



© Philippe ROOSE / IUT de Bayonne / UPPA  
Partiellement sur IUT en Ligne ([www.iutenligne.net](http://www.iutenligne.net))  
2005-2011



## Langage de programmation PHP

<b>1</b>	<b><i>Introduction.....</i></b>	<b>5</b>
1.1	Origine de ce langage.....	5
1.2	Atouts.....	5
1.3	PHP dans une page Web .....	5
<b>2</b>	<b><i>Variables, opérateurs et expressions .....</i></b>	<b>6</b>
2.1	Identifiants.....	6
2.2	Types de données .....	6
2.3	Portée des variables .....	8
2.4	Assignation de valeurs aux variables .....	8
2.5	Constantes .....	9
2.6	Opérateurs logiques, binaires, autres .....	9
<b>3</b>	<b><i>Instructions de contrôle .....</i></b>	<b>10</b>
3.1	If, then, else.....	10
3.2	Switch.....	10
3.3	Boucles for, do...while, for.....	12
3.4	Instructions break et exit .....	12
<b>4</b>	<b><i>Tableaux.....</i></b>	<b>12</b>
4.1	Tableaux à une dimension.....	12
4.2	Initialisation.....	13
4.3	Tri de tableaux .....	14
4.4	Tableaux à dimensions multiples.....	14
<b>5</b>	<b><i>Entrées/Sorties .....</i></b>	<b>15</b>
5.1	Envoi vers le navigateur .....	15
5.2	Récupération des données d'un formulaire.....	15
5.3	Récupération des variables d'environnement.....	16
<b>6</b>	<b><i>Réutilisation de code .....</i></b>	<b>17</b>
6.1	Fonctions include/require .....	17
6.2	Déclaration et utilisation de fonctions.....	17
6.2.1	Déclarations .....	17

6.2.2	Arguments.....	18
6.2.3	Retour de paramètres .....	18
7	<i>Lecture/écriture de fichiers.....</i>	<i>18</i>
7.1	Transfert de fichiers (upload).....	20
8	<i>Fonctions diverses.....</i>	<i>21</i>
8.1	Fonctions de traitement sur des données.....	21
8.2	Fonctions mathématiques .....	22
8.3	Gestion de date, heure, temps.....	22
8.4	Fonctions orientées serveur .....	23
9	Sérialisation.....	23
10	<i>Fonctions orientées réseau.....</i>	<i>24</i>
10.1	Mail.....	24
10.2	FTP .....	24
11	<i>Base de données.....</i>	<i>25</i>
11.1	Ouverture d'une base .....	25
11.2	Requêtes SQL .....	25
11.3	Récupération de données.....	26
11.4	Autres fonctions.....	26
11.5	Utilisation d'ODBC.....	26
12	<i>Manipulations d'images.....</i>	<i>27</i>
12.1	Séquence de génération d'une image.....	27
12.2	Création d'une image.....	27
12.3	Gestion des couleurs.....	28
12.4	Gestion des formes .....	28
12.5	Gestion des caractères/chaînes de caractères .....	29
12.6	Gestion des couleurs.....	29
12.7	Manipulation d'images .....	29
12.8	Récupération des images .....	31
13	<i>Programmation objet : PHP et les classes (PHP 4/PHP 5).....</i>	<i>31</i>
13.1	Syntaxe de la programmation objet .....	31
14	<i>Gestion des Exceptions (PHP 5).....</i>	<i>33</i>
15	<i>Contrôle de sessions.....</i>	<i>34</i>

15.1	Création de cookies .....	34
15.2	Création de variables de sessions.....	35
16	<i>PHP-Ajax</i> .....	36
17	<i>Webservices &amp; SOAP</i> .....	41
18	<i>PHP en chiffre</i> .....	45
19	<i>Conclusion</i> .....	46

# 1 Introduction

## 1.1 Origine de ce langage

PHP, écrit en C, est né en 1994 avec le site de Rasmus Lerdorf. Il désirait mettre son CV en ligne et garder une trace des utilisateurs. Dès le départ, il supportait les requêtes SQL, et devant le nombre de personnes intéressées, il a décidé de mettre en ligne la version 1.0 de PHP (Personal Home Page). Cette version permettait de plus de contrôler les saisies des utilisateurs et de remplacer certaines commandes.

Devant le succès et le nombre de requêtes des programmeurs, en utilisant Bison/Yacc est mis en œuvre PHP 2.0. Cette version ajoutera des structures conditionnelles, des boucles et encore bien d'autres améliorations. Elle permettait également au programmeur d'intégrer des instructions de programmation puissantes à l'intérieur de leur code HTML.

A l'origine, PHP signifiait *Personal Home Page* mais ce nom a changé pour être mis à la mode GNU (*Gnu is not Unix*), c'est à dire un acronyme récursif. Il signifie maintenant *PHP : Hypertext Preprocessor*.

Le script PHP est maintenant directement compilé dans le serveur Web (Apache par exemple), les instructions sont donc exécutées sur le serveur lui même.

Ce pauvre Rasmus devenant un peu débordé, il a fait appel à d'autres programmeurs de renom et le 6 juin 1998 est né PHP 3.0, point de départ de l'invasion PHP !

## 1.2 Atouts

On peut se demander pourquoi utiliser du PHP alors qu'il existe bien d'autres technologies et en particulier le Perl. Microsoft propose les ASP (*Active Server Pages*) avec son serveur Web IIS (*Internet Information Server*) et Sun propose les JSP (*Java Server Pages*) Il existe d'autres solutions plus ou moins propriétaires, à la fois gratuites et payantes (comme *Coldfusion*).

Le choix de PHP se réalise vite : c'est le meilleur ! Il intègre tous les avantages des autres solutions. Son code est rapide à produire et à exécuter. Il tourne sur différents serveurs (Apache, IIS, WebTen, ...) et différents systèmes d'exploitation (UNIX, Windows et Macintosh, etc.). Il permet non seulement des performances élevées mais également possède des interfaces vers la plupart des SGBD du marché, il intègre de nombreuses bibliothèques pour faciliter le développement de tâches Web et enfin il possède un autre avantage, qui, s'il n'est pas son principal, en est un de plus : il est gratuit.

## 1.3 PHP dans une page Web

De manière classique, voici l'éternel "Hello World" dans sa version PHP. La commande *print* (ou *echo*) permet d'afficher le texte entre parenthèses et/ou guillemets.

```
<HTML><BODY>
<?php
print "Hello World<P>";
?>
</BODY></HTML>
```

affichera :Hello World

Le code PHP est placé directement dans le code source HTML, il est encadré par deux balises spéciales qui sont destinées à être reconnues par l'interpréteur PHP. On aurait tout aussi bien pu utiliser les balises

<SCRIPT LANGUAGE="php"> et </SCRIPT> comme on le fait pour du JavaScript.

Attention,

Il ne faut pas confondre PHP et JavaScript. Il s'agit de deux choses différentes. PHP est un langage de script destiné à être exécuté par le serveur, alors que le JavaScript est chargé et exécuté dans le navigateur, donc “côté client”. PHP est donc comparable aux ASP de Microsoft ou aux CGI “standards” dans le sens où ils s'exécutent tous les deux “côté serveur”.

Enfin, il est également possible d'utiliser les tags sous leur forme courte (<? et ?>), mais cette possibilité n'est pas activée par défaut. Il faudra donc modifier le paramétrage de PHP sur la machine serveur.

Quand on affiche dans le navigateur le source HTML d'une page générée par PHP, on ne voit que le code HTML résultant. Si vous regardez le source de la page précédente dans votre navigateur, vous obtiendrez ça :

```
<HTML><BODY>
Hello World<P>
</BODY></HTML>
```

## 2 Variables, opérateurs et expressions

### 2.1 Identifiants

PHP propose trois types de données : les entiers, les décimaux, et les chaînes de caractères. PHP respecte la casse lors de l'identification des variables. Ainsi, *maVariable* est différent que *MaVariable*. Une variable est toujours précédée du signe \$ que les amateurs de Shell connaissent bien.

### 2.2 Types de données

En fait, quand on utilise du PHP, on ne s'occupe pas de distinguer le types des variables. Elles prennent le type des données qui leur sont affectées. Néanmoins, il est possible de contourner ceci si nécessaire en utilisant la fonction *settype* qui attribue un type spécifique à une variable. Il existe une autre solution par l'utilisation des fonctions *intval*, *doubleval* et *strval* permettant de réaliser des conversions de chaînes en entiers par exemple pour *intval*.

Il existe néanmoins un certain nombre de types de données :

- *Entier*: pour les nombres entiers,
- *Float*(ou *Double*) : nombres réels,
- *Chaînes*: pour les chaînes de caractères,
- *Booléen*: pour des valeurs de type *vrai* ou *faux*,
- *Tableau*: pour stocker des données du même type,
- *Objet*: pour stocker des instances de classes,

Deux types particuliers sont également proposés :

- *NULL*: pour des valeurs de type *non définies*,
- *ressource*: retournées fréquemment par des fonctions accédant à des ressources externes (fichiers, BD, etc.). elles sont rarement manipulées.

#### Exemple

```
< ?  
$temperature = " 30°" ;  
settype($temperature, " double ") ;  
print " Double : $temperature <BR> " ;  
settype("$temperature", "integer") ;  
print " Integer : $temperature <BR> " ;  
settype("$temperature", "string") ;  
print " String : $temperature <BR>";  
  
$temperature = " 30°" ;  
print " <BR>String <BR>";  
print "strval($temperature)";  
print "<BR>Double <BR>";  
print "doubleval($temperature)";  
print "<BR>Integer <BR>";  
print "intval($temperature)";  
  
$foo = "0"; // $foo est une chaîne  
$foo++; // $foo est une chaîne  
$foo += 1; // $foo est maintenant un entier  
$foo = $foo + 1.3; // foo est maintenant un double  
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)  
$foo = 5 + "10 Small Pigs"; // $foo is integer (15)  
?>
```

Il est également possible de réaliser ceci de manière implicite en utilisant la coercition vue en C. Ainsi, *intval(\$temperature)* et *(integer) \$temperature* sont identiques.

Afin de connaître le type des variables, un certain nombre de fonctions sont disponibles et retournent *true* ou *false* :

- *is\_array()*,
- *is\_double()*, *is\_float()*, *is\_real()* - c'est la même fonction,
- *is\_long()*, *is\_int()*, *is\_integer()*- c'est la même fonction,
- *is\_string()*,
- *is\_object()*,
- *is\_resource()*,
- *is\_null()*,
- *is\_scalar()* - si la variable est scalaire, c'est à dire si c'est un entier, une chaîne, ou un double.
- *is\_numeric()* - si la variable est un nombre ou une chaîne numérique,
- *is\_callable()*- le la variable est un nom de fonction valide.

Trois fonctions permettent également de connaître l'état de variables :

- *isset()* : retourne *true* ou *false* selon que la variable passée en paramètre existe ou pas,
- *unset()* : supprime la variable passée en paramètre,
- *empty()* : retourne un booléen selon que la variable est non vide et non nulle.

Enfin, trois fonctions aident à la réalisation du transtypage de variables :

- *int intval(variable)*,
- *float floatval(variable)*,
- *string strval(variable)*.

Chacune de ces fonctions retourne la valeur transtypée de l'argument passé en paramètre.

### 2.3 Portée des variables

A l'inverse d'autres langages comme le C, il est inutile en PHP de déclarer les variables avant de les utiliser. C'est l'une des caractéristiques des langages interprétés.

Il est à noter que comme dans la plupart des langages, la portée des variables égale celle du bloc où elle a été utilisée pour la première fois (ce qui équivaut à une déclaration). Dans les cas où une variable déclarée dans une fonction est nécessaire au niveau global, il est possible de lui étendre sa portée de manière à ce qu'elle continue d'exister une fois la fonction terminée. On pourra pour cela la déclarer comme globale :

Exemple

```
function maFonction() {
    global maVar ;
    ...
}
```

Une autre manière plus élégante est d'utiliser le mot clé *static* qui dans la plupart des cas permet de s'affranchir de l'utilisation peu élégante de *global*.

Exemple

```
static mavar ; // static a le même comportement qu'en C
```

### 2.4 Assignment de valeurs aux variables

Lorsque l'on assigne une valeur à une variable (=), son type change pour correspondre au type de données qui lui est affecté. Ce fonctionnement est l'inverse du C qui lui essaie de convertir la donnée au type de la variable.

Exemple

```
$machaine1 = " Gérard " ;
$machaine2 = " Dupont" ;
integer var1 ; // déclaration de variable avec typage
$var1 = 12 ;
$var1 = -45 ;
double var1 ;
$var1 = -3.14
```



le signe \ permet d'éviter l'affichage du caractère suivant :

\ "	Guillemets doubles
\\	Caractère antislash
\n	Nouvelle ligne
\r	Retour chariot
\t	Tabulation
\x00 - \xFF	Caractères hexadécimaux

Il est à noter qu'il est tout à fait possible d'inclure une variable dans une chaîne de caractères :

Exemple

```
print " Je suis : $machaine1 <BR> " ; // Affichera " Je suis Gérard "
```

## 2.5 Constantes

Il existe un certain nombre de constantes créées automatiquement par PHP (nous en verrons quelques unes plus loin). Il est également possible de définir ses propres constantes avec *define(nomconstante, valeur constante)*. En réalité, une constante est une variable dont le contenu ne peut être paramétré qu'une seule fois. Comme les variables, les constantes sont par défaut sensibles à la casse. Associé à la fonction *define*, une autre fonction *defined('nom constante')* peut être utilisée afin de tester l'existence d'une constante (elle retournera *true* ou *false* selon le cas).

Exemple

```
define("MACONSTANTE", "Hello World") ;  
if (defined("MACONSTANTE")) {  
    print "La valeur de ma constante est : ".MACONSTANTE //Attention, on accède au contenu sans le $.  
}
```

## 2.6 Opérateurs logiques, binaires, autres

Nous n'allons pas décrire l'utilisation de chacun des opérateurs, ils sont quasi-identiques au C.

Opérateurs arithmétiques :

+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo
++	Incrément
--	Décrément

Opérateurs logiques

<	Inférieur à
>	Supérieur à
<=	Inférieur ou égal à
>=	Supérieur ou égal à
==	Égal à
!=	Différent
AND &&	Et

OR	Ou
XOR	Ou exclusif
!	Sauf

Autres opérateurs

Le `.` permet de faire une concaténation (*print \$var1.\$var2* affichera le contenu de var1 concaténé à var2).

### 3 Instructions de contrôle

Pas de surprises en PHP pour l'utilisation des instructions de contrôle. Elles sont identiques à celles déjà vues en C.

#### 3.1 *If, then, else*

```
if (expression) {
    Bloc si expression vraie
}
else {
    Bloc si expression fausse
}
```

ou bien

```
if (expression 1) [
    Bloc si expression vraie
}
else if (expressio2) {
    Bloc si expression2 vraie
}
else {
    Bloc si expression2 est fausse
}
```

#### 3.2 *Switch*

```
switch (expression) {
    case exprt1: ...
    default : ...
}
```

Exemple d'utilisation de *switch*

```
<?php

$note=11;
$extension="fr";
```

```

switch($note) {
    case $note>10:
        echo ("Bac obtenu");
        break;

    case $note>=12:
        echo ("Bac obtenu avec mention AB");
        break;

    case $note>=14:
        echo ("Bac obtenu avec mention B");
        break;

    case $note>=16:
        echo ("Bac obtenu avec mention B");
        break;

    default :
        echo ("Bac non obtenu");
        break;
}

print "<br><br>";

switch($extension) {
    case "fr" : $pays="France";
        break;
    case "es" : $pays="Espagne";
        break;
    case "Be" : $pays="Belgique";
        break;
    default : $pays="Inconnu";
        break;
}
print "Pays visité : $pays";

php?>

```

Il n'est pas nécessaire de mettre des accolades pour chacun des cas.

### 3.3 Boucles *for*, *do...while*, *for*

```
for (initialisation; expression; incrémentation) {  
...  
}  
  
while (expression) {  
...  
}  
  
do {  
...  
} while (expression);
```

### 3.4 Instructions *break* et *exit*

Les instructions *break* et *exit* ont le même comportement qu'en C.

## 4 Tableaux

Pas de surprises avec l'utilisation des tableaux, l'utilisation est tout à fait classique.

### 4.1 Tableaux à une dimension

```
< ?  
$bab[0] = "Biarritz";  
$bab[1] = "Anglet";  
$bab[2] = "Bayonne";  
  
print "Je vis à $bab[1]. <BR>" ;  
?>
```

Dans les cas où l'on ne désire pas se préoccuper de mentionner les index, PHP le fera à votre place :

```
$bab [ ] = 'Biarritz';  
$bab [ ] = "Anglet";  
$bab [ ] = "Bayonne";
```

Si l'on désire savoir combien de valeurs ont été entrées dans un tableau, on utilisera la fonction *count(tableau)* :

```
$limiteIndex = count ($bab) ;
```

et pour les afficher :

```
for ($index = 0 ; $index < $limiteIndex ; $index++) {
```

```
print ("${bab[$index]} <BR>");  
}
```

Il est également possible d'indexer les tableaux autrement qu'avec des entiers :

```
print "Exemples de tableaux : <br>";  
$capitale[fr]="Paris";  
$capitale["es"]="Madrid";  
  
print "La capitale de l'Espagne est : $capitale[es]";
```

A noter que les guillemets ne sont pas nécessaires à l'initialisation, mais ils ne doivent pas être mis à l'utilisation...autant dire qu'il est inutile de les mettre !

## 4.2 Initialisation

Il est également possible de déclarer un tableau avec ses initialisations. Ce type de tableau est appelé associatif :

```
< ?  
$mois=array(2=> `Janvier`, `Février`, `Mars`,  
`Avril`, `Mai`, `Juin`, `Juillet`, `Août`, `Septembre`, `Octobre`, `Novembre`, `Décembre`);  
?>  
  
print `Mois de $mois[5]` ; affichera ` Mois de Avril `.
```

L'opérateur => permet ici de faire démarrer le tableau à l'index 2 et non pas 0 comme c'est le cas par défaut.

Enfin, il n'est pas nécessaire d'avoir des tableaux avec des indices numériques successifs.

```
$tabbizarre[12] = "pas bien grand";  
$tabbizarre[1025] = "plutot grand";  
$tabbizarre[15215] = "très grand";
```

Afin d'accéder aux éléments, un ensemble de fonctions sont définies :

```
print "Key : ".key($tabbizarre). " <br>";  
print "Current : ".current($tabbizarre). " <br>";  
print "Next : ".next($tabbizarre). " <br>";  
print "Prev : ".prev($tabbizarre). " <br>";  
  
> Key : 12  
> Current : pas bien grand  
> Next : plutot grand
```

```
> Prev : pas bien grand
```

Afin de réaliser un parcours efficace de ce genre de tableaux, deux fonctions sont bien utiles. La fonction *each* permet de renvoyer uniquement les éléments du tableau contenant les données. La fonction *list* (\$indice, \$valeur) récupère l'indice et la valeur de l'élément.

```
while (list($indice,$contenu) = each ($tabbizarre))  
    print "$contenu <br>";
```

### 4.3 Tri de tableaux

La fonction *sort()* permet de trier dans l'ordre alphabétique le contenu d'un tableau. Sa cousine *ksort()* fait de même mais dans l'ordre inverse. Les deux autres *asort()* et *arsort()* font la même chose mais pour des tableaux avec indice entiers.

```
$tabatrier[] = "albert";  
$tabatrier[] = "george";  
$tabatrier[] = "bacus";  
  
sort($tabatrier);  
while (list($indice,$contenu) = each ($tabatrier))  
    print "$contenu <br>";  
  
print "<br>";  
  
rsort($tabatrier);  
while (list($indice,$contenu) = each ($tabatrier))  
    print "$contenu <br>";  
  
> albert  
> bacus  
> george  
  
> george  
> bacus  
> albert
```

### 4.4 Tableaux à dimensions multiples

L'utilisation de tableaux à dimensions multiples est là encore semblable aux autres langages. Ainsi si un tableau représente un échiquier, nous accéderons aux valeurs en faisant par exemple :

```
$echiquier[5][4] ;
```

## 5 Entrées/Sorties

PHP étant un langage de script, il doit pouvoir être capable de communiquer avec son environnement. Les scripts PHP étant exécutés dans le contexte d'un navigateur Web, il est important de se rappeler que toute sortie vers le navigateur sera interprétée comme du texte HTML.

### 5.1 Envoi vers le navigateur

Trois fonctions permettent d'envoyer du texte vers le navigateur : *echo*, *print* et *printf*. L'intérêt principal de cette dernière fonction est, nous le verrons plus tard, la possibilité de spécifier un format de sortie afin d'éviter d'envoyer la valeur telle quelle.

### 5.2 Récupération des données d'un formulaire

Si l'envoi de données vers le navigateur est simple en utilisant les trois fonctions décrites précédemment, la récupération d'informations en provenance du navigateur l'est un peu moins.

Le langage HTML permet la récupération de données d'un formulaire : champs/zones de texte, listes déroulantes, cases à cocher, ...

PHP transforme tous les champs d'un formulaire en variables ce qui permettra de modifier éventuellement leur contenu.

Soit le formulaire suivant :

```
<FORM ACTION="/php/myprogramme.php" METHOD="POST">
Saisissez votre nom :
<INPUT TYPE="TEXT" NAME="NOM" SIZE="20"> MAXSIZE="50" ><BR>
<INPUT TYPE="SUBMIT" VALUE="Envoyer la requête">
</FORM>
```

Dans le fichier *myprogramme.php*, il sera possible de récupérer la valeur saisie dans le champ NOM de la manière suivante :

```
$nom = $_POST['NOM'] ;
```

Attention, vous trouverez dans de nombreux scripts une récupération de variable de formulaire directement (print "\$NOM ") ou via des tableaux à noms longs comme \$HTTP\_GET\_VARS. Cette technique employée jusqu'à peu montre ses limites avec les dernières versions de PHP (et quel que soit le mode de passage des paramètres : POST, GET, cookie, session, ...). La différence est que depuis la version 4.2.0, l'option *register\_globals* du *php.ini* qui était positionnée par défaut à ON est à OFF. Donc, autant prendre de bonnes habitudes et utiliser la nouvelle "technique". Pour vérifier la version de PHP, utilisez la fonction *phpinfo()* qui retourne une multitude d'informations. Ca vaut le coup d'œil.

Voici un récapitulatif des variables et des changements selon les versions :

Avant la version 4.10 ( <i>deprecated</i> )	Après la version 4.10
\$HTTP_GET_VARS	\$_GET
\$HTTP_POST_VARS	\$_POST
\$HTTP_POST_FILES	\$_FILES
\$HTTP_COOKIE_VARS	\$_COOKIE
\$HTTP_SESSION_VARS	\$_SESSION
\$HTTP_ENV_VARS	\$_ENV
\$HTTP_SERVER_VARS	\$_SERVER

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

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

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

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

La fonction extract() va créer les variables suivantes :

- \$nom
- \$prenom
- \$age

Type	signification
EXTR_OVERWRITE	Écrase les variables existantes
EXTR_SKIP	N'écasse pas les variables existantes
EXTR_PREFIX_SAME	Si une variable existe déjà, une nouvelle variable est créée avec un préfix donné en 3ème argument à la fonction
EXTR_PREFIX_ALL	Crée de nouvelles variables avec le préfix passé en 3ème argument pour toutes les clés du tableau
EXTR_PREFIX_INVALID	Crée de nouvelles variables avec le préfix passé en 3ème argument pour les noms de variable invalides (par exemple \$1)

### 5.3 Récupération des variables d'environnement

Tableau \$\_SERVER

- **PHP\_SELF** : Le nom du fichier du script en cour d'exécution, par rapport au document root.
- **SERVER\_NAME** : Le nom du serveur hôte qui exécute le script suivant. Si le script est exécuté sur un hôte virtuel, ce sera la valeur définie pour cet hôte virtuel.
- **DOCUMENT\_ROOT** : La racine sous laquelle le script courant est exécuté, comme défini dans la configuration du serveur.
- **REMOTE\_ADDR** : L'adresse IP du client qui demande la page courante.
- **REMOTE\_PORT** : Le port utilisé par la machine cliente pour communiquer avec le serveur web.
- **SCRIPT\_FILENAME** : Le chemin absolu jusqu'au script courant.
- **SERVER\_PORT** : Le port de la machine serveur utilisé pour les communications. Par défaut, c'est '80'. En utilisant SSL, par exemple, il sera remplacé par le numéro de port HTTP sécurisé.
- **REQUEST\_URI** : L'URI qui a été fourni pour accéder à cette page. Par exemple : '/index.html'.

Tableau \$\_FILE

- **['NOM OBJET FILE']['name']** : permet de récupérer le nom du fichier sélectionné dans un objet *file* du navigateur (exemple, fichier à télécharger).
- **['NOM OBJET FILE']['tmp\_name']** : retourne le nom du fichier temporaire stocké sur le serveur, en attente de transfert à la destination désirée.
- **['NOM OBJET FILE']['size']** : retourne la taille en octet du fichier.



- ['NOM OBJET FILE']['type'] : donne le type MIME du fichier.

## 6 Réutilisation de code

### 6.1 Fonctions include/require

Tout comme il est possible de réaliser des inclusion de fichiers en C avec la directive de compilation *#include* il est également possible de le faire en PHP avec la fonction *include(nom de fichier)* ou *require(nom de fichier)*. Ces fonctions très utiles permettent une réutilisation maximale de fonctions déjà écrites. Il est à noter que même si l'extension n'a aucune importance, la norme d'utilisation des inclusions est *.inc*.

Il n'y a pas de différence fondamentale entre *require* et *include*. La différence est que si un problème survient, *require* produira une erreur fatale alors que *include* ne produira qu'un avertissement.

La syntaxe des fichiers à inclure a son importance. Dans le cas où le texte inclus ne possède pas de balises *<? et ?>*, le texte sera inclus « tel quel », sans interprétation, ce qui peut être intéressant pour par exemple ajouter systématiquement des entête/pieds de pages à des pages HTML. Par contre si des balises sont trouvées, le code correspondant sera exécuté et inséré à l'endroit où la fonction est appelée :

Codeareutiliser.php

```
<?
print « Je suis du code à réutiliser <br> »;
?>
```

```
codequireutilise.php
print « Debut du code <br> »;
require ("codeareutiliser.php");
print « Fin du code <br>";
```

Donnera :

Debut du code

Je suis du code à réutiliser

Fin du code

### 6.2 Déclaration et utilisation de fonctions

Comme dans la plupart des langages de programmation, il est également possible d'écrire ses propres fonctions en PHP.

#### 6.2.1 Déclarations

La syntaxe est proche de ce qui est réalisé par exemple en C :

```
function nomFonction (paramètres) {
    Corps de la fonction
}
```

### 6.2.2 Arguments

Voici par exemple une fonction qui permet d'afficher en gras une chaîne de caractères :

```
function bold ( $machaine ) {  
    print "<B>" . $machaine . "</B>" ;  
}
```

Comme cela est possible en C++, nous pouvons définir des paramètres par défaut :

```
function afficher($maChaine, $couleur = 'black') {  
    print "<FONT COLOR=\ '$Color\ '> $maChaine </FONT>" ;  
}
```

Si le deuxième paramètre n'est pas précisé lors de l'appel, la fonction choisira le noir.

Par défaut, les passages de paramètres sont réalisés par recopie. Il est possible de réaliser un passage de paramètres par référence (l'original donc !) en précédant le nom du paramètre par & dans la déclaration de la fonction, à l'instar de ce qui se fait en C.

### 6.2.3 Retour de paramètres

Il est également possible de faire retourner une valeur à la fonction :

```
function retourneBold ( $machaine ) {  
    $chaineTmp = "<B>" ;  
    $chaineTmp .= $machaine ; // Le point permet de concaténer  
    $chaineTmp .= "</B>" ;  
  
    return ($chaineTmp) ;  
}
```

## 7 Lecture/écriture de fichiers

Comme dans la plupart des langages, afin de travailler avec un fichier, il est nécessaire d'ouvrir l'accès au fichier, d'y accéder, puis de le fermer.

Il est possible en PHP d'ouvrir un fichier local (fonction *fopen*) ou distant et le désignant par une URL et un protocole (*http* ou *ftp*).

Les différents modes d'ouvertures sont les suivants :

- ❑ *r*: ouverture en lecture seule à partir du début,
- ❑ *r+*: ouverture en lecture/écriture à partir du début,
- ❑ *w*: ouverture en écriture seule. Si le fichier existe déjà, il est écrasé, sinon, il est créé (si possible),
- ❑ *w+*: ouverture en lecture/écriture. Si le fichier existe déjà, il est écrasé, sinon, il est créé (si possible),
- ❑ *a*: ouverture en ajout uniquement. Les données sont écrites à la fin du fichier, s'il n'existe pas, il est créé (si possible),
- ❑ *a+*: ouverture en lecture et ajout. Les données sont écrites à la fin du fichier, s'il n'existe pas, il est créé (si possible).

Afin de revenir au début du fichier, il est nécessaire d'utiliser la fonction *fseek*. Le premier paramètre est l'identifiant du fichier retourné par *fopen*, le second paramètre est l'offset (décalage en nombre d'octets, à partir du début du fichier - exprimé par un entier).

Nous n'allons pas le décrire ici, mais il est également possible d'accéder à des fichiers en ouvrant un canal ou *pipe* avec la fonction *popen* ou en ouvrant une connexion par *socket* avec *fsockopen*.

Voici un exemple d'ouverture fichier :

```
< ?
$monFichier = fopen('monfichier.txt','w') ; // ouverture en écriture
if ( !($monfichier)) {
    print(' Impossible de créer le fichier \n') ;
    exit ;
}

fputs($monfichier, 'ligne 1') ; // on écrit deux lignes
fputs($monfichier, 'ligne 2') ;

fclose($monfichier) ; // on ferme le fichier, on libère les ressources

$monfichier = fopen('monfichier.txt', 'r') ; // ouverture en lecture
if ( !($monfichier)) {
    print(' Impossible d'ouvrir le fichier ') ;
    exit ;
}
while ( !feof($monfichier) ) {
    $ligne = fgets($monfichier,255); // 255 caractères max. ou bien fin de ligne.
    print ' $ligne <BR> ' ;
}

fclose ($monfichier) ;

?>
```

On peut également accéder à des fichiers distants en passant par une URL :

```
$url = 'http://wwwbay.univ-pau.fr/~roose/index.shtml' ;
fopen($url) ;
```

Il est parfois nécessaire de lire un fichier pour l'afficher par la suite dans le navigateur. La fonction *readfile* permet ceci en renvoyant le contenu du fichier vers le navigateur. La valeur retournée est le nombre d'octets lus. Si le nom du fichier commence par **http://** ou **ftp://**, l'accès à ce dernier sera fait via l'un de ces protocoles. Dans le cas où aucun protocole n'est spécifié, l'accès se fait sur le système local, celui sur lequel est hébergée la page PHP correspondante.

Il existe toute une série de fonctions associées aux fichiers, comme :

- ❑ *fgetcsv*(*descripteur*, *longueur max*, *séparateur*) : permet de lire une chaîne de caractère en ne prenant pas le marqueur de fin de ligne comme fin, mais le séparateur précisé. Très pratique pour lire des contenus exportés au format csv.
- ❑ *file\_exists*(*descripteur de fichier*) : teste l'existence d'un fichier (ne fonctionne pas avec des accès HTTP ou FTP).
- ❑ *filesize*('' nom fichier'') qui retourne la taille en octets du fichier.
- ❑ *fseek* (*descripteur fichier*, *offset*) : permet de se positionner à l'octet " offset " dans le fichier. Cette fonction retourne 0 si tout c'est bien passé, -1 sinon.
- ❑ *rewind* (*descripteur*): place le pointeur au début du fichier.
- ❑ *chmod*(*fichier*, *droits unix*) : change les droits du fichier (à condition que le serveur PHP ai également les droits dessus).
- ❑ *unlink*(*nom du fichier*) : supprime ce fichier.
- ❑ *is\_dir*('' nom fichier'') ; *is\_executable*('' nom fichier'') , *is\_link*('' nom fichier'') , *is\_readable*('' nom fichier'') , *is\_writeable*('' nom fichier'') permettant de savoir si un fichier est un répertoire, exécutable, un lien, si les droits en lecture ou écriture sont valides. Lorsque les évaluations sont vérifiées, 1 est retourné, 0 sinon.
- ❑ *opendir*('' nom répertoire'') retourne un état de répertoire exploitable par la suite avec *readdir* et *closedir*.
- ❑ *readdir*(*descripteur de répertoire*), retourne le nom du fichier suivant à partir du descripteur de répertoire obtenu avec *opendir*. On fermera le répertoire avec *closedir* comme on le fait lorsque l'on ouvre un fichier.
- ❑ *fgetcsv*(*descripteur*, *nb max de car à lire*, *délimiteur*). Lorsque l'on a à faire à une fichier encodé au format CSV (chaque champs séparé par un délimiteur), il est possible d'utiliser la fonction : *fgetcsv* qui s'utilise un peu comme la fonction *explode*. Elle retourne un tableau dont chaque « case » est un champ de la ligne lue au format CSV. Sur le même format, il existe la commande *fputcsv*.

## 7.1 Transfert de fichiers (upload)

Il est fréquent de désirer transférer un fichier d'un ordinateur vers le serveur PHP. Afin de réaliser ceci, il est nécessaire d'implémenter la fonction d'*upload*.

Pour ce faire, il faut utiliser la fonction *move\_uploaded\_file*(*source*, *destination*). Il est à noter que la source est un fichier déjà présent sur le serveur mais généralement dans un répertoire temporaire. Ce fichier aura été mis dans ce répertoire à l'aide par exemple du code HTML suivant :

```
<form ENCTYPE="multipart/form-data" action="upload.php" METHOD="POST">
<input type="file" name="nomfichier"> <br>
<input type="submit" value="Télécharger">
```

On se référera au tableau \$\_FILE pour plus d'informations. Bien évidemment, on prendra soin de gérer les droits en écriture du répertoire destinataire.

Une fois le transfert réalisé (appui sur le bouton submit) la première chose à faire est de vérifier que l'opération s'est bien passée en vérifiant la présence du fichier dans le dossier temporaire; pour cela, nous avons à notre disposition la fonction *is\_uploaded\_file*(\$\_FILES['fichier']['tmp\_name']). Une fois la vérification réalisée, on copie le fichier sur notre espace web à l'aide de la fonction *move\_uploaded\_file*(*string \$filename* , *string \$destination*) qui est plus sûre que la fonction *copy*(), car elle vérifie que le fichier à copier vient bien du dossier temporaire (et donc, provient d'un formulaire d'*upload*).

## 8 Fonctions diverses

### 8.1 Fonctions de traitement sur des données

Il est fréquent d’avoir des fichiers contenant différents champs séparés par un délimiteur quelconque. Une fonction très utile dans ces cas là est la fonction *explode*. Sa syntaxe est la suivante

```
explode (" caractère délimiteur ", chaîne de donnée)
```

Prenons un exemple avec l’extrait de fichier suivant :

```
Roose | Philippe | Iut Informatique | Bayonne | Informatique  
Goudin | David | LaBri | Bordeaux | Calcul Parallèle
```

Ce fichier décrit des enseignants avec leur nom, prénom, lieu de poste, matière principale. Si l’on souhaite afficher ce fichier d’une manière “ plus jolie ”, il est nécessaire de récupérer champs après champs et d’afficher chacun d’eux comme désiré. Sans la méthode *explode*, il serait nécessaire de récupérer caractère par caractère, tester si le séparateur est atteint, si oui, ...

*explode* réalise ce travail pour nous en retournant dans un tableau l’ensemble des champs. Il ne reste plus qu’à “ boucler ” sur ce tableau pour récupérer les valeurs et les traiter.

Il est à noter qu’il existe sa réciproque, la fonction *implode* qui retourne une chaîne et qui accepte en paramètre un tableau et un délimiteur (qui peut être lui même une chaîne).

Il existe une variante à *explode* appelée *strok()* (*ok pour token*) qui contrairement à *explode* qui réalise le découpage en une fois, le réalise ici à chaque itération :

```
$token = strok($mavARIABLE," | ");  
while ($token != ' ') {  
    print $token. "<br>";  
    $token = strok($mavARIABLE," | ");  
}
```

L’ensemble des fonctions “ classiques ” rencontrées en C sur les chaînes se retrouvent également en PHP. On y retrouve ainsi : *strlen*, *strcmp*, *strcascmp*(idem *strcmp*, mais respecte la casse), mais aussi :

- *strpos* (chaîne, sous chaîne) : retourne la position de la sous chaîne dans la chaîne. Dans le cas où la chaîne existe en plusieurs exemplaires, c’est la position de la première occurrence qui est retournée. *strrpos* retourne quand à elle la position de la dernière occurrence.
- *strstr* (chaîne, sous chaîne) retourne la portion de la chaîne à partir de la première occurrence de la sous chaîne.
- *str\_replace* (chaîne à remplacer, nouvelle chaîne, chaîne complète) : Permet de remplacer une sous chaîne par une autre dans une chaîne de caractères. Il est à noter que la chaîne à remplacer peut également être un tableau de chaîne afin de contenir par exemple une liste de termes à remplacer par un seul unique autre.
- *foreach* (nom tableau) : A chaque appel, cette fonction retourne la valeur suivante du tableau.
- *strlen* (chaîne) : retourne la taille de la chaîne.
- *strtolower/strtoupper* (chaîne) : retourne la chaîne passée en paramètres en minuscules (resp. majuscules).

- *ucfirst(chaine)* : permet de mettre en majuscule la première lettre de la chaîne.
- *ucwords(chaine)* : met en majuscule la première lettre de chaque mot.
- *str\_replace (car d'origine, car de destination, chaîne)* : remplace le caractère d'origine par le caractère de destination dans la chaîne.
- *trim(chaine)* : supprime les caractères invisibles (espaces, \n, ...) au début et à la fin de la chaîne. Il existe *ltrim* et *rtrim* qui réalisent la même chose mais uniquement pour les blancs à gauche ou à droite.
- *nl2br(chaine)* : remplace chaque caractère de nouvelle ligne par son code HTML. Sur les versions antérieures à PHP 4.05, c'est <BR> qui est mis, sinon, c'est le code en XHTML <BR /> qui est alors ajouté.
- *ereg(chaine à chercher, chaîne)* : retourne vrai si la chaîne à chercher (sous forme de chaîne ou sous forme d'expression régulière) est contenue dans chaîne. Le format des expressions régulières peut être au format POSIX et Perl (POSIX et celui par défaut).
- *addslashes (chaîne)/stripslashes(chaine)* : ces deux fonctions retournent la chaîne passée en paramètre en ajoutant/retirant des caractères d'échappement devant chaque caractère spécial. Il est possible que la version de PHP le fasse automatiquement si la directive de compilation *magic\_quote\_gpc* est activée (elle l'est sur les versions récentes). Il est possible de le vérifier avec la fonction *magic\_quote\_gpc()*.

Il est fréquent d'avoir à retourner des guillemets vers le navigateur, le problème est qu'il faut ajouter un caractère d'échappement devant chaque :

```
print "ceci sont des "guillemets" "; // syntaxe invalide
print "ceci sont des \"guillemets\" "; // syntaxe valide mais lourde
print "ceci sont des 'guillemets' "; // syntaxe valide
print "ceci sont des "guillemets" ' "; // syntaxe valide
```

## 8.2 Fonctions mathématiques

On retrouve en PHP l'ensemble des fonctions mathématiques que l'on retrouve en C, à savoir : *abs*, *cos*, *sin*, *tan*, *acos*, ..., *exp*, *sqrt*, ... Ces fonctions acceptent un nombre en paramètre et retournent, la valeur absolue, le cosinus, le sinus, la tangente, l'arc cosinus, ..., le carré, la racine carrée, ... Il existe également une fonction *pi()* retournant une valeur approchée de PI.

## 8.3 Gestion de date, heure, temps

Dans ce domaine également, PHP fournit un ensemble de fonctions particulièrement appréciables comme *date(format)*.

Les codes des formats de date sont les suivants :

Code	Description
a	am ou pm
A	AM ou PM
d	Jour du mois avec suppression des 0
D	Jour de la semaine, abrégé en trois lettres
F	Nom du mois
h	Heure, de 1 à 12
H	Heure, de 0 à 24
i	Minutes
j	Jour du mois, avec conservation des 0
l	Jour de la semaine

m	Chiffre du mois
M	Abréviation du nom du mois
S	Suffixe ordinal pour le jour du mois
U	Nombre de secondes depuis le 1/1/1970
y	Année, sur 2 unités
Y	Année, sur 4 unités
z	Jour de l'année

Exemple d'utilisation :

```
< ?
$ladate=date(' l j F Y');
$lheure=date('h : i - a') ;
print "Nous sommes le $ladate <BR>" ;
print "Il est : $lheure <BR>" ;
?>
Affichera :
Nous sommes le Thursday 16 November 2000
Il est : 08:56 - am
```

Il est une autre fonction qui peut avoir son utilité, c'est la fonction *sleep(nombre de secondes)* qui réalise une pause dans l'exécution du script. La fonction *usleep* fait de même, mais l'unité est la milliseconde.

#### 8.4 Fonctions orientées serveur

Il est possible d'exécuter des commandes système directement sur le serveur :

- *passthru(commande)/system(commande)* : permet d'exécuter une commande sur le serveur et de visualiser le résultat sur la sortie standard (exemple : *passthru("ls -l")*; donnera à l'écran la liste des fichiers du répertoire contenant le script où est située cette instruction). *passthru* retourne les résultats à la fin de l'exécution de la commande alors que *system* essaie de les envoyer au fur et à mesure qu'ils sont retournés.
- *exec (commande[, resultat])* : réalise la même chose que précédemment mais sans écho sur la sortie standard. Si le second paramètre est donné, il contiendra un tableau de chaînes avec chaque ligne du résultat.

### 9 Sérialisation

Il est parfois nécessaire de sauvegarder un objet ou le contenu d'un tableau dans son état. Un mécanisme de sérialisation est aussi mis à disposition. Il transforme dans un format de type chaîne de caractères le contenu d'un tableau ou un objet quel qu'il soit.

Pour ce faire, deux fonctions de sérialisation (encodage) et désérialisation (décodage) sont nécessaires :

```
$monobjetserialise = serialize($monobjet); // sérialisation
$monnouvelobjet = unserialize ($monobjetserialise); // désérialisation
```

Attention, au moment de la désérialisation d'un objet, il est nécessaire de connaître la structure de classe (d'où l'intérêt de l'utilisation des fonctions *include/require* !)

L'intérêt est qu'au delà de l'encodage, il est ainsi possible d'envoyer via le réseau des objets/tableaux en PHP, mais également de sauvegarder des objets/tableau aisément dans des fichiers/BD pour les retrouver tels quels plus tard.

## 10 Fonctions orientées réseau

### 10.1 Mail

Il existe une méthode PHP permettant d'envoyer un mail directement, sans appeler un quelconque gestionnaire de courrier.

La fonction *mail* (ou *email* parfois) permet de réaliser cela. Elle nécessite au moins trois paramètres :

- ☐ Le destinataire,
- ☐ L'objet du message,
- ☐ Le corps du message.

```
< ?  
mail('dupont@mondomaine.fr', 'Test de la commande mail', 'Voici le corps du mail') ;  
?>
```

Enverra un mail à dupont@mondomaine.fr avec comme sujet de mail " Test de la commande mail ", et comme corps du mail : " Voici le corps du mail ".

### 10.2 FTP

La connexion à un serveur FTP requiert la même chose qu'une authentification à un SGBD : une URL, un identifiant de connexion, un mot de passe.

- *ftp\_connect(URL)* : retourne un identifiant de connexions à l'URL spécifiée
- *ftp\_login ("roose", "mp")* : retourne un code d'erreur dans le cas d'une authentification erronée,
- *ftp\_quit(identifiant)* : termine la connexion,
- *ftp\_fget(identifiant connexion, identifiant fichier local, nom fichier distant, mode[FTP\_BINARY / FTP\_ASCII])*. Avant de récupérer un fichier, il est nécessaire d'ouvrir un fichier sur le serveur en mode création (w). Celui-ci contiendra le fichier récupéré.
- *ftp\_get(identifiant connexion, fichier local, fichier distant, mode)* : identique au précédent, mais ne demande pas l'ouverture préalable du fichier local.
- *ftp\_fput (identifiant connexion, nom fichier distant, identifiant fichier local, mode)* : permet de transférer un fichier préalablement ouvert en lecture vers le serveur FTP sur lequel on est identifié.
- *ftp\_put (identifiant connexion, nom fichier distant, nom fichier local, mode)* : idem mais sans ouverture du fichier préalable.
- *set\_time\_limit (nb secondes)* : permet de limiter le temps d'exécution du script (du script, et pas du transfert) en cas de problème. Par défaut, si cette fonction n'est pas utilisée, le temps est de 30 secondes.
- *ftp\_size (identifiant, nom du fichier distant)* : retourne la taille du fichier distant, -1 en cas d'erreur. Cette fonction permet entre autre de calculer le temps nécessaire au transfert.
- *ftp\_nlist (identifiant, dirname(nom répertoire))* : permet d'obtenir la liste des fichiers dans un répertoire donné. Cette commande est utilisée par exemple afin d'écrire l'équivalent de la commande *mget*.



```
$liste_fichiers = ftp_nlist($odentifiant, dirname($chemin));  
foreach ($liste_fichiers as $fichier)  
    print "$fichier <br>";
```

## 11 Base de données

PHP offre un nombre de fonctions impressionnantes permettant aux scripts de récupérer des données à partir de quantité de SGBD différents. En natif, PHP gère les drivers pour accéder aux SGBD PostgreSQL, *dBase*, *mSQL*, *mySQL*, *Oracle*, *FilePro*, *Informix*, *Sybase*, *InterBase* ainsi que tous les SGBD supportant les accès via ODBC (*Open DataBase Connectivity*).

Nous n'allons pas décrire l'ensemble de ces fonctions, mais uniquement celles qui sont nécessaires pour une utilisation "classique".

Nous allons nous baser ici sur une connexion à un SGBD distant de type *mySQL* avec une interrogation de la base en SQL.

### 11.1 Ouverture d'une base

Il existe essentiellement deux fonctions permettant l'accès à une base. La première, *mysql\_connect* permet de se connecter au SGBD, la seconde *mysql\_select* permet de sélectionner une base.

Exemple

```
<?  
$bdd= "mabase"; // Base de données  
$host= "sql.iutbayonne.univ-pau.fr";  
$user= "roose"; // Utilisateur  
$pass= "12345";  
  
mysql_connect($host,$user,$pass) or die ("Impossible de se connecter à la base de données");  
mysql_select_db($bdd);  
?>
```

L'exemple précédent va permettre de se connecter sur le SGBD *mySQL* situé à l'adresse *sql.iutbayonne.univ-pau.fr*, avec une identification personnelle.

Bien évidemment, comme à l'accoutumé, lorsqu'on ouvre quelque chose en informatique, on le referme. On n'oubliera pas donc de fermer l'accès à la BD lorsque l'on en aura terminé avec elle, avec *mysql\_close()*.

### 11.2 Requêtes SQL

Une fois la base de données ouverte, il est possible de l'interroger en langage SQL classique. Pour cela, il est nécessaire de lui transmettre la requête à l'aide de la fonction : *mysql\_query* dont voici un exemple d'utilisation :

```
$query = "SELECT num, pays, date, circuit FROM $nomtable "; //$nomtable contient le nom de la table.  
$result= mysql_query($query);
```

Bien qu'ici la requête est mise dans une variable, il est tout à fait possible de l'inclure directement dans la commande *mysql\_query*.

Généralement après chaque requête, on teste si tout c'est bien passé. Pour cela, on appelle la fonction *mysql\_error* qui récupère le dernier message d'erreur retourné par une fonction *mySQL*.

```
if (mysql_error()){ // Erreur base de données, sûrement la table qu'il faut créer
    print "Erreur dans la base de données : ".mysql_error(); // On concatène et on affiche l'erreur
    produite.
    exit();
}
```

### 11.3 Récupération de données

La fonction *mysql\_fetch\_row( )* retourne un tableau qui représente tous les champs d'une rangée de résultat (un tuple). Chaque appel produit le tuple jusqu'à ce qu'il n'y en ait plus.

Il existe d'autres fonctions pour réaliser cela, mais celle-ci est la plus rapide pour obtenir des résultats à partir d'une requête.

```
while ($row=mysql_fetch_row($result)) // $result a été obtenu par le msql_query précédent.
{
    // récupération des informations
    $num = $row[0];
    $pays = $row[1];
    $date = $row[2];
    $circuit = $row[3];

    " ... ";
}
```

A noter qu'au lieu d'utiliser les indices 0, 1, ... il est possible de donner le nom du champ (*\$row['pays']*) à la condition d'utiliser *mysql\_fetch\_array()* en lieu et place de *mysql\_fetch\_row()*.

### 11.4 Autres fonctions

*mysql\_list\_db ( )* retourne un tableau contenant les noms des bases disponibles. C'est le pendant de *SHOW DATABASES* en SQL.

*mysql\_list\_tables( )* réalise la même chose que précédemment, mais pour les tables. C'est le pendant de *SHOW TABLES* en SQL.

*mysql\_num\_rows( )* retourne le nombre de lignes d'un résultat obtenu par une requête.

### 11.5 Utilisation d'ODBC

Afin de communiquer avec divers SGBD, un logiciel client utilise une API (*Application Programming Interface*) appelée ODBC. Ces API sont généralement écrites par les éditeurs de SGBD eux mêmes. Il est ainsi possible à partir d'un même programme d'accéder à différents SGBD à partir du moment où l'on possède les drivers de son API.

ODBC utilise le langage SQL supporté par la quasi totalité des SGBD, et particulièrement, les SGBD relationnels.

Nous n'allons pas décrire l'utilisation des fonctions ODBC puisqu'elles s'utilisent de manière identique à celles décrites précédemment pour *mySQL*.

## 12 Manipulations d'images

Grâce à la librairie GD intégrée à PHP, il existe toute une série de fonctions permettant de manipuler, entre autres, des images au format JPEG (*Joine Photographic Experts Group*), PNG (*Portable Network Graphics*) et WBMP (*Wireless BMP*). Elles permettent par exemple de retourner directement une image créée au cours du script PHP.

Le choix du format se fait en fonction des images à générer. Le format JPEG est plutôt utilisé pour des images riches en couleurs et/ou en dégradés (comme les photos). C'est un codage avec compression et avec pertes. Le PNG est en train de remplacer le GIF (intègre le codage Lempel Ziv Welch protégé par un brevet via la société UNISYS – c'est donc un format propriétaire ET payant). Ce format permet une compression sans perte et s'utilise particulièrement pour des images contenant du texte, des tracés et des couleurs unies (boutons de sites web, bannières, etc.). Ce format permet la transparence et l'entrelacement que ne permet pas le JPEG. Les adeptes du GIF animé seront déçus, le PNG ne le permet pas. Un format appelé MNG qui le permettra est en cours de développement. Enfin, le WBMP est spécifique aux périphériques sans fils. Il n'est pas encore généralisé.

Nous n'allons pas décrire ici toutes les (très) nombreuses fonctions qui nous demanderaient plus de temps que nous n'en n'avons mais nous allons décrire les principales qui nous serviront d'illustration afin de comprendre les mécanismes.

### 12.1 Séquence de génération d'une image

La génération d'une image se réalise en 4 étapes incontournables :

1. Création/Lecture de l'image (dimensions) qui servira de fond,
2. Ajout de dessins, formes, textes, actions sur ses paramètres (*taille, correction gamma, etc.*)
3. Génération de l'image – retour vers le navigateur (ou un fichier),
4. Suppression de l'image sur le serveur

Afin que votre navigateur sache que ce sont des images qu'il va recevoir, il est nécessaire que la première information qui lui est communiquée soit le type de l'image.

```
<?
header(" Content-type: image/jpeg ") ; // ATTENTION, respecter scrupuleusement la syntaxe
...
?>
```

### 12.2 Création d'une image

Une fois que le type de l'image à retourner au navigateur est défini, il faut indiquer la taille de l'image en pixel par la fonction :

- *ImageCreate(largeur, hauteur)*. Celle-ci retourne l'identifiant d'une image. Cet identifiant sera utilisé par la plupart des fonctions graphiques. On pensera impérativement à libérer l'espace

mémoire une fois que l'image créée est terminée (*ImageDestroy(identifiant image)* ). La destruction d'une image n'implique pas sa disparition à l'écran puisqu'elle aura au préalable été retournée au navigateur.

- *ImageCreateTrueColor(largeur, hauteur)* : idem précédent, mais en vrai couleurs et pas sur une palette
- *ImageCreateFrom[PNG|JPEG|GIF](« nom de l'image »)* : permet de créer une image à partir d'une autre déjà existante (la création à partir GIF est autorisée – l'image retournée au navigateur ne sera pas un GIF).

### 12.3 Gestion des couleurs

Maintenant que nous connaissons à la fois le format et la taille de notre image, on peut choisir une couleur :

- *ImageColorAllocate(image, rouge, vert, bleu)*.

Cette fonction alloue une couleur à l'image spécifiée. Chaque composant de couleur peut prendre une valeur entre 0 et 255. L'identifiant retourné servira dans d'autres fonctions pour faire référence à cette couleur.

- *ImagegammaCorrect(image, gamma\_in, gamma\_out)*;

Cette fonction permet de jouer sur la luminosité d'une image. Le taux se calcule par un ration entre la gamma\_in et le gamma\_out. Ainsi, si gamma\_in = 1 et gamma\_out = 2, l'image sera 2 fois plus claire. Un gamma\_in inférieur à gamma\_out l'assombrira.

### 12.4 Gestion des formes

- *ImageFill* (image, x début, y début, couleur) permet de remplir un espace vide par une couleur. Utilisée dès le départ, elle permet de donner un fond à une image. Attention, les coordonnées (0,0) correspondent au coin supérieur gauche.
- *Image[Dashed]Line*(image, x début, y début, x fin, y fin, couleur) trace une ligne (en pointillé pour *ImageDashedLine*) en partant des coordonnées (x début, y début) jusqu'à (x fin, y fin).
- *Image[Filled]Polygon*(image, tableau de points, nombre de points, couleur) autorise le dessin de polygone plein (*ImageFilledPolygon*) ou juste le contour (*ImagePolygon*). Les coordonnées sont contenues dans un tableau de points (2 à 2), le paramètre nombre de points indique combien de points sont à récupérer dans le tableau.
- 

```
< ?
imagefilledpolygon($image, array(100,10,50,60,150,60), 3 $color) ; // trace un rectangle
?>
```

- *Image[Filled]Rectangle*(\$image, x haut gauche, y haut gauche, x bas droite, y bas droite, couleur) trace un rectangle plein (*ImageFilledRectangle*) ou son contour (*ImageRectangle*),

Exemple : `imagefill ($monimage, 0, 0, $macouleur) ;`

- Plus simple, mais peut être plus utile (tracé de courbes par exemple), la fonction *imagesetpixel*(image, x, y, couleur) permet de placer un pixel dans la zone image.

## 12.5 Gestion des caractères/chaînes de caractères

Pour dessiner un caractère/chaîne de caractères dans une image :

- *ImageChar(\$image, police de car., x, y, " car ", couleur)*. Le paramètre police correspond au numéro (de 1 à 5) d'une des 5 polices de caractères disponibles en standard dans PHP. Les coordonnées (x,y) réfèrent au coin supérieur gauche de la lettre.
- *ImageCharUp* fait de même mais avec des caractères orientés vers le haut.
- *ImageString* et *ImageStringUp* sont identiques à la précédente mais permettent le dessin de chaînes de caractères.
- *ImageLoadFont(nom de la police de caractère)* permet de charger de nouvelles polices de caractères. Ces polices peuvent ensuite être utilisées par exemple avec la fonction *ImageString* déjà vue précédemment.

```
< ?  
$mapolice=imageloadfont(" helvetica ") ;  
imagestring($image, $mapolice, 10,10, " Coucou ", $couleur ) ;  
?>
```

## 12.6 Gestion des couleurs

- *ImageColorAt(image, x, y)* retourne l'index de la couleur spécifiée aux coordonnées x,y.
- *ImageColorsTotal(\$image)* retourne le nombre de couleurs dans l'image spécifiée.
- *ImageColorTransparent(entier image, entier couleur)* rend transparente la couleur spécifiée (ne fonctionne qu'avec le PNG).

```
< ?  
imagefillrectangle($monimage, 30, 30, 70, 70, $macouleur) ; // dessine un rectangle dans une certaine  
couleur  
imagecolortransparent($monimage, $macouleur) ; // rend cette couleur transparente.  
?>
```

- *GetImageSize (nom de fichier)* : cette fonction est disponible avec toutes les versions de php (ce qui n'est pas le cas avec les fonctions qui suivront). Le nom de fichier doit correspondre à l'un des formats suivants : GIF, JPEG, PNG. Elle retourne un tableau de 4 éléments : [0] : largeur en pixels, [1] : hauteur en pixels, [2] : type d'image (GIF = 1, JPEG = 2, PNG = 3), [3] : chaîne au format 'HTML' du type " HEIGHT=### WIDTH=### " utilisable dans une balise IMG.
- Les fonctions *imageSX(image)* et *imageSY(image)* permettent de connaître la largeur et la hauteur d'une image (souvent nécessaire pour connaître la taille d'une image chargée dynamiquement, par exemple avec *ImageCreateFromPng*).

## 12.7 Manipulation d'images

Pour aller plus loin avec la gestion des images, il faut savoir qu'il existe toute une panoplie de fonctions très utiles comme *ImageCopyResampled* qui permet de redimensionner des images (utilise pour générer

des vignettes par exemple) ou *imagecopy* qui permet de copier une zone d'une image dans une autre. La liste exhaustive est sur <http://fr2.php.net/manual/fr/ref.image.php>.

- *imagecopy*(\$image\_dest, \$image\_src, \$dest\_x, \$dest\_y, \$src\_x, \$src\_y, \$src\_largeur, \$src\_hauteur);

Copie une zone d'*image\_src* dans *image\_dest*. La zone démarre en *src\_x* et *src\_y*, sur *src\_largeur* et *src\_hauteur* pixels dans *image\_dest* à partir de *dest\_x*, *dest\_y*.

- *imagecopymerge*(\$image\_dest, \$image\_src, \$dest\_x, \$dest\_y, \$src\_x, \$src\_y, \$src\_largeur, \$src\_hauteur, \$opacite);

Réalise la fusion (merge) de deux images tout en permettant une gestion de l'opacité – entier entre 0 et 100. Pour le reste, le fonctionnement est identique à la fonction *imagecopy*.

- *imagecopyresampled* (\$image\_dest, \$image\_src, \$dest\_x, \$dest\_y, \$src\_x, \$src\_y, \$dest\_largeur, \$dest\_hauteur, \$src\_largeur, \$src\_hauteur);

Permet de redimensionner tout ou partie rectangulaire d'*image\_src* dans *image\_dest*. Ça démarre en *src\_x*, *src\_y* vers *dest\_x*, *dest\_y* sur *src\_largeur*, *src\_hauteur* vers *dest\_largeur*, *dest\_hauteur*. Si l'on souhaite redimensionner une image entière *dest\_x* = *dest\_y* = *src\_x* = *src\_y* = 0. Si l'on ne veut pas étirer l'image, il faut que le ratio  $\frac{dest\_x}{src\_x} = \frac{dest\_y}{src\_y}$ . Il vaut mieux privilégier cette fonction à *ImageCopyResized* qui s'utilise de façon analogue mais donne des résultats moins jolis et des effets d'escalier.

```
function redimage($img_src,$img_dest,$dst_w,$dst_h) {
    // Lit les dimensions de l'image
    $size = GetImageSize("$img_src");
    $src_w = $size[0]; $src_h = $size[1];

    // Teste les dimensions tenant dans la zone
    $test_h = round(($dst_w / $src_w) * $src_h);
    $test_w = round(($dst_h / $src_h) * $src_w);

    // Crée une image vierge aux bonnes dimensions
    // $dst_im = ImageCreate($dst_w,$dst_h);
    $dst_im = ImageCreateTrueColor($dst_w,$dst_h);
    // Copie dedans l'image initiale redimensionnée
    $src_im = ImageCreateFromJpeg("$img_src");
    //ImageCopyResized($dst_im,$src_im,0,0,0,0,$dst_w,$dst_h,$src_w,$src_h);
    ImageCopyResampled($dst_im,$src_im,0,0,0,0,$dst_w,$dst_h,$src_w,$src_h);

    // Sauve la nouvelle image
    ImageJpeg($dst_im,"$img_dest");

    // Détruis les tampons
    ImageDestroy($dst_im);
    ImageDestroy($src_im);
}
```

## 12.8 Récupération des images

Enfin, une fois les dessins réalisés, il est nécessaire de retourner au navigateur l'image ainsi générée dans le format défini :

- `image[jpg/png/bmp](id image)` : retourne l'image vers le navigateur.

Il est également possible d'enregistrer l'image dans un fichier (utile par exemple pour la génération de vignettes)

```
<?
$fichier = 'images/vignettes/image.png';
imagepng($image, $fichier);
?>
```

Une fois l'image retournée vers le navigateur, celle-ci est devenue inutile sur le serveur. Aussi, afin de libérer ce dernier (n'oublions pas que nous sommes dans un environnement distribué et partagé et donc...nous ne sommes pas seuls !), on appliquera la fonction :

- `imageDestroy(identifiant image)` : supprime l'image du serveur...et pas dans le navigateur !

## 13 Programmation objet : PHP et les classes (PHP 4/PHP 5)

Avec PHP, on programme comme on veut...et même en objet si on le désire. Attention, PHP n'étant pas à la base un langage de programmation objet, il ne supporte pas tous les concepts inhérents à ce style/philosophie de programmation. Néanmoins, l'utilisation de classes bien définies permettra aux utilisateurs d'exploiter au mieux les fonctionnalités déjà développées.

### 13.1 Syntaxe de la programmation objet

La syntaxe est fortement inspirée du C++.

#### 13.1.1 Allocation d'un objet

Tout se passe comme une allocation classique en C++, la seule différence est que l'on retrouve l'éternel \$ propre au PHP et langage de scripts en général.

```
$monObjet = new maClasse(paramètres éventuels du constructeur) ;
```

#### 13.1.2 Accès aux méthodes et attributs

```
$monObjet->nomMethode(...) ;
$monObjet->nomAttribut = ... ou $mavar = $monObjet->nomAttribut
```

On fera attention à mettre le \$ sur le nom de l'objet et pas sur le nom de la méthode/attribut. Lorsqu'on désirera affecter un attribut relatif à soi-même (le *this*), on procédera de la même manière : *\$this->...*  
PHP n'ayant pas de déclaration préalable des variables nécessaires, l'oubli de *\$this->* ne provoquera pas d'erreur, une nouvelle variable sera ainsi définie et utilisée, mais ce ne sera pas l'attribut de la classe.

#### 13.1.3 Définition d'une méthode

```
function maMethode([$param1, $param2) {
```

```
    return $... // permet à la méthode de faire un retour de " ce que l'on veut "  
}
```

#### 13.1.4 Déclaration d'une classe

```
class MaClasse {  
    $attribut1 ;  
    $attribut2 ;  
  
    function maMethode1( ) {  
        ...  
    }  
    function maMethode2($param1, $param2= " je suis une chaîne ") { // paramètre par défaut...  
        ...  
    }  
} // fin de la déclaration de la classe.
```

Par défaut, tout accès à un membre (attribut/méthode) est public. Néanmoins, il est tout à fait possible d'y préciser un accès de type *private* ou *protected* en plaçant le mot clé devant la déclaration de l'attribut et/ou de la méthode.

Communément, les fichiers de déclaration comporteront l'extension *.inc* (pour *include*). Afin de ne pas surcharger le code, et dans un but de réutilisation maximum, nous inclurons le code via la commande *include(" maclasse.inc ")* dans le fichier PHP exploitant cette dernière. On procédera ainsi :

```
< ?php  
    include (" maClasse.inc ") ;  
?>  
  
<HTML> <HEAD> ...</HEAD> <BODY>  
< ?php  
$monObjet = new maClasse(...);  
?>  
</BODY></HTML>
```

#### 13.1.5 Autres concepts

L'héritage et oui, et c'est souhaitable, il est également possible de profiter des avantages de l'héritage, mais contrairement au C++ ou SmallTalk, PHP utilise la philosophie chère à Java en n'autorisant que l'héritage simple (pas d'héritage multiple) :

```
class maClasseHéritée extends maClasse {  
    ...  
    parent::methode1() // permet d'appeler la méthode "originale" de la classe mère  
}
```



La surdéfinition d'attributs et de méthodes est également possible à la condition que le membre concerné n'ait pas été déclaré *final* (*final* méthode (...)),

Afin de palier en partie aux problèmes liés à l'inexistence d'héritage multiple, PHP permet de définir des *interfaces*. Une Interface permet de spécifier le squelette d'une classe qui sera implémentée dans les classes implémentant l'interface :

```
interface AffichezMoi {  
    function afficher();  
}  
  
class maClasseAMoi implements AffichezMoi {  
    function afficher() {  
        //implémentation de la méthode de l'interface  
    }  
    ...// Attention, il ne faut pas implémenter uniquement les méthodes de l'interface.  
}
```

Nous avons ici une classe qui implémente (ce qui n'est pas exclusif avec l'héritage) une classe dont elle contiendra le code. Ceci permet de créer des modèles de classes en délocalisant l'implémentation à la classe qui la nécessite afin que l'implémentation « colle » au mieux aux besoins.

Lors de l'utilisation de classes, il est à noter une fonction bien pratique (également présente en Java par exemple) qui permet de connaître la classe d'un objet et donc de réaliser des traitements particuliers :

```
if ($o1 instanceof ClasseA)  
...  
else  
...
```

Également présente la possibilité de créer une copie d'un objet à l'aide du mot clé *clone* :

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

Enfin, il est également possible d'utiliser la réflexivité sur les classes afin d'interroger les classes et objets pour en connaître leur structure et contenu. Cette API complète ne sera pas décrite, on pourra consulter la bibliographie pour plus d'informations.

## 14 Gestion des Exceptions (PHP 5)

Les programmeurs Java sont familiers avec ce concept. L'idée est d'encapsuler du “code à problème” dans une section. Si ce code se déroule mal, une exception est levée et la procédure associée sera exécutée :

```
try {  
    // code à problème  
}
```

```
catch (identifiant de l'exception) {  
    // procédure associée  
}
```

Il est possible d'associer plusieurs procédures (plusieurs *catch*) à un *try* dans le cas où le code critique peut lever plusieurs exceptions différentes.

Chaque exception étant un objet, un certain nombre de méthodes ont été mise dans la définition de la classe afin de “pister” l'exception :

- *getMessage()* : retourne la chaîne de caractère définie dans le constructeur,
- *getCode()* : donne le code de l'exception passé au constructeur,
- *getFile()* : donne le nom (avec chemin complet) du fichier dans lequel s'est produite l'exception,
- *getLine()* : retourne le numéro de la ligne où s'est produite l'exception,
- *getTrace()/getTraceAsString()* : permet d'avoir une trace complète, une sorte d'historique de ce qui s'est passé. *GetTrace* retourne un tableau, *getTraceAsString*, idem mais sous forme de chaînes de caractères.

```
try {  
    ...  
    throw new Exception ('Aïe, y a un problème', 1);  
}  
catch (Exception $e) {  
    print "<br>Exception n°: ".$e->getCode();  
    print "<br>Message : ".$e->getMessage();  
    print "<br>Dans le fichier : ".$e->getFile();  
    print "<br>A la ligne : ".$e->getLine();  
}
```

Afin de définir ses propres exceptions, il est tout à fait possible de se créer une classe héritée d'*Exception* que l'on lèvera par la suite à l'aide de *throw* et qu'on récupère à l'aide de *catch*.

## 15 Contrôle de sessions

HTTP n'est pas un protocole maintenant un état entre de transactions. Ceci signifie entre autre qu'il n'est pas possible de savoir que deux requêtes qui se suivent ont été réalisées par le même utilisateur.

Afin de palier à ce problème, un certain nombre de mécanismes ont été mis en oeuvre afin de suivre un utilisateur tout au long de sa session. Mais qu'est ce qu'une session ? C'est une instance d'un navigateur. Tant que la fenêtre du navigateur n'a pas été fermée (et que nous sommes dans un délai imparti), nous sommes dans la même session.

Lors d'une création d'une session, un identifiant aléatoire est généré automatiquement : c'est l'identifiant de session. Il peut être stocké dans un *cookie* (*petit fichier texte stocké sur le poste « client »*), envoyé dans l'URL (GET) ou par l'environnement (POST).

### 15.1 Création de cookies

- *setcookie* (*nom, valeur, expiration, chemin, domaine, securite*) permet de créer un cookie sur le poste client. Le premier champ *nom* est obligatoire et définit son nom. Si l'on souhaite lui associer une valeur on renseignera le second champ. Le troisième permet de spécifier une date

d'expiration dans le cas où rien n'est précisé, le cookie devient permanent. Les champs chemin et domaine permettent de préciser les URL et domaines auxquels sont associés le cookie. Enfin le dernier, sécurité permet de n'envoyer le cookie que si la connexion est réalisée via le protocole sécurisé HTTPS (TRUE).

- Une fois le cookie créé, il est possible de le récupérer dans les pages suivantes via le tableau `$_COOKIE['nom cookie']`.

```
setcookie("TestCookie", $value, time()+3600); // expire dans une heure
```

- La suppression d'un cookie se fait en le recréant...mais avec une date d'expiration passée !

```
setcookie ("TestCookie", "", time() - 3600); // a expiré y'a une heure
```

Les cookies ne sont malheureusement pas supportés par l'ensemble des navigateurs pour des raisons de confidentialité. Aussi, il est nécessaire de mettre en œuvre d'autres mécanismes afin de gérer les sessions.

## 15.2 Création de variables de sessions

La gestion par variables de sessions évite de passer par des variables partagées et/ou des cookies. Les variables de session sont stockées dans un SGBD et/ou dans un fichier ASCII, mais contrairement aux cookies, elles sont stockées sur le serveur. Seul l'identifiant de la connexion est stocké sur le poste du client. Les fonctions associées sont les suivantes :

- *bool session\_start()* . Dans le cas où true est retournée, cette fonction permet soit de créer une session (si pas encore définie) soit d'utiliser les variables de la session en cours.
- *string Session\_ID ([string id])* permet d'accéder à l'identifiant unique d'une session. Dans le cas où un paramètre est précisé, l'identifiant de session changera et prendra la valeur fournie
- *string Session\_Name([string name])* retourne le nom de la session en cours ou le change avec la valeur du paramètre précisé.
- L'affectation du tableau `$_SESSION` permet de créer des variables de session (il faut que *session\_start()* soit en entête de page).

```
$_SESSION['mavARIABLE'] = valeur ;
```

Par la suite, l'accès à la variable de session se fera également via ce tableau.

```
if (isset($_SESSION['mavARIABLE']))
    $_SESSION['mavARIABLE'] = valeur ;
else
    print $_SESSION['mavARIABLE'];
```

A la fin de la session, il est utile de désenregistrer la variable en utilisant

- `unset($_SESSION['mavARIABLE']);`

Si l'on souhaite désenregistrer l'ensemble des variables de session, il faut utiliser :

- `$_SESSION=array();`

Enfin, la fonction *session\_destroy()* supprime l'identifiant de la session.

## 16 PHP-Ajax

Jusqu'à présent, les scripts que nous avons réalisés retournent l'intégralité d'une page web. Impossible de ne retourner qu'une partie (c'est que l'on appelle le Web 1.0). Ainsi, lors d'interactions soutenues avec un utilisateur, des lenteurs peuvent apparaître du fait de la lourdeur des échanges, ce qui nuit rend pénible l'usage de certaines pages. La technologie AJAX (qui n'est pas nouvelle) a pour objectif de permettre un rafraichissement partiel de données d'une page web, et uniquement de certaines données (c'est que l'on appelle le Web 1.0). AJAX est l'acronyme de « Asynchronous JavaScript and XML », bien qu'il ne soit en rien lié au XML. AJAX est basé sur l'objet *XMLHttpRequest* qui permet de faire une requête via Javascript à un serveur http (jusque là rien de nouveau !) et d'attendre le retour en ne rafraichissant que certaines données contenues dans le code HTML retourné initialement

Même si dans l'acronyme d'AJAX il est mentionné le mot asynchrone, il n'est pas nécessaire que ce le soit (rappel : un appel synchrone => on attend la réponse du serveur pour continuer/terminer ; asynchrone, => on n'attend pas !). Le choix entre synchrone et asynchrone se fait lors de l'instanciation de l'objet *XMLHttpRequest* avec dans le dernier paramètre *true* pour asynchrone, *false* pour synchrone.

La contrepartie à l'utilisation de la méthode asynchrone est qu'il n'est pas possible de prédire le moment où le serveur va répondre. Lorsque l'objet *XMLHttpRequest* change d'état, il lève un événement (*onreadystatechange*) que l'on va associer à une fonction. Cet événement est levé dès que l'objet *readyState* est modifié. Il peut prendre les valeurs : 0 non initialisée, 1 en chargement, 2 chargée, 3 en cours de traitement, 4 terminée.

Comme précédemment dit, le mode asynchrone permet de ne pas bloquer le navigateur client pendant le chargement de la page. L'exemple qui suit permet d'afficher un message d'attente durant un traitement quelconque, puis l'affichage du résultat de ce traitement. Cela va se faire en deux temps : d'une part afficher un message lors de l'appel initial, puis le retirer lorsque notre *onreadystatechange* passe à 4 (terminé).

On va utiliser la balise HTML DIV qui permet de diviser le document en section. C'est justement une de ces sections que va mettre à jour notre script.

Page PHP qui réalise une requête...

```
<?php
header('Content-Type: text/xml');

//on connect
$dbhost="iparla.iutbayonne.univ-pau.fr";
$dbuser="roose";
$dbpass="drop64";
$dbdb="roose";

$dblink=mysql_connect($dbhost,$dbuser,$dbpass);
mysql_select_db($dbdb);

//on lance la requete
$query = "SELECT * FROM bourse";
$result = mysql_query($query,$dblink) or die (mysql_error($dblink));

sleep(2);
```

```

//On boucle sur le resultat
echo "<?xml version=\"1.0\"?>\n";
echo "<exemple>\n";

while ($row = mysql_fetch_array($result))
{
    echo "<donnee> $row[0] </donnee>\n";
}
echo "</exemple>\n";

?>

```

### Code « Ajax »

```

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="fr" lang="fr">
<head>
    <title>Exemple d'attente</title>
<script type="text/javascript">

function ajax()
{
    var xhr=null;

    if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
    }
    //on définit l'appel de la fonction au retour serveur
    xhr.onreadystatechange = function() { alert_ajax(xhr); };

    //on affiche le message d'accueil
    //document.getElementById("message").className="msg";
    document.getElementById('zonetraitemnt').innerHTML = "Recherche des villes";
    //on appelle le fichier php
    xhr.open("GET", "http://iparla.iutbayonne.univ-pau.fr/~roose/ajax/attente.php", true);

```

```

    xhr.send(null);
}

function alert_ajax(xhr)
{
    if (xhr.readyState==4)
    {
        var docXML= xhr.responseXML;
        var items = docXML.getElementsByTagName("donnee")

        //une boucle sur chaque element "donnee" trouvé
        var Table = '<table border="1">';
        for (i=0;i<items.length;i++) {
            Table += '<tr>';
            Table += '<td>' + items.item(i).firstChild.data + '</td>';
        }
        Table += '</tr>';
        Table += '</table>';

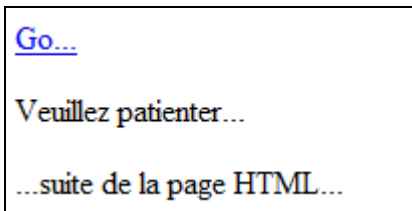
        document.getElementById("zonetraitement").innerHTML=Table;

    }
}
</script>
</head>
<body>
<p>
<a href="javascript:ajax();">Go...</a>
</p>

<div id="zonetraitement">Veuillez patienter...</div>
<p>
...suite de la page HTML...
</p>
</body>
</html>

```

Ce qui donne :



Puis une fois un clic sur « go » et l'attente de la requête...



Si dans le cas présent, la communication (asynchrone) se déclenche sur le « clic » sur « Go... », il est également possible réaliser des interactions plus intéressantes comme une aide à la saisie par exemple :

BDAjax.php

```
<?php
$hote = 'localhost';
$base = 'roose';
$user = 'roose';
$pass = 'drop64';
$cnx = mysql_connect ($hote, $user, $pass) or die (mysql_error ());
$ret = mysql_select_db ($base) or die (mysql_error ());

/* Vérification */
$qer = mysql_query("select ville from bourse where ville='".$$_GET["ville"]."");
if(mysql_num_rows($qer)>=1)
echo "occupe";
else
echo "libre";
?>
```

Formulaire.html

```

<html> <head>
<title>EssayeAjax</title>

<script type="text/javascript">

function writediv(texte)
{
document.getElementById('zonetraitement').innerHTML = texte;
}

function verificationville(ville)
{
if(window.ActiveXObject) // IE
    xhr_object = new ActiveXObject("Microsoft.XMLHTTP");
else
    if(window.XMLHttpRequest) // FIREFOX
        xhr_object = new XMLHttpRequest();

fichier = "http://iparla.iutbayonne.univ-pau.fr/~roose/ajax/BDAjax.php?ville="+ville;

xhr_object.open("GET", fichier, false);
xhr_object.send(null);

if(xhr_object.readyState == 4) {
    texte = xhr_object.responseText;
}

if(texte != "")
{
    if(texte == "occupe"){
        writediv(ville+' : est un Nom de ville est occupé; !');
    }
    else if(texte == "libre") {
        writediv(ville+' : est un nom de ville libre vous pouvez l\'ajouter a la BD');
    }
}

    else
        writediv(texte);
}
}

```



```

</script>
</head>
<body>
<form name="formville" action="" methode="GET">
<input name="ville" type="text" onKeyUp="verificationville(this.value)" >
<!-- onKeyUp : c est un evenement lance la fonction js 'verificationville'
this.value : ce qui est tapé ds la zone de texte, ici alias de ville (nom d'onglet)
-->
<div id="zonetraitemnt"></div>
</form>

</body>
</html>

```

## 17 Webservices & SOAP

Un service web, ou webservice – en anglais ça fait plus chic - voire même SOA (Service Oriented Architecture – ça fait plus pro) permet à des applications de conceptions et de réalisations différentes de communiquer entre elles, et qui plus est, sans avoir à se soucier de l'implémentation. Ce couplage entre applications, appelé 'faible', permet ainsi de réaliser de « nouvelles' applications par assemblages et/ou appels de services fournis par d'autres.

Il existe plusieurs méthodes de communication, nous retiendrons ici uniquement le protocole SOAP, issu du RPC (*Remote Procedure Call*) puis du XML-RPC (le même mais avec des appels en XML pour masquer encore plus l'hétérogénéité des implémentations). Le protocole SOAP (*Simple Object Access Protocol*) permet à des objets d'en appeler d'autres distants (comme avec Java/RMI par exemple) mais en utilisant le protocole HTTP comme protocole de communication (il est possible de faire entre autre également du SMTP) sans se soucier de leur implémentation, et avec une uniformité de représentation des données. Il autorise ainsi la communication et l'échange de messages/données entre objets distants.

Le mécanisme est le suivant :

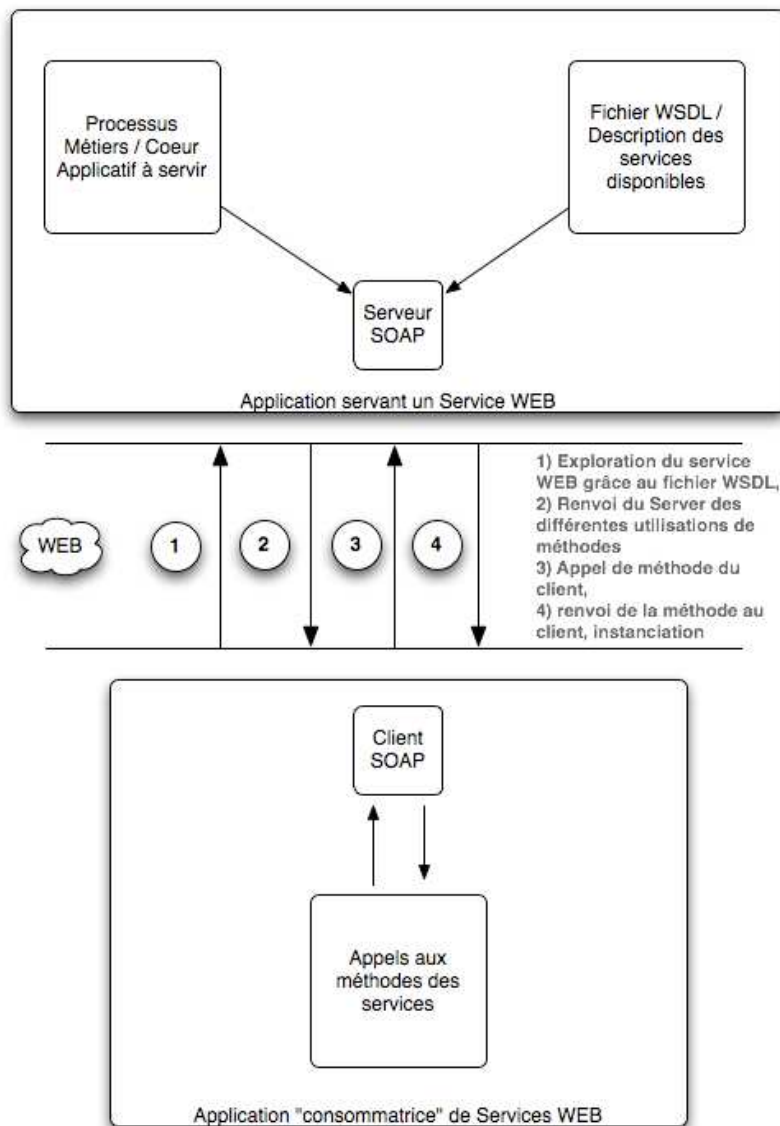


Figure 1: <http://vivien-brissat.developpez.com/tutoriels/php/soap/#LI>

Lors d'un échange SOAP, le message transmis se décompose en 2 parties :

1. L'enveloppe, qui contient toutes les informations relatives au contenu du message ;
2. Le contenu lui-même qui est formé de données structurées (méthodes formatées, interrogation structurée d'après les besoins du serveur, etc.).

Afin de savoir comment utiliser un service web, ce dernier est décrit selon dans un langage de description de service web basé sur XML : le WSDL (*Web Services Description Language*). Il donne une définition abstraite des services, le détail des types de données échangées, les opérations possibles, le protocole à

utiliser ainsi que l'adresse (URL) du service. Le fichier WSDL de chaque webservice peut être publié dans un annuaire. Le fichier WSDL associé à un webservice représente en quelque sorte sa notice d'utilisation.

Dans sa version 5, le PHP intègre en natif une gestion du protocole SOAP.

### *Fichier wsdl*

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- WSDL file generated by Zend Studio. -->
<definitions name="exemple" targetNamespace="urn:exemple" xmlns:typens="urn:exemple"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns="http://schemas.xmlsoap.org/wsdl/" xmlns:typens0="http://iparla.iutbayonne.univ-
pau.fr/~roose/wsdl/moteur.php">
  <message name="retourDate"/>
  <message name="retourDateResponse">
    <part name="retourDateReturn"/>
  </message>
  <portType name="essai_instancePortType">
    <operation name="retourDate">
      <input message="typens:retourDate"/>
      <output message="typens:retourDateResponse"/>
    </operation>
  </portType>
  <binding name="essai_instanceBinding" type="typens:essai_instancePortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="retourDate">
      <soap:operation soapAction="urn:essai_instanceAction"/>
      <input>
        <soap:body namespace="urn:exemple" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </input>
      <output>
        <soap:body namespace="urn:exemple" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/encoding"/>
      </output>
    </operation>
  </binding>
</service name="exempleService">
```

```

        <port name="essai_instancePort" binding="typens:essai_instanceBinding">
            <soap:address location="http://iparla.iutbayonne.univ-
pau.fr/~roose/wsd/moteur.php"/>
        </port>
    </service>
</definitions>

```

#### Moteur.php (serveur)

```

<?php
class DateServer{

//On déclare notre méthode qui renverra la date et la signature du serveur dans un tableau associatif...
function retourDate(){
    $tab = array(
        'serveur' => $_SERVER['SERVER_SIGNATURE'],
        'date' => date("d/m/Y"),
        'auteur' => "service web appele" //attention, unaccent => erreur
    );
    return $tab;
}
}

//Cette option du fichier php.ini permet de ne pas stocker en cache le fichier WSDL, afin de pouvoir faire
nos tests
//Car le cache se renouvelle toutes les 24 heures, ce qui n'est pas idéal pour le développement
ini_set('soap.wsdl_cache_enabled', 0);

//Instanciation du SoapServer
$serversoap=new SoapServer("http://iparla.iutbayonne.univ-pau.fr/~roose/wsd/exemple.wsdl");

// on peut aussi déclarer plus simplement des fonctions
//par l'instruction addFunction() : $serversoap->addFunction("retourDate"); à ce moment-là nous ne
faisons pas de classe.

//Noter le style employé pour la déclaration : le nom de la classe est passé en argument de type String,
et non pas de variable...
$serversoap->setClass("DateServer");

//Ici, on dit très simplement que maintenant c'est à PHP de prendre la main pour servir le Service WEB :
il s'occupera de l'encodage XML, des

```

```
//Enveloppes SOAP, de gérer les demandes clientes, etc. Bref, on en a fini avec le serveur SOAP !!!!
$serversoap->handle();
?>
```

#### Client.php

```
<?php
//Cette option permet d'éviter la mise en cache du WSDL, qui se renouvelle toutes les 24 heures... Pour
le développement, ce n'est pas génial !!!
ini_set('soap.wsdl_cache_enabled', 0);

//On doit passer le fichier WSDL du Service en paramètre de l'objet SoapClient
$service=new SoapClient("http://iparla.iutbayonne.univ-pau.fr/~roose/wsdl/exemple.wsdl");

//On accède à la méthode de notre classe DateServeur, déclaré dans notre SoapServer
$taballservices=$service->retourDate();

//On renvoie le résultat de notre méthode, pour voir...
print_r($taballservices); // affiche les éléments d'un tableau
?>
```

## 18 PHP en chiffre

PHP n'est pas uniquement ce qui a été décrit précédemment. C'est également un poids économique important. PHP, c'est plus de 5 millions de développeurs estimés à travers le monde, c'est un marché de 4,8 milliards d'€.

Selon (<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>), PHP est le 4<sup>ème</sup> (5<sup>ème</sup> selon les périodes – au coude à coude avec C#) langage le plus utilisé, derrière le Java, C, C++.







Le langage (en fait la plate-forme) est utilisé par plus de 1/3 des sites web dans le monde (46 % en France), ce qui représente 17 millions de domaines. Parmi les entreprises du CAC40, 78% en 2003 et 87% en 2004 utilisent le PHP. Parmi les vingt premières sociétés les plus bénéficiaires en France, 95% recourent à PHP pour leur site web.

A quoi est dû cet engouement. Au delà de la simplicité et de la souplesse du langage, PHP s'ouvre sur une facette méconnue de l'interopérabilité. Aussi, PHP peut instancier des objets COM, .NET mais également Java. PHP peut se connecter à l'ensemble des ténors des SGBD, mais aussi aux annuaires (LDAP), à Lotus Notes, à SAP, permet l'utilisation des services web (SOAP/WSDL), gère le XML, s'interface avec les systèmes de paiement en ligne (*VeriSign, Cybercash, Crédit Mutuel, etc.*), génère du Flash (extension Ming), du PDF (classe FPDF). Un autre atout à ne pas négliger, est la robustesse (performances/fiabilités), ce qui en fait le langage privilégié de 90% des sites web les plus fréquentés dans l'hexagone avec des sites dépassant les 500 000 connexions/jour. Une des plus fortes plates-formes au monde supportant PHP permet 150 000 utilisateurs simultanément (avec 220 serveurs en cluster).

Source : Livre Blanc "PHP en entreprise" - <http://www.afup.org/docs/livre-blanc-php-en-entreprise-v4.pdf>

## 19 Conclusion

PHP (*PHP : Hypertext Preprocessor*) est un langage de programmation (comme Java ou C#) possédant deux syntaxes. La première, à mi-chemin entre C et Perl, s'adresse aux développeurs à la recherche d'un langage de script simple à manipuler. Elle est adaptée à la couche présentation. La seconde, très proche de Java, permet de développer dans un paradigme orienté objet. Elle est adaptée au développement de logique métier ou de traitements complexes. PHP permet de développer les types d'applications suivantes :

-  des applications Web dynamiques (site web, intranet, etc.)
-  des « clients riches » (PHP-XUL)
-  des applications client/serveur (PHP-GTK et PHP4Delphi)
-  des services web (SOAP, XML-RPC, REST)
-  des scripts en ligne de commande (CLI)
-  des services s'exécutant en tâche de fond (gestion de quota disque, serveur HTTP, etc.)

Développé par près de 1 000 ingénieurs regroupés au sein de la fondation Apache, PHP réunit autour de lui une communauté qui compte environ 4 500 000 utilisateurs. PHP est un logiciel libre distribué sous une licence Open Source dite non virale qui protège les entreprises utilisatrices en ne les obligeant pas à publier leurs développements. PHP est disponible pour l'ensemble des systèmes d'exploitation courants : *Windows toutes versions, Linux et Unix toutes versions, IBM iSeries (AS/400), SGI IRIX 6.5.x, RISC OS, Novell Netware, Mac OS X, AmigaOS, etc.*