6 œufs
}
Bien mélanger...
```

Cette fois, vous comprendrez que le cheminement logique linéaire se sépare en trois voies parallèles, un peu comme des rails, puis se regroupent à nouveau à la fin. Le programme ne doit passer que par une et une seule des trois voies.

Comment ça marche ?

```
<html>
	<head>
		<title>Ma page de test</title>
	</head>
	<body>
		<h1>Bienvenue sur le site de toto </h1>
		<p>Le blabla de ma page...</p>
	</body>
</html>
```

Voici une page HTML bien traditionnelle comme on les aime. Pour l'instant, enregistrez ce fichier sur votre bureau, et double-cliquez maintenant directement sur votre fichier test.html, cela vous ouvrira une page web comme si vous étiez sur internet...

Pourtant, vous n'y êtes pas, sur internet. Vous êtes "en local", bref, sur votre ordinateur, et vous ne faites que voir votre fichier à la façon d'une page web. Ouvrez à nouveau l'éditeur de texte et copiez/collez le bout de code suivant dans lequel on a introduit des balises PHP.

```
<html>
	<head>
		<title>Ma page d'accueil </title>
	</head>
	<body>
		<h1>Bienvenue sur le site de toto </h1>
		<p> Toto fait de l'anglais :</p>
```

```
            <?php
                  echo '<p>Hello ! What is the day today ?
                                                It is '.date("l").' !</p>';
            ?>
      </body>
</html>
```

Quand vous l'enregistrez, sur votre bureau, renommez-le test.php.

Ainsi, le serveur php est avisé que cette page contient du code PHP. Fermez votre éditeur. Double-cliquez directement sur votre page test.php : Quelle déception ! Cela ne vous ouvre pas une page web mais... le code que vous venez de quitter, sous un éditeur de texte qui est probablement bloc-notes sans coloration syntaxique.

C'est ici que le fait d'avoir installé un serveur web interprétant correctement PHP sur votre machine va vous être utile. En effet, il vous faut un interpréteur PHP installé en local. Le serveur web appelle le composant PHP puis il va droit aux balises PHP de ce fichier, et interprète ce qu'il y a dedans : Ici, il affiche (**echo**) tout ce qui est entre parenthèses, puis concatène ou colle si vous voulez (le point) le résultat de la fonction date qui renvoie le jour en anglais...

Il a donc interprété du PHP pour le traduire en HTML. Ensuite, il repasse le fichier au navigateur, qui lui ne sait afficher QUE du HTML (ou du CSS)... et bien ça tombe bien, il ne voit que du HTML dorénavant...

Si vous revenez demain sur cette page, la fonction date affichera un autre jour dans le HTML. Voici pour l'aspect technique.

Maintenant que tout tourne, notre première étape portera sur les variables et sur la structure **echo** qui signifie donc "affiche" Voici quelques exemples... que vous testerez en les rajoutant sur votre fichier au sein des balises PHP bien entendu.

3 Les variables

Un signe $ devant une chaîne de caractères signifie que nous parlons d'une variable, on la nomme comme on veut ensuite, mais jamais d'accent ni d'espace dans un nom de variable.

Une variable PHP n'est pas typée mais peut prendre 4 types (dans l'ordre ci-dessous) :

Les types des variables

- string (chaîne de caractères)
- integer (nombre entier)
- bool (booléen qui signifie logique binaire genre vrai ou faux)
- float (nombre réel)

```php
<?php

	$nom = 'Mickaël';

	$age = 17;

	$gars = true;

	$taille = 1.75;

?>
```

Étudiez bien chaque détail

- Toutes les instructions se terminent par un point virgule.
- Seule une variable de type string exige des guillemets (simples ou doubles, préférez les apostrophes pour l'instant, nous verrons pourquoi un peu plus loin)...
- Les variables booléennes prennent soit la valeur **`true`**, soit la valeur **`false`**.
- Le nombre réel prend un point à l'anglo-saxonne et non une virgule.

N'hésitez pas à personnaliser ces exemples et à les triturer en en changeant les données pour les assimiler.

3.1 Portée des variables

La portée d'une variable dépend du contexte dans lequel la variable est définie. Pour la majorité des variables, la portée concerne la totalité d'un script PHP. Mais, lorsque vous définissez une fonction, la portée d'une variable définie dans cette fonction est locale à la fonction. Par exemple :

```php
<?php

	$a = 4;

	function test()

	{

		echo 'a = '.$a;

	}

	test();

?>
```

Le script n'affichera rien à l'écran car l'instruction **echo** utilise la variable locale *$a*, et celle-ci n'a pas été assignée préalablement dans la fonction. Vous pouvez noter que ce concept diffère un petit peu du langage C dans lequel une variable globale est automatiquement accessible dans les fonctions, à moins d'être redéfinie localement dans la fonction. Cela peut poser des problèmes si vous redéfinissez des variables globales localement. En PHP, une variable globale doit être déclarée à l'intérieur de chaque fonction afin de pouvoir être utilisée dans cette fonction.

3.2 Le mot-clé *global*

```php
<?php

	$a=2;

	$b=1;

	function somme()

	{

		global $a, $b;

		$b = $a + $b;

	}

	somme();

	echo 'b = '.$b;

?>
```

Le script ci-dessus va afficher la valeur *3*. En déclarant globales les variables *$a* et *$b* locales de la fonction `somme()`, toutes les références à ces variables concerneront les variables globales. Il n'y a aucune limite au nombre de variables globales qui peuvent être manipulées par une fonction.

Une deuxième méthode pour accéder aux variables globales est d'utiliser le tableau associatif prédéfini *$GLOBALS*. Le précédent exemple peut être réécrit de la manière suivante :

```php
<?php

	$a = 1;

	$b = 2;

	function somme()

	{

		$GLOBALS['b'] = $GLOBALS['a'] + $GLOBALS['b'];

	}

	somme();

	echo 'b = '.$b;

?>
```

Le tableau `$GLOBALS` est un tableau associatif avec le nom des variables globales comme clé et les valeurs des éléments du tableau comme valeur des variables. Notez que `$GLOBALS` existe dans tous les contextes, car `$GLOBALS` est un superglobal.

L'utilisation du mot clé `global` à l'extérieur d'une fonction n'est pas une erreur. Il peut être utilisé si le fichier est inclus depuis l'intérieur d'une fonction.

3.3 Utilisation des variables *static*

Une autre caractéristique importante de la portée des variables est la notion de variable *static*. Une variable statique a une portée locale uniquement, mais elle ne perd pas sa valeur lorsque le script appelle la fonction. Prenons l'exemple suivant :

Exemple : Les variables statiques

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

Cette fonction est un peu inutile car à chaque fois qu'elle est appelée, elle initialise *$a* à *0* et affiche "*0*". L'incrémentation de la variable (*$a*++) ne sert pas à grand chose, car dès que la fonction est terminée, la variable *$a* disparaît. Pour faire une fonction de comptage utile, c'est-à-dire qui ne perdra pas la trace du compteur, la variable *$a* est déclarée comme une variable statique :

Exemple : Les variables statiques (2)

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

Maintenant, la variable *$a* est initialisée uniquement lors du première appel à la fonction et, à chaque fois que la fonction *test()* est appelée, elle affichera une valeur de *$a* incrémentée de 1.

Les variables statiques sont essentielles lorsque vous faites des appels récursifs à une fonction. Une fonction récursive est une fonction qui s'appelle elle-même. Il faut faire attention lorsque vous écrivez une fonction récursive car il est facile de faire une boucle infinie. Vous devez vérifier que vous avez bien une condition qui permet de terminer votre récursivité. La fonction suivante compte récursivement jusqu'à 10, en utilisant la variable *$count* pour savoir quand il faut s'arrêter :

Exemple : Les variables statiques et la récursivité

```
<?php
function test()
{
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        test();
    }
    $count--;
}
?>
```

Note:

Les variables statiques doivent être déclarées comme dans l'exemple ci-dessus. Tenter d'assigner des valeurs à ces variables qui sont le résultat d'expressions causera une erreur d'analyse.

Exemple : Déclaration de variables statiques

```
<?php
function foo(){
    static $int = 0;          // correct
    static $int = 1+2;        // faux  (car c'est une expression)
    static $int = sqrt(121);  // faux  (car c'est une expression)

    $int++;
    echo $int;
}
?>
```

Note:

Les déclarations statiques sont résolues au moment de la compilation.

3.4 Les constantes

Seuls les types de données scalaires peuvent être placés dans une constante : c'est à dire les types booléen, entier, double et chaîne de caractères (soit bool, entier, double et string.)

Vous pouvez accéder à la valeur d'une constante en spécifiant simplement son nom. Contrairement aux variables, vous ne devez *PAS* préfixer le nom de la constante avec *$*. Vous pouvez aussi utiliser la fonction constant(), pour lire dynamiquement la valeur d'une constante, dont vous obtenez le nom dynamiquement (retour de fonction, par exemple). Utilisez la fonction get_defined_constants() pour connaître la liste de toutes les constantes définies.

Il y a des différences entre les constantes et les variables :

- Les constantes ne commencent pas par le signe (*$*).
- Les constantes ne peuvent être définies qu'en utilisant la fonction define() ou le mot-clé *const*, pas par simple assignement.
- Les constantes sont définies et accessibles à tout endroit du code, globalement.
- Les constantes ne peuvent pas être redéfinies ou indéfinies une fois qu'elles ont été définies.
- Les constantes ne peuvent contenir que des scalaires.

Exemple : Définir une constante

```php
<?php
  define("CONSTANTE", "Bonjour le monde.");
  echo CONSTANTE; // affiche "Bonjour le monde."
  echo Constante; // affiche "Constante" et une notice.
?>
```

Exemple : Définir des constantes en utilisant le mot-clé *const*

```php
<?php
// Fonctionne depuis PHP 5.3.0.
const CONSTANT = 'Hello World';

echo CONSTANT;
?>
```

Note:

Contrairement aux constantes définies en utilisant l'instruction define(), les constantes définies en utilisant le mot-clé *const* doivent être déclarées au plus haut niveau du contexte, car elles seront définies au moment de la compilation. Cela signifie qu'elles ne peuvent être déclarées à l'intérieur de fonctions, boucles ou instructions *if*.

4 L'instruction echo

Bien maintenant que nous avons défini le type des variables en même temps que nous leur avons affecté une valeur, affichons les grâce à echo, dans un environnement HTML :

```
<?php

	echo '<p>Bonjour à tous.<br />

	Mon vrai nom n\'est pas Toto.<br />

	Mon vrai nom est '.$nom.'<br />

	J\'ai '.$age.' ans et je mesure '.$taille.'m.<br />

	Et comme mon nom l\'indique, je suis ';

	if ($gars == true){

		echo 'un garçon.</p>';

	}

	else{

		echo 'une fille.</p>';

	}

?>
```

Commentaires sur cette instruction **echo** :

Ici il n'y a que deux instructions 'affiche' : Une automatique, et une conditionnelle.

Vous remarquerez que

- on passe à la ligne au sein de l'affichage grâce à la balise `<br />`.
- toutes les apostrophes qui sont seulement du texte ont été neutralisées en plaçant \ devant chacune d'elle pour qu'elles ne soient pas considérées comme la fin de la chaîne de caractères.
- l'affichage du contenu de la variable se fait automatiquement, en mettant la variable nue dans le code.
- le point sert à concaténer (ou coller du texte bout à bout).
- il vous faut gérer les espaces à l'affichage au sein des guillemets. Ici la coloration syntaxique devient indispensable.
- enfin, la condition (si c'est un garçon ou une fille), bien qu'écrite à la ligne dans le code et ouvrant une seconde instruction **echo**, s'affiche tout de même dans la continuité du précédent **echo**...

Autrement dit, c'est toujours au niveau du code HTML que se gèrent les retours à la ligne de l'affichage (balises `<br />`, balises `<p>` ou autres).

Les retours à la ligne que vous faites spontanément dans votre code PHP après chaque instruction ne sont que des retours à la ligne à destination du programmeur pour une meilleure lisibilité de son code

5 Guillemets ou apostrophes

5.1 Ce qui n'est pas conseillé

Vous trouverez certainement bien pratique de savoir que les guillemets ” ” (contrairement aux guillemets simples ou apostrophes) permettent ceci en PHP:

```php
<?php

	$age = 18;

	//Ceci affichera directement

	//J’ai 18 ans.

	echo "J’ai $age ans.";

?>
```

Alors pourquoi utiliser les guillemets simples pour PHP, puisque l’exemple ci-dessus règle le problème de l’interprétation directe de la variable et le problème de l’apostrophe du texte ”J’ai”...

5.2 Ce qui est conseillé

A l’avenir, vous allez fréquemment produire du HTML avec votre code PHP, puisque c’est sa fonction essentielle. Ainsi en adoptant la convention suivante, vous saurez toujours si vous êtes au niveau de PHP ou bien de HTML.

```php
<?php

	echo ’<div class="contenu">blablabla</div>’;

?>
```

Quand vous utilisez un ’ : on est au niveau de PHP.

Quand vous utilisez un ” : on est au niveau du HTML.

Pour la clarté de vos idées, il est donc conseillé ceci :

```php
<?php

	$age = 18;

	echo ’J\’ai ’.$age.’ ans.’;

?>
```

Bien sûr, il vous restera à avoir le réflexe suivant : penser au caractère d’échappement.

6 Les formulaires HTML

Vous le savez, un formulaire en HTML, c'est la suite de balises suivante :

```
<form name="inscription" method="post" action="saisie.php">

      Entrez votre pseudo : <input type="text" name="pseudo" /><br />

      Entrez votre ville : <input type="text" name="ville" /><br />

      <input type="submit" name="valider" value="OK" />

</form>
```

Ici, ce formulaire présente une zone de saisie pour entrer son pseudo, va à la ligne, une zone de saisie pour entrer sa ville, va à la ligne, et enfin un bouton pour valider sur lequel sera écrit 'OK'...

Le but, vous l'aurez compris, c'est de récupérer, via PHP, les infos entrées par n'importe quel usager. Voici comment les choses vont se découper :

Les attributs dans la balise `<form>` précisent le nom du formulaire, puis précisent que les variables contenues dans ce formulaire seront envoyées par la méthode **`post`** (au moment où l'utilisateur cliquera sur le bouton "**`submit`")

Quelles variables y aura-t-il, que comporteront-elles et surtout, quels noms porteront ces variables?

Il y aura la variable `$_POST['pseudo']`, qui constitue le texte entré dans la zone pseudo par l'utilisateur avant d'avoir cliqué sur submit... La formule est immuable : `$_POST['name']` pour chaque input, car les variables sont automatiquement nommées ainsi.

Pareil donc pour `$_POST['ville']`.

Enfin `$_POST['valider']` sera la variable qui dira, si elle existe bien entendu, qu'il y a eu clic sur la validation, et si elle n'existe pas, qu'il n'y a pas eu d'événement clic...

Récupérer l'information :

```
<html>

      <head>

            <title>Ma page d'accueil</title>

      </head>

      <body>

            <h1>Bienvenue sur le site de toto</h1>

            <h2>Commencez-donc par vous inscrire :</h2>

            <form name="inscription" method="post" action="saisie.php">

                  Entrez votre pseudo : <input type="text" name="pseudo"
/><br />

                  Entrez votre ville : <input type="text" name="ville"
/><br />
```

```
                <input type="submit" name="valider" value="OK" />
            </form>
        </body>
</html>
```

En PHP, il faut toujours commencer par classer vos idées ainsi :

Quelle est la condition pour que mon code s'exécute ? Autrement dit, quelle action de l'utilisateur va déclencher mon code...

Tout cela sera donc dans une condition, voyons donc immédiatement la syntaxe du if :

L'instruction if

```
if(user a cliqué sur valider){
        récupère la variable pseudo;
        récupère la variable ville;
        écris 'Salut (son pseudo) de (sa ville). Bienvenue sur mon site !';
}
```

Voilà donc la structure d'une condition...

On entoure la condition entre des parenthèses, on encadre toutes les instructions dans des accolades. Et encore une fois bien sûr, chaque instruction se termine toujours par un point-virgule.

Oubliez un détail de ce genre, et plus rien ne fonctionne. Tout en restant près de notre formulaire, imaginons que nous compliquons un peu notre condition : Dans le premier si (il a cliqué), nous aimerions dire qu'en plus, si sa ville est Paris, nous lui proposons de le rencontrer... Ce sera un 'si' imbriqué.

Même principe, mais cette fois, pensez à indenter votre code dès son élaboration, pour augmenter la clarté de celui-ci.

Une condition imbriquée :

```
if(user a cliqué sur valider){
        récupère la variable pseudo;
        récupère la variable ville;
        écris 'Salut (son pseudo) de (sa ville). Bienvenue sur mon site !';
        if(sa ville) == 'Paris' {
                écris 'On est plusieurs de Paris sur le site.';
                écris 'Si tu veux qu'on se voie, contacte-nous !';
        }
}
```

Remarquez une chose essentielle :

`if(sa ville) == 'Paris'` comporte deux signes =.

Cela est indispensable pour une comparaison. En effet, la variable ne reçoit pas 'Paris' (signe = d'affectation), elle est comparée (==) à 'Paris'.

Vous ferez forcément cette erreur au début, d'autant plus difficile à repérer qu'elle ne provoquera pas de message d'erreur : Le programme comprendra "Si la variable reçoit Paris", et ce sera toujours vrai. Autrement dit, votre programme ne fera pas ce que vous voulez, mais le langage ne pourra pas vous préciser votre bug.

Une dernière chose pour les **if**, s'il y a un **if**, il faut parfois un ou plusieurs **else**, puis un **else** pour conclure en tout dernier... Restons toujours dans notre thématique formulaire : Un formulaire m'a renvoyé l'âge entré par l'utilisateur :

```php
<?php

$age = $_POST['age'];

?>
```

La syntaxe donnera ceci pour des conditions successives qui s'excluent l'une, l'autre :

```php
<?php

if($age < 5){

   $verdict = 'Ouh le bébé !';

}

else if($age < 13){

            $verdict = 'Vous êtes un enfant !';

      }

      else if($age < 18){

                  $verdict,= 'Vous êtes un(e) ado !';

            }

            else{

                  $verdict = 'Ah ! enfin un(e) adulte !';

            }

echo $verdict;

?>
```

Ici, pas d'imbrication, puisque ce sont des conditions successives et non imbriquées l'une dans l'autre.

Dans tous les cas, il est affecté une valeur à $verdict au moment où il parvient à l'instruction 'affiche moi le verdict'. L'instruction n'est pas dans une condition, elle s'exécute donc toujours. Mais la variable $verdict elle, n'est pas initialisée de la même façon selon que ce soit l'une ou l'autre des 4 conditions qui est satisfaite.

En d'autres termes, dès l'instant où le programme tombe sur une condition qui se réalise, il cesse de tester la suite et passe à l'exécution de l'instruction qui suit le bloc alternatif.

6.1 Les formulaires et php

Revenons maintenant sur le traitement de notre formulaire de départ, avec une toute petite

condition bien classique :

```
if(user a cliqué sur valider){

      récupère la variable pseudo;

      récupère la variable ville;

      écris 'Salut (son pseudo) de (sa ville). Bienvenue sur mon site !';

}
```

Comment allons-nous traduire cela en PHP? De la façon suivante :

```
<?php

if(isset($_POST['valider'])){

      $pseudo = $_POST['pseudo'];

      $ville = $_POST['ville'];

      echo 'Salut '.$pseudo.'de '.$ville.'<br />Bienvenue sur mon site !';

}

?>
```

Traduction :

S'il existe une variable `$_POST` qui a pour nom '`valider`' et que donc user a validé, `$pseudo` reçoit la variable `$_POST` qui a pour nom '`pseudo`', `$ville` reçoit la variable `$_POST` qui a pour nom '`ville`'

Remarque :

On peut très bien se passer de ranger la variable de type `$_POST['name']` dans une variable au nom simplifié de type `$name`, et la manipuler directement !

Mais si nous devons la manipuler sans arrêt, il est plus agréable (et plus sûr) de la renommer.

Autre remarque :

Puisque c'est l'attribut `name` dans la balise d'un objet type formulaire qui va déterminer le nom de la variable `$_POST`, pensez-bien à ne pas y mettre d'accent, ni d'espace...

6.2 Un formulaire et son traitement

```
<html>
	<head>
		<title>Ma page d'accueil</title>
	</head>
	<body>
		<h1>Bienvenue sur le site de toto </h1>
		<h2>Commencez-donc par vous inscrire :</h2>
		<form name="inscription" method="post" action="saisie.php">
			Entrez votre pseudo : <input type="text" name="pseudo" />
<br />
			Entrez votre ville : <input type="text" name="ville"
/><br />
			<input type="submit" name="valider" value="OK" />
		</form>

		<?php
			if(isset($_POST['valider'])){
				$pseudo=$_POST['pseudo'];
				$ville=$_POST['ville'];
				echo 'Salut '. $pseudo.'de '. $ville
					.'<br/>Bienvenue sur mon site !';
			}
		?>
	</body>
</html>
```

Conseil : EXPERIMENTEZ!

Vous expérimenterez tout ça en local bien entendu... Changez des détails, triturez, appropriez-vous la méthode et la syntaxe.

7 Les tableaux

Vous avez un formulaire qui propose des saisies au client (client au sens informatique du mot, par opposition à serveur). Disons une dizaine.

Vous vous en souvenez, cela créera une dizaine de variables $_POST[’nom_de_l_input’].

Il se trouve que PHP range automatiquement ces variables dans un tableau. Nous y reviendrons à la fin de cette étape.

Commençons par voir ce qu’est un tableau :

7.1 Les tableaux simples

Un tableau, c’est un moyen de stocker plusieurs variables, selon un plan qui vous paraît logique. C’est comparable à un meuble avec ses tiroirs.

Dans le tiroir 0, (oui, le tableau commence par le tiroir zéro), vous rangez la variable dimanche par

exemple, dans le tiroir 1, vous rangez la variable lundi, etc.

En informatique, on appelle index ou indice le numéro de tiroir (la position de la variable dans le tableau), et valeur, la valeur de la variable entreposée.

Voici la syntaxe d’un tableau tout simple.

On construit le tableau des jours de la semaine

```
<?php

$semaine = array('dimanche', 'lundi', 'mardi', 'mercredi', 'jeudi',

                                                    'vendredi', 'samedi');

?>
```

Par cette simple ligne, vous venez de construire un tableau (qui vous le remarquerez, est une variable en elle-même puisqu’elle commence par `$`, mais une variable complexe, organisée). `$semaine` est le nom du tableau entier.

Vous lui avez affecté des valeurs, (via la commande **`array`**, l’interpréteur sait que c’est un tableau).

Et ici, par défaut, l’index commence à 0, donc dimanche aura pour index... 6 et non 7.

Une fois construit ce tableau, comment invoquer une valeur ?

`$semaine[2]` sera...mardi et ainsi de suite... selon la règle : `$semaine[index]`

Vous pouvez le vérifier en tapant la commande suivante :

```
<?php

echo $semaine[2];

?>
```

Bien sûr ici, nous avons construit un tableau contenant des variables string (les jours de la semaine) et des indices numériques (0,1,2 etc...).

On peut tout-à-fait ranger des valeurs numériques dans un tableau. Ainsi ce tableau qui stocke

quelques années marquantes de l'histoire de France...

Histoire de France

```php
<?php

$dates=array(1789,1830,1848,1851,1871,1914,1918,1936,1939,1945,1958,1968);

echo $dates[3];

?>
```

La commande **echo** renverra... 1851 ici.

7.2 Les tableaux associatifs

Ici, la seule différence, par rapport à un tableau simple, c'est que l'index n'est plus numérique, mais lui-même une variable string.

Par exemple, votre tableau veut stocker une adresse. Sachant que chaque adresse comporte en gros et dans cet ordre, un nom, un prénom, un numéro, une rue, un code postal et une ville...

On peut construire le tableau suivant :

```php
<?php

//On signale que notre variable $adresse4 sera un tableau

$adresse4 = array();

//on le remplit

$adresse4['nom'] = 'DUPONT';

$adresse4['prenom'] = 'Mickaël';

$adresse4['num'] = 12;

$adresse4['rue'] = 'rue des églantines';

$adresse4['cp'] = 93000;

$adresse4['ville'] = 'SAINT-DENIS';

?>
```

'num' est ici un index du tableau adresse. 12 est la valeur stockée à l'index 'num'.

```php
<?php

// Pour invoquer 12

echo $adresse4['num'];

?>
```

Bien entendu, l'intérêt d'un tableau d'adresses, c'est d'en stocker plus d'une ! Alors nous utiliserons un tableau de tableaux, un tableau imbriqué en fait...

Nous venons de voir que $adresse4 est un tableau.

Construction d'un tableau de tableaux

```php
<?php
//construction de mon tableau $agenda
$agenda=array($adresse0, $adresse1, $adresse2, $adresse3 ,$adresse4);
?>
```

Nous procédons ensuite en deux étapes :

Invoquer une donnée dans un tableau de tableaux - Méthode 1

```php
<?php
//récupérer l'adresse totale de Mickaël
$adresseMick= $agenda[4];
/*
En effet, l'adresse de Mickaël se trouve
dans notre agenda à l'index 4
(l'index numérique construit automatiquement)...
*/
//Récupérer enfin le nom de famille de Mickaël
echo $adresseMick['nom'];
?>
```

Il est également possible d'utiliser la syntaxe suivante, plus condensée :

Invoquer une donnée dans un tableau de tableaux - Méthode 2

```php
<?php
echo $agenda[4]['nom'];
?>
```

7.3 La boucle foreach

Bien sûr, l'idéal pour parcourir les valeurs d'un tableau, c'est une boucle. La boucle foreach (pour chaque élément) présente l'avantage de parcourir la totalité d'un tableau, même si l'on n'a aucune idée du nombre d'éléments qu'il contient.

Reprenons notre tableau des jours de la semaine, comment le parcourir?

```php
<?php
// construction du tableau semaine
$semaine=array('dimanche', 'lundi', 'mardi',
```

```
 'mercredi', 'jeudi', 'vendredi', 'samedi');
// parcours du tableau
foreach($semaine as $jour){
      echo'- '.$jour.'<br/>';
}
/*
Pour chaque valeur du tableau $semaine,
compose la variable $jour
et affiche le jour puis va à la ligne...
*/
?>
```

Recopiez maintenant le gros morceau de code qui suit, dont nous avons vu l'essentiel tout à l'heure, la construction à la main d'un agenda d'adresses :

L'agenda : un tableau de tableaux

```
<html>
      <head>
            <title>Mon agenda</title>
      </head>
      <body>
            <?php
                  $adresse0 = array();

                  //on le remplit
                  $adresse0 ['nom']='ZERO';
                  $adresse0 ['prenom']='Toto';
                  $adresse0 ['num'] = 10;
                  $adresse0 ['rue'] = 'rue des rosiers';
                  $adresse0 ['cp'] = 94000;
                  $adresse0 ['ville'] = 'IVRY-SUR-SEINE';

                  $adresse1 = array();
```

```php
//on le remplit
$adresse1 ['nom']='AIN';
$adresse1 ['prenom']='Anne';
$adresse1 ['num'] = 11;
$adresse1 ['rue'] = 'rue des moineaux';
$adresse1 ['cp'] = 57000;
$adresse1 ['ville'] = 'METZ';

$adresse2 = array();

//on le remplit
$adresse2 ['nom']='DEUX';
$adresse2 ['prenom']='Al';
$adresse2 ['num'] = 2;
$adresse2 ['rue'] = 'rue des arbres';
$adresse2 ['cp'] = 88000;
$adresse2 ['ville'] = 'EPINAL';

$adresse3 = array();

//on le remplit
$adresse3 ['nom']='TROIS';
$adresse3 ['prenom']='Léa';
$adresse3 ['num'] = 3;
$adresse3 ['rue'] = 'rue des éléphants';
$adresse3 ['cp'] = 69000;
$adresse3 ['ville'] = 'LYON';

$adresse4 = array();
```

```
                    //on le remplit
                    $adresse4 ['nom']='DUPONT';
                    $adresse4 ['prenom']='Mick';
                    $adresse4 ['num'] = 4;
                    $adresse4 ['rue'] = 'rue des églantines';
                    $adresse4 ['cp'] = 93000;
                    $adresse4 ['ville'] = 'SAINT-DENIS';

                    //on déclare et remplit l'agenda avec les adresses
précédentes
                    $agenda=array($adresse0,$adresse1,$adresse2,
                                                        $adresse3,$adresse4);
            ?>
      </body>
</html>
```

Vous l'aurez deviné, qui dit tableau de tableaux dit... boucles imbriquées pour la lecture du tableau : Une boucle pour lire chaque champ dans l'adresse imbriquée dans une boucle pour lire chaque adresse.

```
<?php
//pour chaque élément de $agenda crée la variable $adresse
foreach($agenda as $adresse){
      //pour chaque élément de $adresse crée la variable $element
      foreach($adresse as $element){
            //écris le $element sur la même ligne avec un tiret et des
espaces
            echo '- '.$element.' ';
      }
      //A chaque nouveau $adresse, saute une ligne
      echo'<br />';
}
?>
```

Et voici notre agenda qui s'affiche de façon assez lisible... Bien, nous arrivons maintenant au cœur de notre problème, et nous revenons à notre formulaire :

Comment récupérer des variables `$_POST`, résultats d'un formulaire de saisie, quand elles sont nombreuses ?

7.4 Boucle foreach et variables POST

PHP construit automatiquement un tableau associatif dès que l'on soumet un formulaire. Ce tableau se nomme `$_POST`, chaque élément a pour index le '`name`' d'un élément du formulaire, et chaque valeur, la valeur entrée par l'utilisateur dans chaque champ avant de cliquer sur le bouton '**`submit`**'.

On peut donc égrener tranquillement notre tableau de variables `$_POST` de cette façon :

```
<?php

/*

pour chaque élement du tableau $_POST,

récupère et affecte la valeur de l'index,

puis récupère et affecte la valeur associée à cet index

*/

foreach($_POST as $index=>$valeur){

      echo '- '.$valeur.'<br/>';

}

?>
```

Pour vous en convaincre, voyons ce que cette boucle produit :

```
<html>

      <head>

            <title>Votre IMC</title>

      </head>

      <body>

            <h1>

                  Déterminez votre IMC

                  et sachez quelle est votre corpulence

                  d'un point de vue médical

            </h1>

            <h2>

                  Entrez les données suivantes
```

```
            </h2>
            <form name="formulaire" method="post" action="tableau.php">
                  Entrez votre prénom :
                  <input type="text" name="prenom" /> <br />
                        Entrez votre taille (sous la forme 1.70) :
                  <input type="text" name="taille" /> <br />
                        Entrez votre poids (en kilos) :
                  <input type="text" name="poids" /> <br />
                  <input type="submit" name="valider" value="OK" />
            </form>
            <?php
                  echo 'Vos données entrées : '.'<br />';
                  if(isset($_POST['valider'])){
                        foreach($_POST as $index=>$valeur){
                              echo '- '.$index.' : '.$valeur.'<br />';
                        }
                  }
            ?>
      </body>
</html>
```

Ce petit bout de code ne fait qu'afficher la valeur des variables que le client vient d'entrer plus la valeur (permanente) de valider qui est 'ok'...

Enfin, un autre exemple, permettant, lui de ne pas afficher cette valeur de "valider" puisqu'ici, nous l'évitons par une condition :

```
<html>
      <head>
            <title>Identité</title>
      </head>
      <body>
            <h1>Identité</h1>
            <form name="formulaire" method="post" action="tp4.php">
```

```
                Entrez votre nom: <input type="text" name="nom" /><br />
                Entrez votre prénom: <input type="text" name="prenom"
/><br />
                Entrez votre âge: <input type="text" name="age" /><br />
                Entrez votre ville: <input type="text" name="ville" /><br
/>
                Entrez votre activité: <input type="text" name="activite"
/>               <br />
                <input type="submit" name="valider" value="OK" />
            </form>
            <?php
                if(isset($_POST['valider'])){
                    echo '<h3>Vous venez d\'entrer : </h3>';
                    foreach($_POST as $index=>$valeur){
                        if ($index!='valider'){
                            echo '- '.$valeur.'<br />';
                        }
                    }
                }
            ?>
        </body>
</html>
```

8 Les opérateurs, les fonctions, for et switch

8.1 Les opérateurs

```php
<?php
/*
LES OPERATEURS ARITHMETIQUES
*/
//la variable reçoit 5 (simple égal : opérateur d'affectation)
$valeur = 5;

//plus et moins
$nombre = (4 + 6) - 2;
//$nombre vaudra 8;

//multiplier ou diviser
$nombre = (4 * 6) / 2;
//$nombre vaudra 12;

//Priorités si vous ne mettez pas de parenthèses :
$valeur = 4 + 6 * 2;
//multiplier ou diviser est prioritaire.

//Si vous risquez d'oublier, sécurisez avec des parenthèses...
$valeur = 4 + (6 * 2);
//Sans parenthèses, cela revient à 16;

//Si vous souhaitez une autre priorité, déplacez les parenthèses :
$valeur = (4 + 6) * 2;
//Cela revient à 20
```

```php
//modulo ou ce qui reste après division entière
$nombre = 6 % 2;
//$nombre renverra 0 (car 6 est divisible par 2 donc reste zéro).

$nombre = 27 % 4,
//$nombre renverra 3 (car 6x4=24 reste 3).

//comparer
//RAPPEL IMPORTANT : penser au double égal
if ($nombre == 4)

//son contraire
if($nombre != 4)
//Différent de

//inférieur, supérieur
if($nombre < 4)
if($nombre > 4)

//inférieur ou égal, supérieur ou égal
if($nombre <= 4)
if($nombre >= 4)

/*
LES COMPARAISONS LOGIQUES
*/
//si l'âge est compris entre 15 et 25 ans :
if (($age >= 15) && ($age <= 25))
//Notez && pour le 'et' logique (AND marche aussi)
//Notez les parenthèses imbriquées pour chaque sous-condition
```

```
//si $truc est 'papa' ou 'maman'

//RAPPEL IMPORTANT : penser au double égal de comparaison

if(($truc == 'papa')||($truc == 'maman')){

      echo 'Chouette, mes parents !';

}

//Notez || pour le 'ou' logique

//on l'appelle aussi le ou inclusif (OR marche aussi)

//Notez les parenthèses imbriquées pour chaque sous-condition

?>
```

8.2 Les fonctions

8.2.1 Les fonctions dans la page

Imaginons le problème suivant. Votre programme affiche les notes /20 d'un élève. Vous souhaitez mettre en rouge toutes les notes (strictement) inférieures à 10. Vous souhaitez mettre en vert toutes les notes supérieures (ou égales) à 15.

Cela veut dire qu'à chaque apparition d'une note, il faut la tester et lui appliquer la couleur adéquate. Comme c'est une tâche répétitive que nous utiliserons à plusieurs endroits d'un programme complexe, nous la placerons dans une fonction.

Voici comment opérer :

La fonction que l'on va construire est placée en début de code, avant de l'utiliser. Par exemple, dans le début du body (au sein de balises PHP bien sûr).

Bien entendu, il faut aussi retenir une chose : c'est que si le nom de votre fonction est libre, une fois que vous l'avez choisi, vous ne pourrez invoquer celle-ci qu'en ne changeant rien (ni une lettre, ni une majuscule ou minuscule).

Le cadre de notre fonction se présentera ainsi :

```
function colore($nombre){

      toutes les instructions se situeront à l'intérieur de ces accolades;

      et se termineront par un ;

}
```

Dans la création de la fonction, il est obligatoire de commencer par écrire **function**, ensuite vient le nom (libre).

Enfin ici, notre fonction comporte un paramètre (une variable imaginaire, $nombre, dont on ne connaît pas encore la valeur)...

On fait comme si `$nombre` existait déjà, et on fait le petit bout de code qui effectue ce qu'on veut obtenir :

Ma fonction qui colore les notes

```php
<?php
function colore($nombre){
	if($nombre<10){
		echo '<font color="red">'.$nombre.'</font>';
	}
	else if($nombre>=15){
		echo '<font color="green">'.$nombre.'</font>';
	}
	//cas par défaut(noir)
	else{
		echo $nombre;
	}
}
?>
```

Comme exercice, remplacez ces notions HTML (font color=...) par des balises HTML couplées à une ”mise en page” CSS.

Vous voulez tester jusqu'au bout cette nouvelle fonction ?

Invoquez notre fonction dans une boucle

```php
<?php
//Construisons notre tableau de notes :
$notes=array(2,5,7,10,11,13,15,17,18);
/*Scannons-le grâce à une boucle foreach du type
echo 'Vos notes du trimestre :<br />';
foreach($notes as $note){
	echo '- '.$note.'<br />';
}*/
// Et maintenant, précisons notre boucle foreach
// pour y invoquer notre fonction.
echo 'Vos notes du trimestre :<br />';
foreach($notes as $note){
	echo '- '.colore($note).'<br />';
```

```
}
?>
```

En définitive, notre fichier notes.php complet donnera ceci :

```
<html>
	<head>
		<title>Notes du trimestre</title>
	</head>
	<body>
		<?php
			//Cette fonction colore en rouge les notes<10
			//et en vert les notes >=15
			function colore($nombre){
				if($nombre<10){
					echo '<font color="red">'.$nombre.'</font>';
				}
				else if($nombre>=15){
					echo '<font color="green">'.$nombre.'</font>';
				}
				//cas par défaut(affiche sans modifier couleur)
				else{
					echo $nombre;
				}
			}
			//Construisons notre tableau de notes :
			$notes=array(2,5,7,10,11,13,15,17,18);
			//La boucle foreach scanne le tableau
			//en appliquant la fonction colore
			echo 'Vos notes du trimestre :<br />';
			foreach($notes as $note){
				echo '- ';
				colore($note);
```

```
                    echo '<br />';
                }
            ?>
        </body>
</html>
```

8.2.2 Plusieurs paramètres passés à la fonction

Les fonctions peuvent être définies avec plusieurs paramètres. Par exemple, nous pouvons définir une fonction moyenne annuelle telle qu'elle soit la moyenne des trois trimestres scolaires :

```
<?php
function MoyenneAnnuelle($trim1, $trim2, $trim3){
    $MA=($trim1+$trim2+$trim3)/3;
    echo 'Votre moyenne annuelle : '.$MA;
}
?>
```

Dans ce cas, pour l'invoquer, on pourra écrire :

```
<?php
MoyenneAnnuelle(8,10,12);
// Ceci aura pour résultat :
// Votre moyenne annuelle : 10
?>
```

8.2.3 Les fonctions dans un fichier séparé

Si vous développez un programme un peu ambitieux, vous aurez un certain nombre de fonctions. Vous pouvez tout-à-fait les mettre à part dans un fichier voisin que vous appelleriez `fonctions.php` par exemple.

Il vous suffira ensuite d'inclure une seule fois au tout début de chacune de vos pages php, au sein de balises PHP bien entendu par la formule suivante :

```
<?php
include('fonctions.php');
?>
```

Ceci vous permet ensuite d'invoquer n'importe quelle fonction définie dans votre page `fonctions.php`.

Par exemple, vous avez développé tout un site et vous souhaitez sur chacune des pages signaler le dernier événement mis à jour.

Vous pourriez procéder ainsi :

Dans fonctions.php, vous rédigez cette fonction :

une fonction de simple affichage

```
<?php

function DerniereMaj(){

      echo 'Dernière parution mise à jour : le 10/01/2009';

}

?>
```

Dans chacune des pages de votre site maintenant, vous vous contentez de mettre, à l'endroit où vous souhaitez que votre avis de mise à jour (maj) apparaisse, la ligne suivante :

```
<?php

DerniereMaj();

?>
```

Bien entendu aussi, vous aurez songé auparavant à mettre votre "**include**" de la page `fonctions.php` dans chaque page de votre site... Ceci vous permettra, à chaque nouvel avis de maj, de changer centralement et une seule fois dans `fonctions.php` votre texte. Et cela aura pour conséquence de le changer sur chacune de vos pages.

Vous remarquerez que la fonction `DerniereMaj` n'a pas de paramètres. C'est une simple fonction d'affichage. Une fonction sans paramètres conserve toutefois les parenthèses vides, dans sa définition comme à son invocation.

8.2.4 Fonction qui renvoie une valeur de retour

Une dernière chose sur les fonctions : Si l'on ne veut pas une fonction qui affiche, mais une fonction qui par exemple calcule et renvoie une valeur, on utilisera le mot clé **return**.

Par exemple, réalisons une fonction qui nous retourne le verdict du nombre que l'on vient d'entrer : pair ou impair ? Un formulaire de saisie, et une analyse du nombre entré

```
<html>

      <head>

            <title>Pair ou impair ?</title>

      </head>

      <body>

            <?php

                  //fonction qui fait le diagnostic
```

```
                function parite($nombre){
                        //si le reste de la division est zéro, c'est pair
                        if (($nombre%2)==0){
                                //on initialise les deux valeurs de verdict
                                $verdict='pair';
                        }
                        else{
                                $verdict='impair';
                        }
                        //on renvoie le verdict, tout à la fin
                        return $verdict;
                }
        ?>
        <form method="POST" action="fonction.php">
                Entrez votre nombre <input type="text" name="num" />
                <input type="submit" name="valider" value="OK" />
        </form>
        <?php
                //si user a cliqué OK
                if(isset($_POST['valider'])){
                        //récupère la valeur entrée
                        $nombre=$_POST['num'];
                        //place dans $toto la valeur de retour de ma
fonction
                        $toto=parite($nombre);
                        //affiche le verdict entier
                        echo 'Ce nombre est '.$toto.'.';
                }
        ?>
    </body>
</html>
```

L'instruction **return** stoppe la fonction. Il faut donc le placer en tout dernier, juste avant l'accolade finale qui ferme la fonction. Pour la même raison, **return** ne peut renvoyer qu'une seule valeur, libre à vous cependant de rendre cette valeur complexe comme un tableau. Au moment de l'invocation d'une fonction qui retourne une valeur, pensez à "ranger" la valeur retournée dans une variable "réception".

8.3 La boucle for

Une boucle for, c'est une boucle qui dit "pour chaque valeur de ... à ..., recommence".

Traditionnellement, $i est le nom de la variable qui sert un peu de compteur pour une boucle for, mais rien n'interdit de lui donner un nom plus explicite.

La syntaxe de cette boucle est la suivante :

```
<?php

//Copiez-moi 50 fois la punition !!!

//pour i égal zéro, i inférieur à 50, i plus plus

for($i=0;$i<50;$i++){

      echo 'Je ne tricherai plus à un devoir.<br />';

}

?>
```

Remarques

Puisque `$i` commence à 0 et non à 1, pensez bien à mettre ensuite strictement inférieur à la valeur de la limite supérieure de la boucle.

Ici on peut traduire par : pour `$i` de 0 à 49, ce qui fait bien... 50 tours de boucle.

8.4 L'instruction switch

Voici une commande qui est une condition, en tout cas, qui gère très simplement, le principe des

conditions multiples un peu lourd à gérer avec des if, elseif, else, if imbriqués et ordre des if.

Switch, c'est un interrupteur pour des conditions qui s'excluent l'une l'autre

```
<?php

switch($corpulence){

      case 'denutrition':

            $verdict='Vous êtes en dénutrition.';

            // placer ici un lien vers un centre d'aide

            // aux troubles du comportement alimentaire

            break;
```

```
        case 'maigre':
                $verdict='Vous êtes maigre.';
                //placer ici quelques conseils d'hygiène de vie
                //MANGEZ bougez
                break;
        case 'normal':
                $verdict='Vous avez une corpulence normale.';
                //placer ici quelques conseils "Continuez commme ça"
                break;
        case 'surpoids':
                $verdict='Vous êtes en surpoids.';
                //placer ici quelques conseils d'hygiène de vie
                //BOUGEZ mangez
                break;
        case 'obese_modere':
                $verdict='Vous êtes en état d\'obésité modérée.';
                //placer ici quelques conseils d'hygiène de vie
                //BOUGEZ BOUGEZ mangez
                break;
        case 'obese_severe':
                $verdict='Vous êtes en état d\'obésité sévère.';
                //placer ici quelques conseils d'hygiène de vie
                //BOUGEZ BOUGEZ BOUGEZ REVOYEZ TOUT VOTRE MODE DE VIE
                break;
        case 'obese-massive':
                $verdict='Vous êtes en état d\'obésité massive.';
                //placer ici un lien vers un centre d'aide aux
                //troubles du comportement alimentaire
}
?>
```

Dans le cas (**case**) où la variable `$corpulence` est égale à 'denutrition' : affecte telle 'formule' à la variable `$verdict` et procède à telle série d'instructions, dans le case 'maigre' affecte telle autre valeur à `$verdict` etc. etc.

On sait que ces conditions s'excluent l'une l'autre grâce à l'instruction **break**; qui signifie : 'quand tu as finis, sors de ce **switch**'.

D'où la raison pour laquelle le dernier case 'obese_massive' ne nécessite pas de **break**. Bien sûr, avant ce switch, il faudrait s'assurer que `$corpulence`, via des conditions, se voit bien affecter tout cet éventail de valeurs.

```
<?php

echo '<h3>'.$verdict.'</h3><br />';

echo '<p>Mais ne prenez pas ce constat, aussi brut soit-il, trop à coeur.

<br />Une corpulence, s\'il en est besoin,

peut changer grâce aux conseils ci-dessus.

Ce n\'est qu\'une norme qui doit vous indiquer

si votre hygiène de vie est à faire évoluer ou non.

<br /><br />

Pas le verdict du tribunal d\'inquisition !.</p>';

?>
```

L'instruction **switch** rendra votre code beaucoup plus lisible qu'une série de **if** imbriqués, lorsque le remplacement d'une série de **if** est possible par cette instruction.

9 Gestion des sessions

Le support des sessions de PHP est un moyen de préserver des données entre plusieurs accès. Cela vous permet de créer des applications personnalisées, et d'augmenter l'attrait de votre site.

Chaque visiteur accédant à votre page web se voit assigner un identifiant unique, appelé "identifiant de session". Il peut être stocké soit dans un cookie, soit propagé dans l'URL.

Le support des sessions vous permet d'enregistrer un nombre illimité de variables qui doivent être préservées entre les requêtes. Lorsqu'un visiteur accède à votre site, PHP va vérifier automatiquement (si **session.auto_start** est activé) ou sur demande (explicitement avec session_start() ou implicitement avec session_register()) s'il existe une session du même nom. Si c'est le cas, l'environnement précédemment sauvé sera recréé.

9.1 Passer l'identifiant de session (session ID)

Il y a deux méthodes de propagation de l'identifiant de session :

- Cookies
- Par URL

Le module de session supporte les deux méthodes. Les cookies sont optimaux, mais comme ils ne sont pas sûrs (tous les internautes ne les acceptent pas), ils ne sont pas fiables. La seconde méthode place l'identifiant de session directement dans les URL.

PHP est capable de faire cela de manière transparente, lorsqu'il est compilé avec l'option --enable-trans-sid. Si vous activez cette option et que l'option d'exécution *session.use_trans_sid* est activée, les URLs relatives seront modifiées pour contenir l'identifiant de session automatiquement.

Alternativement, vous pouvez utiliser la constante **SID** qui est définie si la session a commencé. Si le client n'envoie pas un cookie de session approprié, il aura la forme *session_name=session_id*. Sinon, il vaudra une chaîne vide. Ainsi, vous pouvez dans tous les cas l'inclure dans l'URL.

L'exemple suivant vous montre comment enregistrer une variable et comment réaliser un lien correct avec une autre page, avec **SID**.

Exemple #1 Compter le nombre de passages d'un utilisateur sur une page

```
<?php

session_start();

if (empty($_SESSION['count'])) {
 $_SESSION['count'] = 1;
} else {
 $_SESSION['count']++;
}
?>

<p>
 Bonjour visiteur, vous avez vu cette page <?php echo $_SESSION['count']; ?
> fois.
</p>

<p>
 Pour continuer, <a href="nextpage.php?<?php echo htmlspecialchars(SID); ?>
">cliquez ici</a>.
</p>
```

La fonction htmlspecialchars() est utilisée lors de l'affichage du **SID** dans le but de contrer les attaques XSS.

L'affichage du **SID**, comme montré dans l'exemple ci-dessus, n'est pas nécessaire si --enable-trans-sid a été utilisé pour compiler PHP.

9.2 Cookies

PHP supporte les cookies HTTP de manière transparente. Les cookies sont un mécanisme d'enregistrement d'informations sur le client, et de lecture de ces informations. Ce système permet d'identifier et de suivre les visiteurs. Vous pouvez envoyer un cookie avec la fonction setcookie() ou setrawcookie(). Les cookies font partie des en-têtes HTTP, ce qui impose que setcookie() soit appelée avant tout affichage de texte.

Tous les cookies qui sont envoyés au client seront automatiquement inclus dans un tableau auto-global *$_COOKIE* si variables_order contient "C". Si vous souhaitez affecter plusieurs valeurs à un seul cookie, ajoutez *[]* au nom du cookie.

Selon la configuration de register_globals, des variables PHP peuvent être créées à partir des cookies. Cependant, il n'est pas recommandé de se fier à elles, puisque cette fonctionnalité est souvent désactivée pour des raisons de sécurité.

bool **setcookie** (string `$name` [, string `$value` [, int `$expire` = 0 [, string `$path` [, string `$domain` [, bool `$secure` = false [, bool `$httponly` = false]]]]]])

Note: expire:

> Le temps après lequel le cookie expire. C'est un timestamp Unix, donc, ce sera un nombre de secondes depuis l'époque Unix (1 Janvier 1970). En d'autres termes, vous devriez fixer cette valeur à l'aide de la fonction time() en y ajoutant le nombre de secondes après lequel on veut que le cookie expire.

Si vous ne spécifiez pas ce paramètre ou s'il vaut 0, le cookie expirera à la fin de la session (lorsque le navigateur sera fermé).

10 Les bases de données

10.1 Introduction

Pour rappel, le code HTML ne sert qu'à l'affichage. Le code PHP sert à créer un HTML dynamique jusqu'à un certain point, en tout cas, PHP peut organiser à chaque chargement de votre page web le HTML de façon différente.

Alors si vous voulez conserver sur la durée les données entrées par les visiteurs de votre site via votre formulaire, de façon automatique et sans plus vous occuper de rien, il va falloir enregistrer ces données dans une vase de données.

phpMyAdmin vous permet de gérer tout ce qui concerne les bases de données de type MySQL.

Nous allons commencer par une base qui ne comporte qu'une seule table, pour faire simple.

Cliquez le bouton "Créer une base", nommez-la MaBase, puis cliquez "Créer".

Créez une nouvelle table "Utilisateurs" avec 5 champs...

Ils se découperont ainsi (ce qui nous intéresse surtout, c'est le type des champs)...

Nos 5 champs et leur type respectif

- ID, qui est un INT (integer : entier), cochez AI (auto-incrémenté pour que phpMyAdmin gère ce champ tout seul)
- pseudo qui est un VARCHAR (caractères dont le nombre varie), entrez 15 à taille/longueur (nombre maximum de caractères autorisés)
- sexe qui est un CHAR (caractères dont le nombre est fixe), entrez 1 à taille/longueur (nous mettrons simplement G ou F)
- age qui est un INT, entrez 3 à taille (le nombre de chiffres). Pour un âge, c'est sympa de penser aux centenaires...
- et enfin dateInscription (sélectionnez DATE)

Maintenant, nous allons la remplir "à la main".

A gauche dans votre arborescence, sélectionnez votre table 'Utilisateurs'. Vous voyez la structure de votre table qui s'affiche... sous forme de tableau.

Cliquez 'Insérer' dans le menu du haut... Par défaut, phpMyAdmin vous donne un formulaire d'insertion deux entrées par deux entrées...

Laissez toujours le champ ID vide. On a en effet indiqué qu'il se remplirait (et s'incrémenterait) automatiquement.

Et remplissez ainsi vos 5 insertions :

Si les dates vous paraissent bizarres, c'est qu'elles sont à l'anglaise (année/mois/jour)...

Nos 5 premiers enregistrements

- Lili F 18 2009-01-02
- Toto G 13 2008-01-02
- Loulou G 25 2008-12-30
- Zézette F 20 2009-01-02
- Mimi F 15 2009-01-10

On dira que dans votre table Utilisateurs (qui compte donc 5 champs), il y a 5 enregistrements, 5 personnes enregistrées autrement dit, avec toutes leurs données. La seule façon certaine de les distinguer, c'est leur ID (identification data).

10.2 Alimenter sa base de données via php

De même que PHP vous permet de créer des pages plus dynamiques et personnalisées, PHP vous permet maintenant de créer vos requêtes SQL de façon à automatiser l'alimentation de vos bases.

Voici un exemple très concret :

Vous allez créer un formulaire qui permettra de continuer à remplir automatiquement votre base de données MaBase, plus précisément sur votre table Utilisateurs, table que pour l'instant vous avez commencé à remplir à la main, via l'interface PHPMyAdmin.

10.2.1 Présentation du formulaire

Dans un fichier form.php, entrez ce code :

```
<html>

	<head>

		<title>Formulaire de saisie utilisateur </title>

	</head>

	<body>

		<h1>Inscrivez-vous !</h1>

		<h2>Entrez les données demandées :</h2>

		<form name="inscription" method="post" action="form.php">

			Entrez votre pseudo : <input type="text" name="pseudo" />
<br />

			Garçon ou fille ?

			<input type="radio" name="sexe" value="G" />Garçon

			<input type="radio" name="sexe" value="F" />Fille<br />

			Entrez votre age : <input type="text" name="age" /><br />

			<input type="submit" name="valider" value="OK" />

		</form>

	</body>

</html>
```

Vous remarquerez que nous ne demandons que 3 éléments d'identification à l'utilisateur, alors que la base en comporte 5 par enregistrement : C'est normal, le premier champ de notre base est rempli automatiquement par mysql et le dernier, la date d'inscription, nous allons le remplir aussi

automatiquement, via PHP cette fois, en entrant la date du jour dès que l'utilisateur clique 'OK'.

Vous remarquerez que pour des données aussi peu variées que le sexe de la personne, nous avons préféré contrôler la saisie en proposant à l'utilisateur deux boutons radio. Pour limiter les erreurs dans la base SQL, bien songer à proposer des listes déroulantes ou des cases à cocher quand le champ s'y prête.

10.2.2 Connexion à la base de données en php

A chaque fois que nous voulons que PHP se connecte à notre base, on lui donnera une série d'instructions... Donc on va commencer par coller les paramètres de connexion dans une fonction pour ne pas trop rabâcher dans notre code... On crée donc un fichier fonctions.php, dans lequel (entre autres) on insère la fonction suivante

```
<?php

function connectMaBase(){

      $base = mysql_connect('localhost', 'root', '');

      mysql_select_db('MaBase', $base) ;

}

?>
```

Vous n'avez rien à modifier... Sachez simplement que " est votre mot de passe, laissé vierge ici comme il l'est par défaut dans votre configuration de départ.

Cette fonction crée une variable `$base` qui prépare la commande de connexion à votre base avec le nom du serveur (ici `localhost`), le nom de l'utilisateur (ici `root` qui signifie administrateur de votre propre base locale et enfin le mot de passe. La ligne suivante lance la commande de sélection de votre base (où l'on entre donc le nom de votre base), puis on reprend la variable qui contient toute la commande de connexion... Bien entendu, pour invoquer la fonction que vous venez de créer sur fonctions.php, il vous faudra sur form.php faire un **`include`** de ce fichier fonctions.php, puis lancer la fonction au moment opportun par la ligne suivante :

```
<?php

connectMaBase();

?>
```

10.2.3 Pour travailler proprement

Quand on rédige une commande d'insertion SQL via PHP, on préfère procéder ainsi, qui paraît un peu compliqué au début, mais simplifie toute la compréhension ensuite.

Insérer une donnée SQL via PHP, l'ossature

1. On se **connecte** à la base (en utilisant notre fonction de connexion toute prête).
2. On **prépare la commande sql** en la stockant dans une variable PHP du type `$sql` (pour langage sql);
3. On la **lance**.
4. On **ferme** la connexion.

10.2.4 Insérer des données dans notre base de données via php

le code suivant ira sur le fichier form.php dans notre condition => si l'utilisateur a cliqué OK Insérer une donnée sql via PHP, le code

```
<?php

//On récupère les valeurs entrées par l'utilisateur :

$pseudo=$_POST['pseudo'];

$age=$_POST['age'];

$sexe=$_POST['sexe'];


//On crée une variable date du jour grâce à la fonction date() de PHP

$today = date("d.m.y");


//On se connecte

connectMaBase();


//On prépare la commande sql d'insertion

$sql = 'INSERT INTO Utilisateurs VALUES("","'.$pseudo.'","'.$sexe.'","'

.$age.'","'.$today.'")';


/*on lance la commande (mysql_query) et au cas où,

on rédige un petit message d'erreur si la requête ne passe pas (or die)

(Message qui intègrera les causes d'erreur sql)*/

mysql_query ($sql) or die ('Erreur SQL !'.$sql.'<br />'.mysql_error());
```

```
// on ferme la connexion

mysql_close();

?>
```

```
<?php

//On prépare la commande sql d'insertion

$sql = 'INSERT INTO Utilisateurs VALUES("","'.$pseudo.'","'

.$sexe.'","'.$age.'","'.$today.'")';

?>
```

La première valeur est laissée en blanc car c'est l'ID auto-incrémenté... Si vous l'oubliez, ce blanc, il vous renverra un message d'erreur sql disant que le nombre de données insérées ne coïncident pas avec le nombre de champs. Ensuite il faut gérer au sein des guillemets simples (apostrophes) tout ce qui est PHP et au sein des guillemets doubles tout ce qui est sql... On place les guillemets au sein des apostrophes pour les afficher dans le sql, on place les variables 'nues' dans PHP.

10.2.5 Synthèse

fonctions.php

```
<?php

function connectMaBase(){

      $base = mysql_connect ('localhost', 'root', '');

      mysql_select_db ('MaBase', $base) ;

}

?>
```

form.php

```
<?php

include("fonctions.php");

?>

<html>

      <head>
```

```
        <title>Formulaire de saisie utilisateur </title>
</head>
<body>
        <h1>Inscrivez-vous !</h1>
        <h2>Entrez les données demandées :</h2>
        <form name="inscription" method="post" action="form.php">
                Entrez votre pseudo : <input type="text" name="pseudo" />
                <br />
                Garçon ou fille ?
                <input type="radio" name="sexe" value="G" />Garçon
                <input type="radio" name="sexe" value="F" />Fille<br />
                Entrez votre age : <input type="text" name="age" /><br />
                <input type="submit" name="valider" value="OK" />
        </form>
        <?php
        if (isset ($_POST['valider'])){

                //On récupère les valeurs entrées par l'utilisateur :
                $pseudo=$_POST['pseudo'];
                $age=$_POST['age'];
                $sexe=$_POST['sexe'];

                //On construit la date d'aujourd'hui
                //strictement comme sql la construit
                $today = date("y-m-d");

                //On se connecte
                connectMaBase();

                //On prépare la commande sql d'insertion
                $sql = 'INSERT INTO Utilisateurs VALUES("","'
```

```
                .$pseudo.'","'.$sexe.'","'.$age.'","'.$today.'")';

                /*
                on lance la commande (mysql_query) et au cas où,
                on rédige un petit message d'erreur
                si la requête ne passe pas (or die)
                (Message qui intègrera les causes d'erreur sql)
                */
                mysql_query ($sql)
                or
                die ('Erreur SQL !'.$sql.'<br />'.mysql_error());

                // on ferme la connexion
                mysql_close();
            }
            ?>
        </body>
</html>
```

10.3 Récupérer les données de sa base de données via php

10.3.1 Pour travailler proprement

Voici cette fois l'ordre logique pour un code PHP propre qui lance une requête sql

1. On se **connecte** à la base (en utilisant notre fonction de connexion toute prête).
2. On **prépare la commande sql** en la stockant dans une variable PHP du type $sql (pour langage sql). On la **lance**, en récupérant le résultat dans une variable que nous appellerons $req (pour requete sql et qui pourra être un tableau si le résultat dépasse un élément).
3. Si c'est un tableau : **On scanne $req avec une boucle while** (car on ne sait pas toujours le nombre de champs, ni d'enregistrements) et grâce à la fonction **mysql_fetch_array($req)**, chaque élément de ce tableau se invoquera ainsi : **$data['champ']**.
4. Maintenant qu'on a tout récupéré dans des variables "solides" PHP, on **libère la mémoire sql mobilisée par cette requête**.
5. On **ferme** la connexion sql.

10.3.2 Le code pour 'récupérer toutes les filles'

```
<?php
```

```php
include("fonctions.php");
?>
<html>
	<head>
		<title>TOUTES LES INFOS SUR LES INSCRITS DU SITE</title>
	</head>
	<body>
	<?php
		//On se connecte
		connectMaBase();

		// On prépare la requête
		$sql = 'SELECT * FROM utilisateurs WHERE sexe="F"';

		// On lance la requête (mysql_query) et on impose un message
		// d'erreur si la requête ne se passe pas (or die)
		$req = mysql_query($sql) or die('Erreur SQL !<br />'.$sql
		.'<br />'.mysql_error());

		//on organise $req en tableau associatif $data['champ']
		//en scannant chaque enregistrement récupéré
		//on en profite pour gérer l'affichage
		//titre de la page avant la boucle
		echo'<h2>TOUTES LES FILLES INSCRITES :</h2>';

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

			// on affiche les résultats
			echo 'Pseudo : '.$data['Pseudo'].'<br />';
```

```
            echo 'Son âge : '.$data['Age'].'<br />';
            echo 'Sa date d\'inscription : '.$data['DateInscription']
            echo '<br /><br/>';
        }

        //On libère la mémoire mobilisée pour cette requête dans sql
        //$data de PHP lui est toujours accessible !
        mysql_free_result ($req);

        //On ferme sql
        mysql_close ();
    ?>
    </body>
</html>
```

11 PHP Data Objects

L'extension *PHP Data Objects* (PDO) définit une excellente interface pour accéder à une base de données depuis PHP. Chaque pilote de base de données implémenté dans l'interface PDO peut utiliser des fonctionnalités spécifiques de chacune des bases de données en utilisant des extensions de fonctions. Notez que vous ne pouvez exécuter aucune fonction de base de données en utilisant l'extension PDO par elle-même ; vous devez utiliser un driver PDO spécifique à la base de données pour accéder au serveur de base de données.

PDO fournit une interface d'abstraction à l'*accès de données*, ce qui signifie que vous utilisez les mêmes fonctions pour exécuter des requêtes ou récupérer les données quelque soit la base de données utilisée.

11.1 Connexions et gestionnaire de connexion

Les connexions sont établies en créant des instances de la classe de base de PDO. Peu importe quel driver vous voulez utiliser ; vous utilisez toujours le nom de la classe PDO. Le constructeur accepte des paramètres pour spécifier la source de la base de données (connue en tant que DSN) et optionnellement, le nom d'utilisateur et le mot de passe (s'il y en a un).

Exemple #1 Connexion à MySQL

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

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

Exemple #2 Gestion des erreurs de connexion

```
<?php
try {
    $dbh = new PDO('mysql:host=localhost;dbname=test', $user, $pass);
    foreach($dbh->query('SELECT * from FOO') as $row) {
        print_r($row);
    }
    $dbh = null;
} catch (PDOException $e) {
    print "Erreur !: " . $e->getMessage() . "<br/>";
    die();
}
?>
```

Lorsque la connexion à la base de données a réussi, une instance de la classe PDO est retournée à votre script. La connexion est active tant que l'objet PDO l'est. Pour clore la connexion, vous devez détruire l'objet en vous assurant que toutes ses références sont effacées. Vous pouvez faire cela en assignant **`NULL`** à la variable gérant l'objet. Si vous ne le faites pas explicitement, PHP fermera automatiquement la connexion lorsque le script arrivera à la fin.

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

Exemple #4 Connexions persistantes

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

Note:

Si vous souhaitez utiliser des connexions persistantes, vous devez utiliser la valeur **PDO::ATTR_PERSISTENT** dans le le tableau des options du driver passé au constructeur PDO. Si vous définissez cet attribut avec la méthode PDO::setAttribute() après l'instanciation de l'objet, le driver n'utilisera pas les connexions persistantes.

11.2 Transactions et validation automatique (autocommit)

Maintenant que vous êtes connecté via PDO, vous devez comprendre comment PDO gère les transactions avant d'exécuter des requêtes. Si vous n'avez jamais utilisé les transactions, elles offrent 4 fonctionnalités majeures : Atomicité, Consistance, Isolation et Durabilité (ACID). En d'autres termes, n'importe quel travail mené à bien dans une transaction, même s'il est effectué par étapes, est garanti d'être appliqué à la base de données sans risque, et sans interférence pour les autres connexions, quand il est validé. Le travail des transactions peut également être automatiquement annulé à votre demande (en supposant que vous n'avez encore rien validé), ce qui rend la gestion des erreurs bien plus simple dans vos scripts.

Les transactions sont typiquement implémentées pour appliquer toutes vos modifications en une seule fois ; ceci a le bel effet d'éprouver drastiquement l'efficacité de vos mises à jour. Dans d'autres termes, les transactions rendent vos scripts plus rapides et potentiellement plus robustes (vous devez les utiliser correctement pour avoir ces bénéfices).

Malheureusement, toutes les bases de données ne supportent pas les transactions, donc, PDO doit s'exécuter en mode "autocommit" lorsque vous ouvrez pour la première fois la connexion. Le mode "autocommit" signifie que toutes les requêtes que vous exécutez ont leurs transactions implicites, si la base de données le supporte ou aucune transaction si la base de données ne les supporte pas. Si vous avez besoin d'une transaction, vous devez utiliser la méthode PDO::beginTransaction() pour l'initialiser. Si le driver utilisé ne supporte pas les transactions, une exception PDO sera lancée (en accord avec votre gestionnaire d'erreurs : ceci est toujours une erreur sérieuse). Une fois que vous êtes dans une transaction, vous devez utiliser la fonction PDO::commit() ou la fonction PDO::rollBack() pour la terminer, suivant le succès de votre code durant la transaction.

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

Exemple #1 Exécution d'un groupe dans une transaction

Dans l'exemple suivant, supposons que nous allons créer un jeu d'entrées pour un nouvel employé, dont le numéro d'ID sera 23. En plus des données basiques sur cette personne, nous devons également lui enregistrer son salaire. Il est très simple d'effectuer deux mises à jour séparées, mais

en les enfermant dans les appels des fonctions PDO::beginTransaction() et PDO::commit(), nous garantissons que personne ne pourra voir ces modifications tant qu'elles ne seront pas complètes. Si quelque chose tourne mal, le bloc de capture annulera toutes les modifications effectuées depuis le début de la transaction et affichera un message d'erreur.

```
   <?php
try {
  $dbh = new PDO('odbc:SAMPLE', 'db2inst1', 'ibmdb2',
      array(PDO::ATTR_PERSISTENT => true));
  echo "Connecté\n";
} catch (Exception $e) {
  die("Impossible de se connecter: " . $e->getMessage());
}

try {
  $dbh->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

  $dbh->beginTransaction();
  $dbh-
>exec("insert into staff (id, first, last) values (23, 'Joe', 'Bloggs')");
  $dbh->exec("insert into salarychange (id, amount, changedate)
      values (23, 50000, NOW())");
  $dbh->commit();

} catch (Exception $e) {
  $dbh->rollBack();
  echo "Failed: " . $e->getMessage();
}
?>
```

11.3 Requêtes préparées et procédures stockées

La plupart des bases de données supportent le concept des requêtes préparées. Qu'est-ce donc ? Vous pouvez les voir comme une sorte de modèle compilé pour le SQL que vous voulez exécuter, qui peut être personnalisé en utilisant des variables en guise de paramètres. Les requêtes préparées offrent deux fonctionnalités essentielles :

- La requête ne doit être analysée (ou préparée) qu'une seule fois, mais peut être exécutée plusieurs fois avec des paramètres identiques ou différents. Lorsque la requête est préparée, la base de données va analyser, compiler et optimiser son plan pour exécuter la requête. Pour les requêtes complexes, ce processus peut prendre assez de temps, ce qui peut ralentir vos applications si vous devez répéter la même requête plusieurs fois avec différents paramètres. En utilisant les requêtes préparées, vous évitez ainsi de répéter le cycle analyser/compilation/optimisation. Pour résumer, les requêtes préparées utilisent moins de ressources et s'exécutent plus rapidement.
- Les paramètres pour préparer les requêtes n'ont pas besoin d'être entre guillemets ; le driver le gère pour vous. Si votre application utilise exclusivement les requêtes préparées, vous pouvez être sûr qu'aucune injection SQL n'est possible (Cependant, si vous construisez d'autres parties de la requête en vous basant sur des entrées utilisateurs, vous continuez à prendre un risque).

Les requêtes préparées sont tellement pratiques que c'est l'unique fonctionnalité que PDO émule pour les drivers qui ne les supportent pas. Ceci assure de pouvoir utiliser la même technique pour accéder aux données, sans se soucier des capacités de la base de données.

Exemple #1 Insertions répétitives en utilisant les requêtes préparées

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

```
<?php
$stmt = $dbh-
>prepare("INSERT INTO REGISTRY (name, value) VALUES (:name, :value)");
$stmt->bindParam(':name', $name);
$stmt->bindParam(':value', $value);

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();

// insertion d'une autre ligne avec des valeurs différentes
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

Exemple #2 Insertions répétées en utilisant des requêtes réparées

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

```
<?php
$stmt = $dbh->prepare("INSERT INTO REGISTRY (name, value) VALUES (?, ?)");
$stmt->bindParam(1, $name);
$stmt->bindParam(2, $value);

// insertion d'une ligne
$name = 'one';
$value = 1;
$stmt->execute();

// insertion d'une autre ligne avec différentes valeurs
$name = 'two';
$value = 2;
$stmt->execute();
?>
```

Exemple #3 Récupération des données en utilisant des requêtes préparées

Cet exemple récupère des données basées sur la valeur d'une clé fournie par un formulaire. L'entrée utilisateur est automatiquement échappée, il n'y a donc aucun risque d'attaque par injection SQL.

```
<?php
$stmt = $dbh->prepare("SELECT * FROM REGISTRY where name = ?");
if ($stmt->execute(array($_GET['name']))) {
  while ($row = $stmt->fetch()) {
    print_r($row);
  }
}
?>
```

Si le driver de la base de données le supporte, vous pouvez également lier des paramètres aussi bien pour l'entrée que pour la sortie. Les paramètres de sortie sont utilisés typiquement pour récupérer les valeurs d'une procédure stockée. Les paramètres de sortie sont un peu plus complexe à utiliser que les paramètres d'entrée car vous devez savoir la longueur d'un paramètre donné pourra atteindre lorsque vous le liez. Si la valeur retournée est plus longue que la taille qui vous auriez suggéré, une erreur sera émise.

Exemple #4 Appel d'une procédure stockée avec un paramètre de sortie

```
<?php
$stmt = $dbh->prepare("CALL sp_returns_string(?)");
$stmt->bindParam(1, $return_value, PDO::PARAM_STR, 4000);

// Appel de la procédure stockée
$stmt->execute();

print "La procédure a retourné : $return_value\n";
?>
```

12 Fonctions de gestion des variables

12.1 Sommaire

- debug_zval_dump — Extrait une représentation sous forme de chaîne d'une valeur interne à Zend pour affichage
- doubleval — Alias de floatval
- empty — Détermine si une variable est vide
- floatval — Convertit une chaîne en nombre à virgule flottante
- get_defined_vars — Liste toutes les variables définies
- get_resource_type — Retourne le type de ressource
- gettype — Retourne le type de la variable
- import_request_variables — Importe les variables de GET/POST/Cookie dans l'environnement global
- intval — Retourne la valeur numérique entière équivalente d'une variable
- is_array — Détermine si une variable est un tableau
- is_bool — Détermine si une variable est un booléen
- is_callable — Détermine si l'argument peut être appelé comme fonction
- is_double — Alias de is_float
- is_float — Détermine si une variable est de type nombre décimal
- is_int — Détermine si une variable est de type nombre entier
- is_integer — Alias de is_int
- is_long — Alias de is_int
- is_null — Indique si une variable vaut NULL
- is_numeric — Détermine si une variable est un type numérique
- is_object — Détermine si une variable est de type objet
- is_real — Alias de is_float
- is_resource — Détermine si une variable est une ressource
- is_scalar — Indique si une variable est un scalaire
- is_string — Détermine si une variable est de type chaîne de caractères
- isset — Détermine si une variable est définie et est différente de NULL
- print_r — Affiche des informations lisibles pour une variable
- serialize — Linéarise une variable
- settype — Affecte un type à une variable
- strval — Récupère la valeur d'une variable, au format chaîne
- unserialize — Crée une variable PHP à partir d'une valeur linéarisée
- unset — Détruit une variable
- var_dump — Affiche les informations d'une variable
- var_export — Retourne le code PHP utilisé pour générer une variable

13 Fonctions sur les tableaux

Voir aussi is_array(), explode(), implode(), split(), preg_split() et unset().

13.1 Sommaire

- array_change_key_case — Change la casse des clés d'un tableau
- array_chunk — Sépare un tableau en tableaux de taille inférieure
- array_combine — Crée un tableau à partir de deux autres tableaux
- array_count_values — Compte le nombre de valeurs d'un tableau
- array_diff_assoc — Calcule la différence de deux tableaux, en prenant aussi en compte les clés
- array_diff_key — Calcule la différence de deux tableaux en utilisant les clés pour comparaison
- array_diff_uassoc — Calcule la différence entre deux tableaux associatifs, à l'aide d'une fonction de rappel
- array_diff_ukey — Calcule la différence entre deux tableaux en utilisant une fonction de rappel sur les clés pour comparaison
- array_diff — Calcule la différence entre deux tableaux
- array_fill_keys — Remplit un tableau avec des valeurs, en spécifiant les clés
- array_fill — Remplit un tableau avec une même valeur
- array_filter — Filtre les éléments d'un tableau grâce à une fonction utilisateur
- array_flip — Remplace les clés par les valeurs, et les valeurs par les clés
- array_intersect_assoc — Calcule l'intersection de deux tableaux avec des tests sur les index
- array_intersect_key — Calcule l'intersection de deux tableaux en utilisant les clés pour comparaison
- array_intersect_uassoc — Calcule l'intersection de deux tableaux avec des tests sur les index, compare les index en utilisant une fonction de rappel
- array_intersect_ukey — Calcule l'intersection de deux tableaux en utilisant une fonction de rappel sur les clés pour comparaison
- array_intersect — Calcule l'intersection de tableaux
- array_key_exists — Vérifie si une clé existe dans un tableau
- array_keys — Retourne toutes les clés ou un ensemble des clés d'un tableau
- array_map — Applique une fonction sur les éléments d'un tableau
- array_merge_recursive — Combine plusieurs tableaux ensemble, récursivement
- array_merge — Fusionne plusieurs tableaux en un seul
- array_multisort — Trie les tableaux multidimensionnels
- array_pad — Complète un tableau avec une valeur jusqu'à la longueur spécifiée
- array_pop — Dépile un élément de la fin d'un tableau
- array_product — Calcule le produit des valeurs du tableau
- array_push — Empile un ou plusieurs éléments à la fin d'un tableau
- array_rand — Prend une ou plusieurs valeurs, au hasard dans un tableau
- array_reduce — Réduit itérativement un tableau
- array_replace_recursive — Replaces elements from passed arrays into the first array recursively
- array_replace — Remplace les éléments d'un tableau par ceux d'autres tableaux
- array_reverse — Inverse l'ordre des éléments d'un tableau
- array_search — Recherche dans un tableau la clé associée à une valeur
- array_shift — Dépile un élément au début d'un tableau
- array_slice — Extrait une portion de tableau
- array_splice — Efface et remplace une portion de tableau
- array_sum — Calcule la somme des valeurs du tableau

- array_udiff_assoc — Calcule la différence entre des tableaux avec vérification des index, compare les données avec une fonction de rappel
- array_udiff_uassoc — Calcule la différence de deux tableaux associatifs, compare les données et les index avec une fonction de rappel
- array_udiff — Calcule la différence entre deux tableaux en utilisant une fonction rappel
- array_uintersect_assoc — Calcule l'intersection de deux tableaux avec des tests sur l'index, compare les donnée en utilisant une fonction de rappel
- array_uintersect_uassoc — Calcule l'intersection de deux tableaux avec des tests sur l'index, compare les données et les indexs des 2 tableaux en utilisant une fonction de rappel
- array_uintersect — Calcule l'intersection de deux tableaux, compare les données en utilisant une fonction de rappel
- array_unique — Dédoublonne un tableau
- array_unshift — Empile un ou plusieurs éléments au début d'un tableau
- array_values — Retourne toutes les valeurs d'un tableau
- array_walk_recursive — Applique une fonction de rappel récursivement à chaque membre d'un tableau
- array_walk — Exécute une fonction sur chacun des éléments d'un tableau
- array — Crée un tableau
- arsort — Trie un tableau en ordre inverse
- asort — Trie un tableau et conserve l'association des index
- compact — Crée un tableau à partir de variables et de leur valeur
- count — Compte tous les éléments d'un tableau ou le nombre de propriétés d'un objet
- current — Retourne l'élément courant du tableau
- each — Retourne chaque paire clé/valeur d'un tableau
- end — Positionne le pointeur de tableau en fin de tableau
- extract — Importe les variables dans la table des symboles
- in_array — Indique si une valeur appartient à un tableau
- key — Retourne une clé d'un tableau associatif
- krsort — Trie un tableau en sens inverse et suivant les clés
- ksort — Trie un tableau suivant les clés
- list — Assigne des variables comme si elles étaient un tableau
- natcasesort — Trie un tableau avec l'algorithme à "ordre naturel" insensible à la casse
- natsort — Trie un tableau avec l'algorithme à "ordre naturel"
- next — Avance le pointeur interne d'un tableau
- pos — Alias de current
- prev — Recule le pointeur courant de tableau
- range — Crée un tableau contenant un intervalle d'éléments
- reset — Remet le pointeur interne de tableau au début
- rsort — Trie un tableau en ordre inverse
- shuffle — Mélange les éléments d'un tableau
- sizeof — Alias de count
- sort — Trie un tableau
- uasort — Trie un tableau en utilisant une fonction de rappel
- uksort — Trie un tableau par ses clés en utilisant une fonction de rappel
- usort — Trie un tableau en utilisant une fonction de comparaison

14 Fonctions sur les chaînes de caractères

14.1 Sommaire

- addcslashes — Ajoute des slash dans une chaîne, à la mode du langage C
- addslashes — Ajoute des antislashs dans une chaîne
- bin2hex — Convertit des données binaires en représentation hexadécimale
- chop — Alias de rtrim
- chr — Retourne un caractère à partir de son code ASCII
- chunk_split — Scinde une chaîne
- convert_cyr_string — Convertit une chaîne d'un jeu de caractères cyrillique à l'autre
- convert_uudecode — Décode une chaîne au format uuencode
- convert_uuencode — Encode une chaîne de caractères en utilisant l'algorithme uuencode
- count_chars — Retourne des statistiques sur les caractères utilisés dans une chaîne
- crc32 — Calcule la somme de contrôle CRC32
- crypt — Hachage à sens unique (indéchiffrable)
- echo — Affiche une chaîne de caractères
- explode — Coupe une chaîne en segments
- fprintf — Écrit une chaîne formatée dans un flux
- get_html_translation_table — Retourne la table de traduction des entités utilisée par htmlspecialchars et htmlentities
- hebrev — Convertit un texte logique hébreux en texte visuel
- hebrevc — Convertit un texte logique hébreux en texte visuel, avec retours à la ligne
- html_entity_decode — Convertit toutes les entités HTML en caractères normaux
- htmlentities — Convertit tous les caractères éligibles en entités HTML
- htmlspecialchars_decode — Convertit les entités HTML spéciales en caractères
- htmlspecialchars — Convertit les caractères spéciaux en entités HTML
- implode — Rassemble les éléments d'un tableau en une chaîne
- join — Alias de implode
- lcfirst — Met le premier caractère en minuscule
- levenshtein — Calcule la distance Levenshtein entre deux chaînes
- localeconv — Lit la configuration locale
- ltrim — Supprime les espaces (ou d'autres caractères) de début de chaîne
- md5_file — Calcule le md5 d'un fichier
- md5 — Calcule le md5 d'une chaîne
- metaphone — Calcule la clé metaphone
- money_format — Met un nombre au format monétaire
- nl_langinfo — Rassemble des informations sur la langue et la configuration locale
- nl2br — Insère un retour à la ligne HTML à chaque nouvelle ligne
- number_format — Formate un nombre pour l'affichage
- ord — Retourne le code ASCII d'un caractère
- parse_str — Analyse une requête HTTP
- print — Affiche une chaîne de caractères
- printf — Affiche une chaîne de caractères formatée
- quoted_printable_decode — Convertit une chaîne quoted-printable en chaîne 8 bits
- quoted_printable_encode — Convertit une chaîne 8 bits en une chaîne quoted-printable
- quotemeta — Protège les métacaractères
- rtrim — Supprime les espaces (ou d'autres caractères) de fin de chaîne
- setlocale — Modifie les informations de localisation
- sha1_file — Calcule le sha1 d'un fichier

- sha1 — Calcule le sha1 d'une chaîne de caractères
- similar_text — Calcule la similarité de deux chaînes
- soundex — Calcule la clé soundex
- sprintf — Retourne une chaîne formatée
- sscanf — Analyse une chaîne à l'aide d'un format
- str_getcsv — Analyse une chaîne de caractères CSV dans un tableau
- str_ireplace — Version insensible à la casse de str_replace
- str_pad — Complète une chaîne jusqu'à une taille donnée
- str_repeat — Répète une chaîne
- str_replace — Remplace toutes les occurrences dans une chaîne
- str_rot13 — Effectue une transformation ROT13
- str_shuffle — Mélange les caractères d'une chaîne de caractères
- str_split — Convertit une chaîne de caractères en tableau
- str_word_count — Compte le nombre de mots utilisés dans une chaîne
- strcasecmp — Comparaison insensible à la casse de chaînes binaires
- strchr — Alias de strstr
- strcmp — Comparaison binaire de chaînes
- strcoll — Comparaison de chaînes localisées
- strcspn — Trouve un segment de chaîne ne contenant pas certains caractères
- strip_tags — Supprime les balises HTML et PHP d'une chaîne
- stripcslashes — Décode une chaîne encodée avec addcslashes
- stripos — Recherche la première occurrence dans une chaîne, sans tenir compte de la casse
- stripslashes — Supprime les antislashs d'une chaîne
- stristr — Version insensible à la casse de strstr
- strlen — Calcule la taille d'une chaîne
- strnatcasecmp — Comparaison de chaînes avec l'algorithme d'"ordre naturel" (insensible à la casse)
- strnatcmp — Comparaison de chaînes avec l'algorithme d'"ordre naturel"
- strncasecmp — Compare en binaire des chaînes de caractères
- strncmp — Comparaison binaire des n premiers caractères
- strpbrk — Recherche un ensemble de caractères dans une chaîne de caractères
- strpos — Trouve la position d'un caractère dans une chaîne
- strrchr — Trouve la dernière occurrence d'un caractère dans une chaîne
- strrev — Inverse une chaîne
- strripos — Trouve la position de la dernière occurrence d'une chaîne dans une autre, de façon insensible à la casse
- strrpos — Trouve la position de la dernière occurrence d'une sous-chaine dans une chaîne
- strspn — Trouve la longueur du premier segment d'une chaîne contenant tous les caractères d'un masque donné
- strstr — Trouve la première occurrence dans une chaîne
- strtok — Coupe une chaîne en segments
- strtolower — Renvoie une chaîne en minuscules
- strtoupper — Renvoie une chaîne en majuscules
- strtr — Remplace des caractères dans une chaîne
- substr_compare — Compare deux chaînes depuis un offset jusqu'à une longueur en caractères
- substr_count — Compte le nombre d'occurrences de segments dans une chaîne
- substr_replace — Remplace un segment dans une chaîne
- substr — Retourne un segment de chaîne
- trim — Supprime les espaces (ou d'autres caractères) en début et fin de chaîne

- ucfirst — Met le premier caractère en majuscule
- ucwords — Met en majuscule la première lettre de tous les mots
- vfprintf — Écrit une chaîne formatée dans un flux
- vprintf — Affiche une chaîne formatée
- vsprintf — Retourne une chaîne formatée
- wordwrap — Effectue la césure d'une chaîne

15 Fonctions diverses

15.1 Sommaire

- connection_aborted — Indique si l'internaute a abandonné la connexion HTTP
- connection_status — Retourne les bits de statut de la connexion HTTP
- connection_timeout — Indique si le script a expiré
- constant — Retourne la valeur d'une constante
- define — Définit une constante
- defined — Vérifie l'existence d'une constante
- die — Alias de la fonction exit
- eval — Exécute une chaîne comme un script PHP
- exit — Affiche un message et termine le script courant
- get_browser — Indique les capacités du navigateur client
- __halt_compiler — Stoppe l'exécution du compilateur
- highlight_file — Colorisation syntaxique d'un fichier
- highlight_string — Applique la syntaxe colorisée à du code PHP
- ignore_user_abort — Active l'interruption de script sur déconnexion du visiteur
- pack — Compacte des données dans une chaîne binaire
- php_check_syntax — Vérifie la syntaxe PHP (et exécute) d'un fichier spécifique
- php_strip_whitespace — Retourne la source sans commentaires, ni espaces blancs
- show_source — Alias de highlight_file
- sleep — Arrête l'exécution durant quelques secondes
- sys_getloadavg — Récupère la charge moyenne du système
- time_nanosleep — Attendre pendant un nombre de secondes et de nanosecondes
- time_sleep_until — Arrête le script pendant une durée spécifiée
- uniqid — Génère un identifiant unique
- unpack — Déconditionne des données depuis une chaîne binaire
- usleep — Arrête l'exécution durant quelques microsecondes

16 Les Superglobales

Les Superglobales — Les Superglobales sont des variables internes qui sont toujours disponibles, quel que soit le contexte

16.1 Description

Plusieurs variables prédéfinies en PHP sont "superglobales", ce qui signifie qu'elles sont disponibles quel que soit le contexte du script. Il est inutile de faire **global $variable;** avant d'y accéder dans les fonctions ou les méthodes.

Les variables superglobales sont :

- *$GLOBALS*
- *$_SERVER*
- *$_GET*
- *$_POST*
- *$_FILES*
- *$_COOKIE*
- *$_SESSION*
- *$_REQUEST*
- *$_ENV*

Note: Disponibilité des variables

Par défaut, toutes les superglobales sont disponibles, et seules les directives de configuration peuvent les rendre indisponibles. Pour plus d'informations, reportez-vous à la documentation sur l'ordre des variables.

Note: Gérer la directive register_globals ('off' par défaut / obsolète depuis PHP 5.3)

Si la directive obsolète register_globals est définie à *on*, alors les variables simples seront également disponibles dans le contexte global du script. Par exemple, *$_POST['foo']* existera également sous la forme *$foo*. Vivement déconseillé! Utilisez directement *$_POST['foo']* ou passez par une variable temporaire *$temp = $_POST['foo']*