INTRODUCTION A PHP5 PAR L'EXEMPLE

Serge Tahé, université d'Angers, juillet 2011
http://tahe.developpez.com

1 Avant-propos

Ce document propose une liste de scripts Php5 dans différents domaines. Ce n'est pas un cours Php mais un recueil d'exemples destinés à des développeurs ayant déjà utilisé un langage de script tel que Perl, Python, Vbscript. Il est inapproprié pour des lecteurs n'ayant pas cette connaissance préalable. Ces derniers préféreront un tutoriel approfondi tel que celui de Guillaume Rossolini :

```
http://g-rossolini.developpez.com/tutoriels/php/cours/
```

ou plus généralement les tutoriels de *developpez.com* sur Php :

```
http://php.developpez.com/cours
```

On pourra lire également "PHP 5 Power Programming" de Andi Gutmans, Stig Saether Bakken, Derick Rethans disponible à l'url [http://ptgmedia.pearsoncmg.com/images/013147149X/downloads/013147149X_book.pdf].

Les scripts de ce document sont commentés et leur exécution console reproduite. Des explications supplémentaires sont parfois fournies. Le document nécessite une lecture active : pour comprendre un script, il faut à la fois lire son code, ses commentaires et ses résultats d'exécution.

Les exemples du document sont disponibles à l'adresse [http://tahe.developpez.com/divers/php/exemple/].

Table des matières

1AVANT-PROPOS....2
2INSTALLATION D'UN ENVIRONNEMENT DE TRAVAIL....5
2.1.1Installation de WampServer....5
2.1.2Installation de l'IDE Netbeans 7.0....5
3LES BASES DE PHP....7
3.1Un premier exemple....7
3.1.1Le programme (exemple_01)....7
3.2La portée des variables....10
3.2.1Programme 1 (exemple_02)....10
3.2.2Programme 2 (exemple_03)....10
3.3Les tableaux....11
3.3.1Tableaux classiques à une dimension (exemple_04)....11
3.3.2Le dictionnaire (exemple_05)....13
3.3.3Les tableaux à plusieurs dimensions (exemple_06)....14
3.3.4Liens entre chaînes et tableaux (exemple_07)....15
3.4Les chaînes de caractères....16
3.4.1Notation (exemple_08)....16
3.4.2Comparaison (exemple_09)....17
3.5Les expressions régulières (exemple_10)....17
3.6Mode de passage des paramètres des fonctions (exemple_11)....20
3.7Résultats rendus par une fonction (exemple_12)....20
3.8Les fichiers texte (exemple_13)....21
4EXERCICE D'APPLICATION - IMPOTS....24
4.1Le problème....24
4.2Version avec tableaux (impots_01)....24
4.3Version avec fichiers texte (impots_02)....26
5LES OBJETS....28
5.1Toute variable peut devenir un objet doté d'attributs (exemple_14)....28
5.2Une classe Personne sans attributs déclarés (exemple_15)....29
5.3La classe Personne avec attributs déclarés (exemple_16)....29
5.4La classe Personne avec un constructeur (exemple_17)....30
5.5La classe Personne avec contrôle de validité dans le constructeur (exemple_18)....31
5.6Ajout d'une méthode faisant office de second constructeur (exemple_19)....33
5.7Un tableau d'objets Personne (exemple_20)....34
5.8Création d'une classe dérivée de la classe Personne (exemple_21)....35
5.9Création d'une seconde classe dérivée de la classe Personne (exemple_22)....36
5.10Les interfaces (exemple_23)....38
6EXERCICE D'APPLICATION - IMPOTS AVEC OBJETS (IMPOTS_03)....40
7UTILISATION DU SGBD MYSQL....42
7.1Connexion à une base MySQL – 1 (mysql_01)....48
7.2Création d'une table MySQL (mysql_02)....49
7.3Remplissage de la table personnes (mysql_03)....50
7.4Exécution de requêtes SQL quelconques (mysql_04)....53
8EXERCICE IMPOTS AVEC MYSQL....58
8.1Transfert d'un fichier texte dans une table MySQL (txt2mysql)....58
8.2Le programme de calcul de l'impôt (impots_04)....60
9LES FONCTIONS RÉSEAU DE PHP....63
9.1Obtenir le nom ou l'adresse IP d'une machine de l'Internet (inet_01)....63
9.2Un client web (inet_02)....64
9.3Un client smtp (inet_03)....65
9.4Un second programme d'envoi de mail (inet_04)....71
10DES SERVEURS EN PHP....76
10.1Application client/ serveur de date/heure....79
10.1.1Le serveur (web_01)....79
10.1.2Un client (client1_web_01)....86
10.1.3Un deuxième client (client2_web_01)....88
10.2Récupération par le serveur des paramètres envoyés par le client....88
10.2.1Le client GET (client1_web_02)....89
10.2.2Le serveur (web_02)....90
10.2.3Le client POST (client2_web_03)....91
10.2.4Le serveur (web_03)....92

10.3Récupération des variables d'environnement du serveur WEB....93
10.3.1Le serveur (web_04)....93
10.3.2Le client (client1_web_04)....96
10.4Gestion des sessions WEB....98
10.4.1Le fichier de configuration....98
10.4.2Le serveur 1 (web_05)....99
10.4.3Le client 1 (client1_web_05)....101
10.4.4Le serveur 2 (web_06)....104
10.4.5Le client 2 (client1_web_06)....105
10.4.6Le serveur 3 (web_07)....106
10.4.7Le client 3 (client1_web_07)....107
11EXERCICE IMPOTS AVEC UN SERVICE WEB ET UNE ARCHITECTURE À TROIS COUCHES....109
11.1Le script client (clients_impots_05_web)....109
11.2Le service web de calcul de l'impôt....112
11.2.1Les entités du service web (impots_05_entites)....113
11.2.2La couche [dao] (impots_05_dao)....114
11.2.3La couche [métier] (impots_05_metier)....116
11.2.4La couche [web] (impots_05_web)....118
12TRAITEMENT DE DOCUMENTS XML (SIMPLEXML_01)....120
13EXERCICE IMPOTS AVEC XML....122
13.1Le serveur (impots_05B_web)....122
13.2Le client (client_impots_05b_web)....123

2 Installation d'un environnement de travail

Les scripts ont été écrits et testés dans l'environnement suivant :

- un environnement serveur web Apache / Sgbd MySQL / PHP 5.3 appelé **WampServer**.
- l'IDE de développement **Netbeans 7.0**

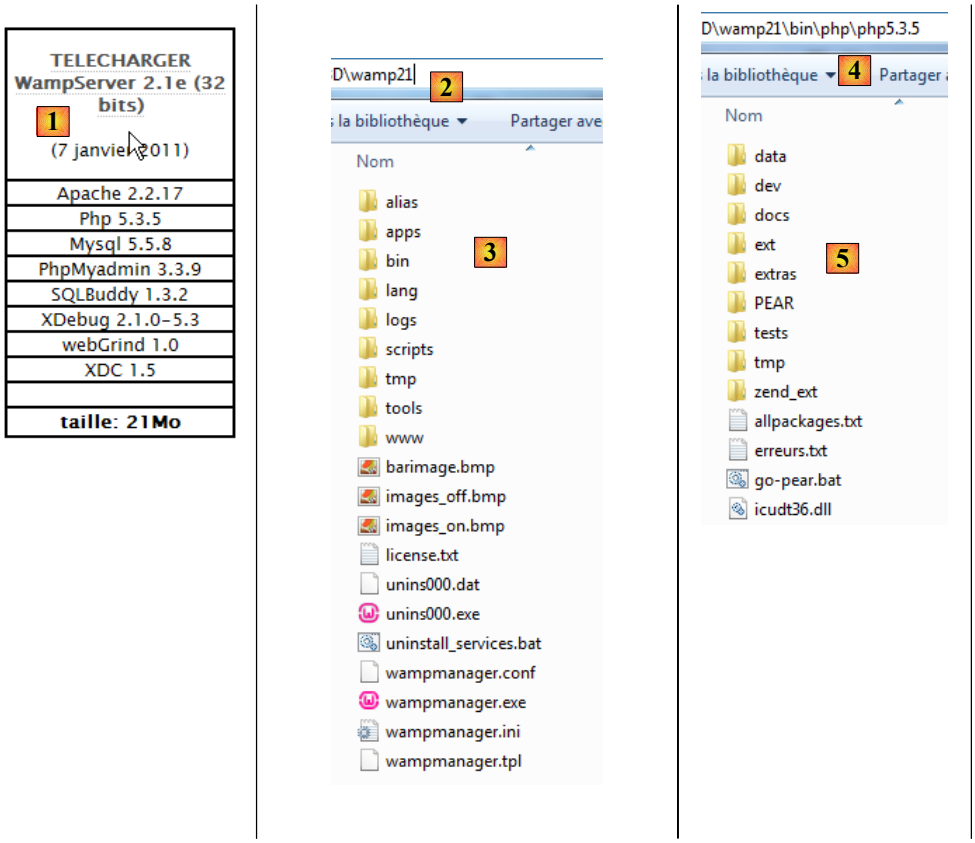
2.1.1 Installation de WampServer

WampServer est un package réunissant plusieurs logiciels :

- un serveur web Apache. Nous l'utiliserons pour l'écriture de scripts web en Php
- le SGBD MySQL
- le langage de script PHP
- un outil d'administration du SGBD MySQL écrit en PHP : phpMyAdmin

Wamp Server peut être téléchargé (juillet 2011) à l'adresse suivante :

```
http://www.wampserver.com/download.php
```

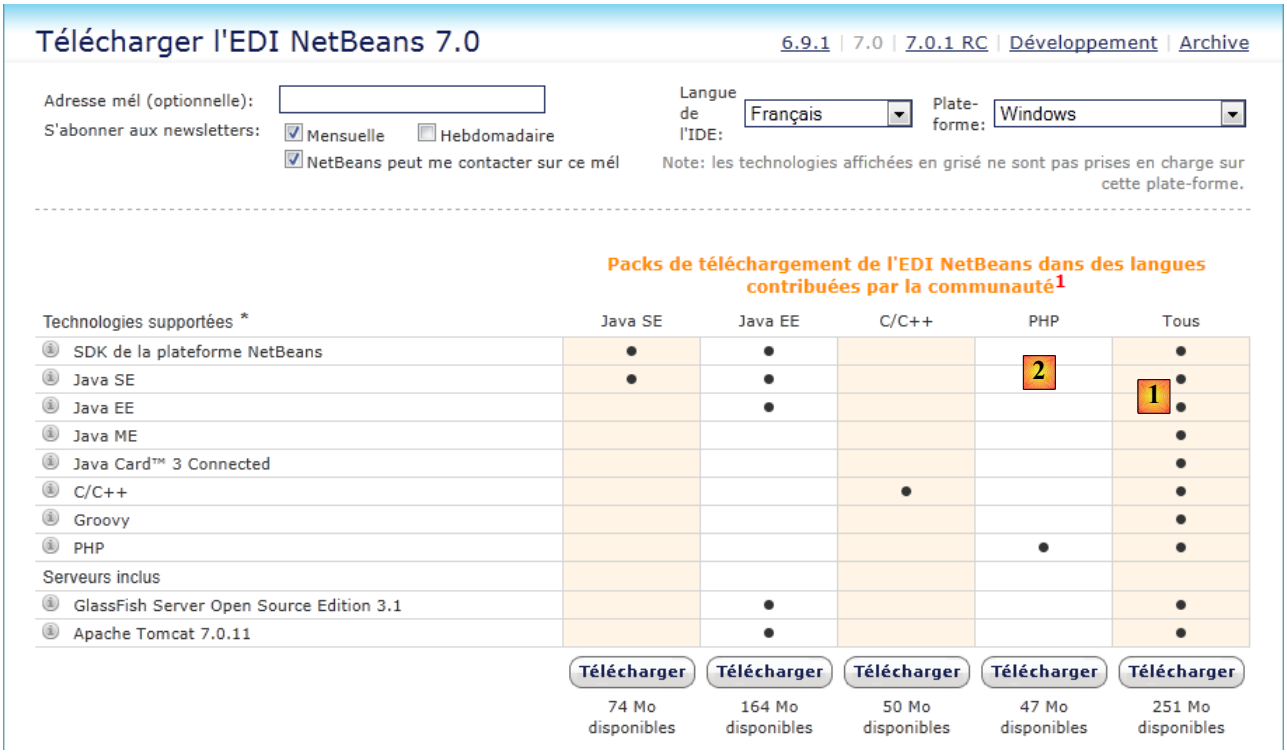


- en [1], on télécharge la version de WampServer qui va bien
- l'installation donne naissance à l'arborescence [2] et [3]
- en [4] et [5] le dossier d'installation de PHP

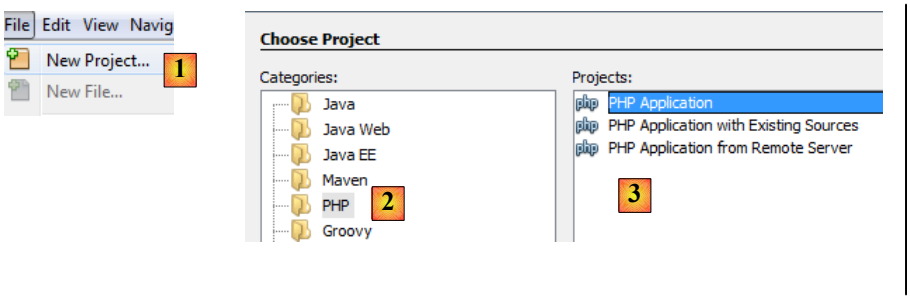
2.1.2 Installation de l'IDE Netbeans 7.0

L'IDE Netbeans 7.0 peut être téléchargé à l'adresse suivante (juillet 2011) :

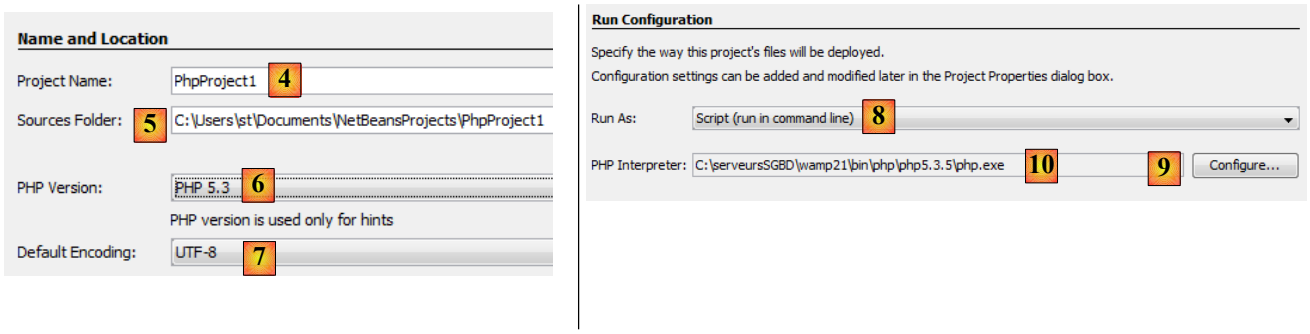
```
http://netbeans.org/downloads/
```



On téléchargera soit la version complète [1], soit la version limitée à PHP [2] qui est plus légère. Une fois Netbeans installé et lancé, on peut créer un premier projet PHP.

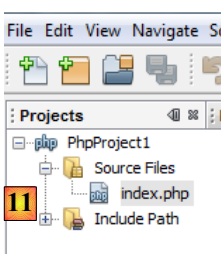


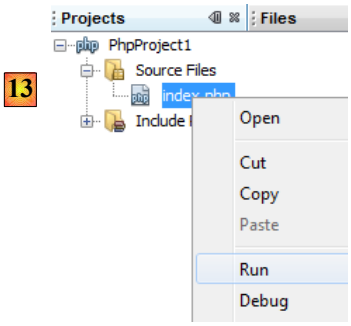
- En [1], prendre l'option File / New Project
- en [2], prendre la catégorie [PHP]
- en [3], prendre le type de projet [PHP Application]



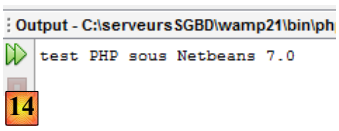
- en [4], donner un nom au projet
- en [5], choisir un dossier pour le projet
- en [6], choisir la version de PHP téléchargée
- en [7], choisir l'encodage UTF-8 pour les fichiers PHP
- en [8], choisir le mode [Script] pour exécuter les scripts PHP en mode ligne de commande. Choisir [Local Web Server] pour exécuter un script PHP dans un environnement web.

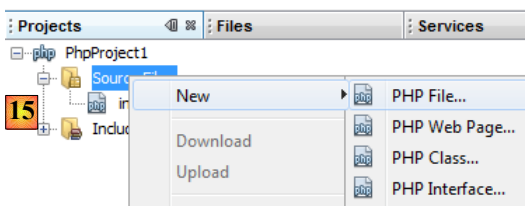
- En [9,10], indiquer le répertoire d'installation de l'interpréteur PHP du package WampServer.
- Choisir [Finish] pour terminer l'assistant de création du projet PHP.

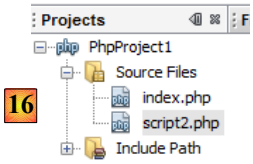




- En [11], le projet est créé avec un script [index.php]
- en [12], on écrit un script PHP minimal
- en [13], on exécute [index.php]







- en [14], les résultats dans la fenêtre [output] de Netbeans.
- en [15], on crée un nouveau script
- en [16], le nouveau script

Le lecteur pourra créer tous les scripts qui vont suivre dans le même projet PHP.

Maintenant que l'environnement de travail a été installé, nous pouvons aborder les bases de PHP.

3 Les bases de PHP

3.1 Un premier exemple

Ci-dessous, on trouvera un programme présentant les premières caractéristiques de PHP.

3.1.1 Le programme (exemple_01)

```
1.  <?php
2.
3.  // ceci est un commentaire
4.  // variable utilisée sans avoir été déclarée
5.  $nom = "dupont";
6.
7.  // un affichage écran
8.  print "nom=$nom\n";
9.
10. // un tableau avec des éléments de type différent
11. $tableau = array("un", "deux", 3, 4);
12.
13. // son nombre d'éléments
```

```
$n = count($tableau);

// une boucle
for ($i = 0; $i < $n; $i++)
  print "tableau[$i]=$tableau[$i]\n";

// initialisation de 2 variables avec le contenu d'un tableau
list($chaine1, $chaine2) = array("chaine1", "chaine2");

// concaténation des 2 chaînes
$chaine3 = $chaine1 . $chaine2;

// affichage résultat
print "[$chaine1,$chaine2,$chaine3]\n";

// utilisation fonction
affiche($chaine1);

// le type d'une variable peut être connu
afficheType($n);
afficheType($chaine1);
afficheType($tableau);

// le type d'une variable peut changer en cours d'exécution
$n = "a changé";
afficheType($n);

// une fonction peut rendre un résultat
$res1 = f1(4);
print "res1=$res1\n";

// une fonction peut rendre un tableau de valeurs
list($res1, $res2, $res3) = f2();
print "(res1,res2,res3)=[$res1,$res2,$res3]\n";

// on aurait pu récupérer ces valeurs dans un tableau
$t = f2();
for ($i = 0; $i < count($t); $i++)
  print "t[$i]=$t[$i]\n";

// des tests
for ($i = 0; $i < count($t); $i++)
// n'affiche que les chaînes
  if (getType($t[$i]) == "string")
    print "t[$i]=$t[$i]\n";

// d'autres tests
for ($i = 0; $i < count($t); $i++){
// n'affiche que les entiers >10
  if (getType($t[$i]) == "integer" and $t[$i] > 10)
    print "t[$i]=$t[$i]\n";
}

// une boucle while
$t = array(8, 5, 0, -2, 3, 4);
$i = 0;
$somme = 0;
while ($i < count($t) and $t[$i] > 0) {
  print "t[$i]=$t[$i]\n";
  $somme+=$t[$i];   //$somme=$somme+$t[$i]
  $i++;             //$i=$i+1
}
print "somme=$somme\n";

// fin programme
exit;
```

```php

//---------------------------------
function affiche($chaine) {
  // affiche $chaine
  print "chaine=$chaine\n";
}

//---------------------------------
function afficheType($variable) {
  // affiche le type de $variable
  print "type[$variable]=" . getType($variable) . "\n";
}

//---------------------------------
function f1($param) {
  // ajoute 10 à $param
  return $param + 10;
}

//---------------------------------
function f2() {
  // rend 3 valeurs
  return array("un", 0, 100);
}
?>
```

Les résultats :

```
nom=dupont
tableau[0]=un
tableau[1]=deux
tableau[2]=3
tableau[3]=4
[chaine1,chaine2,chaine1chaine2]
chaine=chaine1
type[4]=integer
type[chaine1]=string
type[Array]=array
type[a changé]=string
res1=14
(res1,res2,res3)=[un,0,100]
t[0]=un
t[1]=0
t[2]=100
t[0]=un
t[2]=100
t[0]=8
t[1]=5
somme=13
```

Commentaires

- ligne 5 : en PHP, on ne déclare pas le type ds variables. Celles-ci ont un type dynamique qui peut varier au cours du temps
- $nom représente la variable d'identifiant *nom*
- ligne 8 : pour écrire à l'écran, on peut utiliser l'instruction *print* ou l'instruction *echo*
- ligne 11 : le mot clé *array* permet de définir un tableau. La variable *$nom[$i]* représente l'élément *$i* du tableau *$tableau*.
- ligne 14 : la fonction *count($tableau)* rend le nombre d'éléments du tableau *$tableau*
- ligne 18 : les chaînes de caractères sont entourées de guillemets " ou d'apostrophes '. A l'intérieur de guillemets, les variables $*variable* sont évaluées mais pas à l'intérieur d'apostrophes.
- ligne 21 : la fonction *list* permet de rassembler des variables dans une liste et de leur attribuer une valeur avec une unique opération d'affectation. Ici *$chaine1="chaine1"* et *$chaine2="chaine2*".
- ligne 24 : l'opérateur . est l'opérateur de concaténation de chaînes.
- ligne 88 : le mot clé *function* définit une fonction. Une fonction rend ou non des valeurs par l'instruction *return*. Le code appelant peut ignorer ou récupérer les résultats d'une fonction. Une fonction peut être définie n'importe où dans le code.
- ligne 90 : la fonction prédéfinie *getType($variable)* rend une chaîne de caractères représentant le type de *$variable*. Ce type peut changer au cours du temps.
- ligne 79 : la fonction prédéfinie *exit* arrête le script.

3.2 La portée des variables

3.2.1 Programme 1 (exemple_02)

```
1.  <?php
2.
3.  // portée des variables
4.
5.  function f1() {
6.    // on utilise la variable globale $i
7.    $i = &$GLOBALS["i"];
8.    $i++;
9.    $j = 10;
10.   print "f1[i,j]=[$i,$j]\n";
11. }
12.
13. function f2() {
14.   // on utilise la variable globale $i
15.   $i = &$GLOBALS["i"];
16.   $i++;
17.   $j = 20;
18.   print "f2[i,j]=[$i,$j]\n";
19. }
20.
21. function f3() {
22.   // on utilise une variable locale $i
23.   $i = 4;
24.   $j = 30;
25.   print "f3[i,j]=[$i,$j]\n";
26. }
27.
28. // tests
29. $i = 0;
30. $j = 0;  // ces deux variables sont connues d'une fonction f via le tableau $GLOBALS
31. f1();
32. f2();
33. f3();
34. print "test[i,j]=[$i,$j]\n";
35. ?>
```

Les résultats :

```
1.  f1[i,j]=[1,10]
2.  f2[i,j]=[2,20]
3.  f3[i,j]=[4,30]
4.  test[i,j]=[2,0]
```

Commentaires

- lignes 29-30 : définissent 2 variables *$i* et *$j* du programme principal. Ces variables ne sont pas connues à l'intérieur des fonctions. Ainsi, ligne 9, la variable *$j* de la fonction *f1* est une variable locale à la fonction *f1* et est différente de la variable *$j* du programme principal. Une fonction peut accéder à une variable *$variable* du programme principal via un tableau de variables globales appelé *$GLOBALS*. Ligne 7, la notation *$GLOBALS["i"]* désigne la variable globale *$i* du programme principal. Si nous écrivons
  ```
  $i=$GLOBALS["i"]
  ```
  une variable *$i* locale se voit affecter la valeur de la variable *$i* globale. On a là deux variables différentes et modifier la variable locale *$i* ne modifiera pas la variable globale *$i*. La notation
  ```
  $i=&Globals["i"]
  ```
  fait que la variable locale *$i* a la même adresse mémoire que la variable globale *$i*. Manipuler la variable locale *$i* revient alors à manipuler la variable globale *$i*.

3.2.2 Programme 2 (exemple_03)

```
1.  <?php
```

```
2.
3.  // la portée d'une variable est globale aux blocs de code
4.  $i = 0; {
5.    $i = 4;
6.    $i++;
7.  }
8.  print "i=$i\n";
9.  ?>
```

Les résultats :

```
i=5
```

Commentaires

Dans certains langages, une variable définie à l'intérieur d'accolades a la portée de celles-ci : elle n'est pas connue à l'extérieur de celles-ci. Les résultats ci-dessus montrent qu'il n'en est rien en PHP. La variable $i définie ligne 5 à l'intérieur des accolades est la même que celle utilisée lignes 4 et 8 à l'extérieur de celles-ci.

3.3 Les tableaux

3.3.1 Tableaux classiques à une dimension (exemple_04)

```
1.  <?php
2.
3.  // tableaux classiques
4.  // initialisation
5.  $tab1 = array(0, 1, 2, 3, 4, 5);
6.
7.  // parcours - 1
8.  print "tab1 a " . count($tab1) . " éléments\n";
9.  for ($i = 0; $i < count($tab1); $i++)
10.   print "tab1[$i]=$tab1[$i]\n";
11.
12. // parcours - 2
13. print "tab1 a " . count($tab1) . " éléments\n";
14. reset($tab1);
15. while (list($clé, $valeur) = each($tab1))
16.   print "tab1[$clé]=$valeur\n";
17.
18. // ajout d'éléments
19. $tab1[] = $i++;
20. $tab1[] = $i++;
21.
22. // parcours - 3
23. print "tab1 a " . count($tab1) . " éléments\n";
24. $i = 0;
25. foreach ($tab1 as $élément) {
26.   print "tab1[$i]=$élément\n";
27.   $i++;
28. }
29.
30. // suppression dernier élément
31. array_pop($tab1);
32.
33. // parcours - 4
34. print "tab1 a " . count($tab1) . " éléments\n";
35. for ($i = 0; $i < count($tab1); $i++)
36.   print "tab1[$i]=$tab1[$i]\n";
37.
38. // suppression premier élément
39. array_shift($tab1);
40.
41. // parcours - 5
```

```php
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++)
  print "tab1[$i]=$tab1[$i]\n";

// ajout en fin de tableau
array_push($tab1, -2);

// parcours - 6
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++)
  print "tab1[$i]=$tab1[$i]\n";

// ajout en début de tableau
array_unshift($tab1, -1);

// parcours - 7
print "tab1 a " . count($tab1) . " éléments\n";
for ($i = 0; $i < count($tab1); $i++)
  print "tab1[$i]=$tab1[$i]\n";
?>
```

Les résultats :

```
tab1 a 6 éléments
tab1[0]=0
tab1[1]=1
tab1[2]=2
tab1[3]=3
tab1[4]=4
tab1[5]=5
tab1 a 6 éléments
tab1[0]=0
tab1[1]=1
tab1[2]=2
tab1[3]=3
tab1[4]=4
tab1[5]=5
tab1 a 8 éléments
tab1[0]=0
tab1[1]=1
tab1[2]=2
tab1[3]=3
tab1[4]=4
tab1[5]=5
tab1[6]=6
tab1[7]=7
tab1 a 7 éléments
tab1[0]=0
tab1[1]=1
tab1[2]=2
tab1[3]=3
tab1[4]=4
tab1[5]=5
tab1[6]=6
tab1 a 6 éléments
tab1[0]=1
tab1[1]=2
tab1[2]=3
tab1[3]=4
tab1[4]=5
tab1[5]=6
tab1 a 7 éléments
tab1[0]=1
tab1[1]=2
tab1[2]=3
tab1[3]=4
tab1[4]=5
tab1[5]=6
tab1[6]=-2
tab1 a 8 éléments
tab1[0]=-1
tab1[1]=1
tab1[2]=2
tab1[3]=3
tab1[4]=4
tab1[5]=5
```

```
54. tab1[6]=6
55. tab1[7]=-2
```

Commentaires

Le programme ci-dessus montre des opérations de manipulation d'un tableau de valeurs. Il existe deux notations pour les tableaux en PHP :

```
1.  $tableau=array("un",2,"trois")
2.  $contraires=array("petit"=>"grand", "beau"=>"laid", "cher"=>"bon marché")
```

Le tableau 1 est appelé tableau et le tableau 2 un dictionnaire où les éléments sont notés clé => valeur. La notation *$contraires["beau"]* désigne la valeur associée à la clé "beau". C'est donc ici la chaîne "laid". Le tableau 1 n'est qu'une variante du dictionnaire et pourrait être noté :

```
$tableau=array(0=>"un",1=>2,2=>"trois")
```

On a ainsi *$tableau[2]="trois"*. Au final, il n'y a que des dictionnaires. Dans le cas d'un tableau classique de *n* éléments, les clés sont les nombres entiers de l'intervalle [*0,n-1*].

- ligne 15 : la fonction *each($tableau)* permet de parcourir un dictionnaire. A chaque appel, elle rend une paire (clé,valeur) de celui-ci.
- ligne 14 : la fonction *reset($dictionnaire)* positionne la fonction *each* sur la première paire (clé,valeur) du dictionnaire.
- ligne 15 : la boucle *while* s'arrête lorsque la fonction *each* rend une paire vide à la fin du dictionnaire.
- ligne 19 : la notation *$tableau[]=valeur* ajoute l'élément *valeur* comme dernier élément de *$tableau*.
- ligne 25 : le tableau est parcouru avec un *foreach*. Cet élément syntaxique permet de parcourir un dictionnaire, donc un tableau, selon deux syntaxes :

```
foreach($dictionnaire as $clé=>$valeur)
foreach($tableau as $valeur)
```

  La première syntaxe ramène une paire (*clé,valeur*) à chaque itération alors que la seconde syntaxe ne ramène que l'élément *valeur* du dictionnaire.
- ligne 31 : la fonction *array_pop($tableau)* supprime le dernier élément de $*tableau*
- ligne 39 : la fonction *array_shift($tableau)* supprime le premier élément de $*tableau*
- ligne 47 : la fonction *array_push($tableau,valeur)* ajoute *valeur* comme dernier élément de $*tableau*
- ligne 39 : la fonction *array_unshift($tableau,valeur)* ajoute *valeur* comme premier élément de $*tableau*

3.3.2 Le dictionnaire (exemple_05)

```
1.  <?php
2.
3.  // dictionnaires
4.  $conjoints = array("Pierre" => "Gisèle", "Paul" => "Virginie", "Jacques" => "Lucette", "Jean" => "");
5.
6.  // parcours - 1
7.  print "Nombre d'éléments du dictionnaire : " . count($conjoints) . "\n";
8.  reset($conjoints);
9.  while (list($clé, $valeur) = each($conjoints))
10.   print "conjoints[$clé]=$valeur\n";
11.
12. // tri du dictionnaire sur la clé
13. ksort($conjoints);
14.
15. // parcours - 2
16. reset($conjoints);
17. while (list($clé, $valeur) = each($conjoints))
18.   print "conjoints[$clé]=$valeur\n";
19.
20. // liste des clés du dictionaire
21. $clés = array_keys($conjoints);
22. for ($i = 0; $i < count($clés); $i++)
23.   print "clés[$i]=$clés[$i]\n";
24.
25. // liste des valeurs du dictionnaire
26. $valeurs = array_values($conjoints);
```

```
27. for ($i = 0; $i < count($valeurs); $i++)
28.   print "valeurs[$i]=$valeurs[$i]\n";
29. 
30. // recherche d'une clé
31. existe($conjoints, "Jacques");
32. existe($conjoints, "Lucette");
33. existe($conjoints, "Jean");
34. 
35. // suppression d'une clé-valeur
36. unset($conjoints["Jean"]);
37. print "Nombre d'éléments du dictionnaire : " . count($conjoints) . "\n";
38. foreach ($conjoints as $clé => $valeur) {
39.   print "conjoints[$clé]=$valeur\n";
40. }
41. 
42. // fin
43. exit;
44. 
45. function existe($conjoints, $mari) {
46.   // vérifie si la clé $mari existe dans le dictionnaire $conjoints
47.   if (isset($conjoints[$mari]))
48.     print "La clé [$mari] existe associée à la valeur [$conjoints[$mari]]\n";
49.   else
50.     print "La clé [$mari] n'existe pas\n";
51. }
52. 
53. ?>
```

Les résultats :

```
1.  Nombre d'éléments du dictionnaire : 4
2.  conjoints[Pierre]=Gisèle
3.  conjoints[Paul]=Virginie
4.  conjoints[Jacques]=Lucette
5.  conjoints[Jean]=
6.  conjoints[Jacques]=Lucette
7.  conjoints[Jean]=
8.  conjoints[Paul]=Virginie
9.  conjoints[Pierre]=Gisèle
10. clés[0]=Jacques
11. clés[1]=Jean
12. clés[2]=Paul
13. clés[3]=Pierre
14. valeurs[0]=Lucette
15. valeurs[1]=
16. valeurs[2]=Virginie
17. valeurs[3]=Gisèle
18. La clé [Jacques] existe associée à la valeur [Lucette]
19. La clé [Lucette] n'existe pas
20. La clé [Jean] existe associée à la valeur []
21. Nombre d'éléments du dictionnaire : 3
22. conjoints[Jacques]=Lucette
23. conjoints[Paul]=Virginie
24. conjoints[Pierre]=Gisèle
```

Commentaires

Le code précédent applique à un dictionnaire ce qui a été vu auparavant pour un simple tableau. Nous ne commentons que les nouveautés :

- ligne 13 : la fonction *ksort (key sort)* permet de trier un dictionnaire dans l'ordre naturel de la clé.
- ligne 21 : la fonction *array_keys($dictionnaire)* rend la liste des clés du dictionnaire sous forme de tableau
- ligne 26 : la fonction *array_values($dictionnaire)* rend la liste des valeurs du dictionnaire sous forme de tableau
- ligne 47 : la fonction *isset($variable)* rend true si la variable $*variable* a été définie, false sinon.
- ligne 36 : la fonction *unset($variable)* supprime la variable $*variable*.

3.3.3 Les tableaux à plusieurs dimensions (exemple_06)

```php
<?php

// tableaux classiques multidimensionnels
// initialisation
$multi = array(array(0, 1, 2), array(10, 11, 12, 13), array(20, 21, 22, 23, 24));
// parcours
for ($i1 = 0; $i1 < count($multi); $i1++)
  for ($i2 = 0; $i2 < count($multi[$i1]); $i2++)
    print "multi[$i1][$i2]=" . $multi[$i1][$i2] . "\n";
// dictionnaires multidimensionnels
// initialisation
$multi = array("zéro" => array(0, 1, 2), "un" => array(10, 11, 12, 13), "deux" => array(20, 21, 22, 23, 24));
// parcours
foreach ($multi as $clé => $valeur)
  for ($i2 = 0; $i2 < count($valeur); $i2++)
    print "multi[$clé][$i2]=" . $multi[$clé][$i2] . "\n";
?>
```

Résultats :

```
multi[0][0]=0
multi[0][1]=1
multi[0][2]=2
multi[1][0]=10
multi[1][1]=11
multi[1][2]=12
multi[1][3]=13
multi[2][0]=20
multi[2][1]=21
multi[2][2]=22
multi[2][3]=23
multi[2][4]=24
multi[zéro][0]=0
multi[zéro][1]=1
multi[zéro][2]=2
multi[un][0]=10
multi[un][1]=11
multi[un][2]=12
multi[un][3]=13
multi[deux][0]=20
multi[deux][1]=21
multi[deux][2]=22
multi[deux][3]=23
multi[deux][4]=24
```

Commentaires

- ligne 5 : les éléments du tableau $*multi* sont eux-mêmes des tableaux
- ligne 12 : le tableau $*multi* devient un dictionnaire (*clé,valeur*) où chaque *valeur* est un tableau

3.3.4 Liens entre chaînes et tableaux (exemple_07)

```php
<?php

// chaîne vers tableau
$chaine = "1:2:3:4";
$tab = explode(":", $chaine);

// parcours tableau
print "tab a " . count($tab) . " éléments\n";
for ($i = 0; $i < count($tab); $i++)
  print "tab[$i]=$tab[$i]\n";

// tableau vers chaîne
$chaine2 = implode(":", $tab);
print "chaine2=$chaine2\n";

// ajoutons un champ vide
$chaine.=":";
```

```php
print "chaîne=$chaine\n";
$tab = explode(":", $chaine);

// parcours tableau
print "tab a " . count($tab) . " éléments\n";
for ($i = 0; $i < count($tab); $i++)
  print "tab[$i]=$tab[$i]\n"; // on a maintenant 5 éléments, le dernier étant vide

// ajoutons de nouveau un champ vide
$chaine.=":";
print "chaîne=$chaine\n";
$tab = explode(":", $chaine);

// parcours tableau
print "tab a " . count($tab) . " éléments\n";
for ($i = 0; $i < count($tab); $i++)
  print "tab[$i]=$tab[$i]\n"; // on a maintenant 6 éléments, les deux derniers étant vides
?>
```

Résultats :

```
tab a 4 éléments
tab[0]=1
tab[1]=2
tab[2]=3
tab[3]=4
chaine2=1:2:3:4
chaîne=1:2:3:4:
tab a 5 éléments
tab[0]=1
tab[1]=2
tab[2]=3
tab[3]=4
tab[4]=
chaîne=1:2:3:4::
tab a 6 éléments
tab[0]=1
tab[1]=2
tab[2]=3
tab[3]=4
tab[4]=
tab[5]=
```

Commentaires

- ligne 5 : la fonction *explode($séparateur,$chaine)* permet de récupérer les champs de $*chaine* séparés par $*séparateur*. Ainsi *explode(":",$chaine)* permet de récupérer sous forme de tableau les éléments de $*chaine* qui sont séparés par la chaîne ":".
- la foncion implode($séparateur,$tableau) fait l'opération inverse de la fonction explode. Elle rend une chaîne de caractères formée des éléments de $tableau séparés par $séparateur.

3.4 Les chaînes de caractères

3.4.1 Notation (exemple_08)

```php
<?php

// notation des chaînes
$chaine1 = "un";
$chaine2 = 'un';
print "[$chaine1,$chaine2]\n";
?>
```

Résultats :

```
[un,un]
```

3.4.2 Comparaison (exemple_09)

```php
<?php

// tests de comparaisons de chaînes
compare("abcd", "abcd");
compare("", "");
compare("1", "");
exit;

function compare($chaine1, $chaine2) {
  // compare chaine1 et chaine2
  if ($chaine1 == $chaine2)
    print "[$chaine1] est égal à [$chaine2]\n";
  else
    print "[$chaine1] est différent de [$chaine2]\n";
}

?>
```

Résultats :

```
[abcd] est égal à [abcd]
[] est égal à []
[1] est différent de []
```

3.5 Les expressions régulières (exemple_10)

```php
<?php

// expression régulières en php
// récupérer les différents champs d'une chaîne

// le modèle : une suite de chiffres entourée de caractères quelconques
// on ne veut récupérer que la suite de chiffres
$modèle = "/(\d+)/";
// on confronte la chaîne au modèle
compare($modèle, "xyz1234abcd");
compare($modèle, "12 34");
compare($modèle, "abcd");

// le modèle : une suite de chiffres entourée de caractères quelconques
// on veut la suite de chiffres ainsi que les champs qui suivent et précèdent
$modèle = "/^(.*?)(\d+)(.*?)$/";
// on confronte la chaîne au modèle
compare($modèle, "xyz1234abcd");
compare($modèle, "12 34");
compare($modèle, "abcd");

// le modèle - une date au format jj/mm/aa
$modèle = "/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/";
compare($modèle, "10/05/97");
compare($modèle, "  04/04/01  ");
compare($modèle, "5/1/01");

// le modèle - un nombre décimal
$modèle = "/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/";
compare($modèle, "187.8");
compare($modèle, "-0.6");
compare($modèle, "4");
compare($modèle, ".6");
compare($modèle, "4.");
compare($modèle, " + 4");

```

```php
// fin
exit;

// ---------------------------------------------------------------------------
function compare($modèle, $chaîne) {
  // compare la chaîne $chaîne au modèle $modèle
  // on confronte la chaîne au modèle
  $correspond = preg_match($modèle, $chaîne, $champs);
  // affichage résultats
  print "\nRésultats($modèle,$chaîne)\n";
  if ($correspond) {
    for ($i = 0; $i < count($champs); $i++)
      print "champs[$i]=$champs[$i]\n";
  } else
    print "La chaîne [$chaîne] ne correspond pas au modèle [$modèle]\n";
}
?>
```

Résultats :

```
Résultats(/(\d+)/,xyz1234abcd)
champs[0]=1234
champs[1]=1234

Résultats(/(\d+)/,12 34)
champs[0]=12
champs[1]=12

Résultats(/(\d+)/,abcd)
La chaîne [abcd] ne correspond pas au modèle [/(\d+)/]

Résultats(/^(.*?)(\d+)(.*?)$/,xyz1234abcd)
champs[0]=xyz1234abcd
champs[1]=xyz
champs[2]=1234
champs[3]=abcd

Résultats(/^(.*?)(\d+)(.*?)$/,12 34)
champs[0]=12 34
champs[1]=
champs[2]=12
champs[3]= 34

Résultats(/^(.*?)(\d+)(.*?)$/,abcd)
La chaîne [abcd] ne correspond pas au modèle [/^(.*?)(\d+)(.*?)$/]

Résultats(/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/,10/05/97)
champs[0]=10/05/97
champs[1]=10
champs[2]=05
champs[3]=97

Résultats(/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/,  04/04/01  )
champs[0]=  04/04/01
champs[1]=04
champs[2]=04
champs[3]=01

Résultats(/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/,5/1/01)
La chaîne [5/1/01] ne correspond pas au modèle [/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/]

Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,187.8)
champs[0]=187.8
champs[1]=
champs[2]=187.8

Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,-0.6)
champs[0]=-0.6
champs[1]=-
champs[2]=0.6

Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,4)
champs[0]=4
champs[1]=
champs[2]=4

Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,.6)
```

```
58. champs[0]=.6
59. champs[1]=
60. champs[2]=.6
61.
62. Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/,4.)
63. champs[0]=4.
64. champs[1]=
65. champs[2]=4.
66.
67. Résultats(/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/, + 4)
68. champs[0]= + 4
69. champs[1]=+
70. champs[2]=4
```

Commentaires

- lignes 3-4 : nous utilisons ici les expression régulières pour récupérer les divers champs d'une chaîne de caractères. Les expressions régulières permetent de dépasser les limites de la fonction *implode*. Le principe est de comparer une chaîne de caractères à une autre chaîne appelée *modèle* à l'aide de la fonction *preg_match* :

```
$correspond = preg_match($modèle, $chaîne, $champs);
```

La fonction *preg_match* rend un booléen si le *modèle* peut être trouvé dans la *chaîne*. Si oui, $*champs[0]* représente la sous-chaîne correspondant au modèle. Par ailleurs, si modèle contient des sous-modèles entre parenthèses, $*champs[1]* est le morceau de $*chaîne* correspondant au 1er sous-modèle, $*champs[2]* est le morceau de $*chaîne* correspondant au 2ième sous-modèle, etc...

Considérons le 1er exemple. Le modèle est défini ligne 8 : il désigne une suite de un ou plusieurs (+) chiffres (\d) placés n'importe où dans une chaîne. Par ailleurs, le modèle définit un sous-modèle entouré de parenthèses.

- ligne 10 : le modèle **/(\d+)/** (suite d'un ou plusieurs chiffres n'importe où dans la chaîne) est comparé à la chaîne "**xyz1234abcd**". On voit que la sous-chaîne 1234 correspond au modèle. On aura donc $champs[0] égal à "1234". Par ailleurs, le modèle a des sous-modèles entre parenthèses. On aura $champs[1]="1234".
- ligne 11 : le modèle **/(\d+)/** est comparé à la chaîne "**12 34**". On voit que les sous-chaînes 12 et 34 correspondent au modèle. La comparaison s'arrête à la première sous-chaîne correspondant au modèle. On aura donc, $champs[0]=12 et $champs[1]=12.
- ligne 12 : le modèle **/(\d+)/** est comparé à la chaîne "**abcd**". Aucune correspondance n'est trouvée.

Explicitons les modèles utilisés dans la suite du code :

```
$modèle = "/^(.*?)(\d+)(.*?)$/";
```

début de chaîne (^), puis 0 ou plusieurs (*) caractères quelconques (.) puis 1 ou plusieurs (+) chiffres, puis de nouveau 0 ou plusieurs (*) caractères quelconques (.). Le modèle (.*) désigne 0 ou plusieurs caractères quelconques. Un tel modèle va correspondre à n'importe quelle chaîne. Ainsi le modèle /^(.*)(\d+)(.*)$/ ne sera-t-il jamais trouvé car le premier sous-modèle (.*) va absorber toute la chaîner. Le modèle (.*?)(\d+) désigne lui 0 ou plusieurs caractères quelconques **jusqu'au sous-modèle suivant (?)**, ici \d+. Donc les chiffres ne sont maintenant plus absorbés par le modèle (.*). Le modèle ci-dessus correspond donc à [début de chaîne (^), une suite de caractères quelconques (.*?), une suite d'un ou plusieurs chiffres (\d+), une suite de caractères quelconques (.*?), la fin de la chaîne ($)].

```
$modèle = "/^\s*(\d\d)\/(\d\d)\/(\d\d)\s*$/";
```

correspond à [début de chaîne (^), 2 chiffres (\d\d), le caractère / (\/), 2 chiffres, /, 2 chiffres, une suite de 0 ou plusieurs espaces (\s*), la fin de chaîne ($)]

```
$modèle = "/^\s*([+|-]?)\s*(\d+\.\d*|\.\d+|\d+)\s*/";
```

[début de chaîne (^), 0 ou plusieurs espaces (\s*), un signe + ou - [+|-] présent 0 ou 1 fois (?), une suite de 0 ou plusieurs espaces (\s*), 1 ou plusieurs chiffres suivis d'un point décimal suivi de zéro ou plusieurs chiffres (\d+\.\d*) ou (|) un point décimal (\.) suivi d'un ou plusieurs chiffres (\d+) ou (|) un ou plusieurs chiffres (\d+), une suite de 0 ou plusieurs espaces (\s*)].

3.6 Mode de passage des paramètres des fonctions (exemple_11)

```php
<?php

// mode de pasage des paramètres

function f(&$i, $j) {
  // $i sera obtenu par référence
  // $j sera obtenu par valeur
  $i++;
  $j++;
  print "f[i,j]=[$i,$j]\n";
}

// ----------------------------------------tests
$i = 0;
$j = 0;
// $i et $j passés par valeurs
f($i, $j);
print "test[i,j]=[$i,$j]\n";
?>
```

Résultats :

```
f[i,j]=[1,1]
test[i,j]=[1,0]
```

Commentaires

Le code ci-dessus montre les deux modes de passage de paramètres à une fonction. Prenons l'exemple suivant :

```
function f(&$a,$b){
…
}

// programme principal
$i=10; $j=20;
f($i,$j);
```

- ligne 1 : définit les paramètres formels $a et $b de la fonction f. Celle-ci manipule ces deux paramètres formels et rend un résultat.
- ligne 7 : appel de la fonction f avec deux paramètres effectifs $i et $j. Les liens entre les paramètres formels ($a,$b) et les paramètres effectifs ($i,$j) sont définis par la ligne 7
  - &$a : le signe & indique que le paramètre formel $a prendra pour valeur l'adresse du paramètre effectif $i. Dit autrement, $a et $i sont deux références sur un même emplacement mémoire. Manipuler le paramètre formel $a revient à manipuler le paramètre effectif $i. C'est ce que montre l'exécution du code. Ce mode de passage convient aux paramètres de sortie et aux données volumineuses telles que les tableaux et dictionnaires. On appelle ce mode passage, passage par référence.
  - $b : le paramètre formel $b prendra pour valeur celle du paramètre effectif $j. C'est un passage par valeur. Les paramètres formels et effectifs sont deux variables différentes. Manipuler le paramètre formel $b n'a aucune incidence sur le paramètre effectif $j. C'est ce que montre l'exécution du code. Ce mode de passage convient aux paramètres d'entrée.
  - Soit la fonction *échange* qui admet deux paramètres formels $*a* et $*b*. La fonction échange la valeur de ces deux paramètres. Ainsi lors d'un appel échange *($i,$j)*, le code appelant s'attend à ce que les valeurs des deux paramètres effectifs soient échangés. Ce sont donc des paramètres de sortie (ils sont modifiés). On écrira donc :

```
function échange(&$a,&$b){....}
```

3.7 Résultats rendus par une fonction (exemple_12)

```php
<?php

// résultats rendus par une fonction
// une fonction peut rendre plusieurs valeurs dans un tableau
list($res1, $res2, $res3) = f1(10);
```

```
6.  print "[$res1,$res2,$res3]\n";
7.  $res = f1(10);
8.  for ($i = 0; $i < count($res); $i++)
9.    print "res[$i]=$res[$i]\n";
10.
11. // une fonction peut rendre un pseudo-objet
12. $res = f2(10);
13. print "[$res->res1,$res->res2,$res->res3]\n";
14.
15. // fin
16. exit;
17.
18. // fonction f1
19. function f1($valeur) {
20.   // rend un tableau ($valeur+1,$aleur+2,$baleur+3)
21.   return array($valeur + 1, $valeur + 2, $valeur + 3);
22. }
23.
24. // fonction f2
25. function f2($valeur) {
26.   // rend un pseudo-objet ($valeur+1,$aleur+2,$baleur+3)
27.   $res->res1 = $valeur + 1;
28.   ;
29.   $res->res2 = $valeur + 2;
30.   $res->res3 = $valeur + 3;
31.   // rend l'objet
32.   return $res;
33. }
34.
35. ?>
```

Résultats

```
1. [11,12,13]
2. res[0]=11
3. res[1]=12
4. res[2]=13
5. [11,12,13]
```

Commentaires

- le programme précédent montre qu'une fonction php peut rendre un ensemble de résultats et non un seul, sous la forme d'un tableau ou d'un objet. La notion d'objet est explicitée un peu plus loin
- lignes 19-22 : la fonction f1 rend plusieurs valeurs sous la forme d'un tableau
- lignes 25-33 : la fonction f2 rend plusieurs valeurs sous la forme d'un objet

3.8 Les fichiers texte (exemple_13)

```
1.  <?php
2.
3.  // exploitation séquentielle d'un fichier texte
4.  // celui-ci est un ensemble de lignes de la forme login:pwd:uid:gid:infos:dir:shell
5.  // chaque ligne est mis dans un dictionnaire sous la forme login => uid:gid:infos:dir:shell
6.
7.  // on fixe le nom du fichier
8.  $INFOS = "infos.txt";
9.
10. // on l'ouvre en création
11. if (!$fic = fopen($INFOS, "w")) {
12.   print "Erreur d'ouverture du fichier $INFOS en écriture\n";
13.   exit;
14. }
15.
16. // on génère un contenu arbitraire
17. for ($i = 0; $i < 100; $i++)
```

```
  fputs($fic, "login$i:pwd$i:uid$i:gid$i:infos$i:dir$i:shell$i\n");
// on ferme le fichier
fclose($fic);

// on l'exploite - fgets garde la marque de fin de ligne
// cela permet de ne pas récupérer une chaîne vide lors de la lecture d'une ligne blanche
// on l'ouvre en création
if (!$fic = fopen($INFOS, "r")) {
  print "Erreur d'ouverture du fichier $INFOS en lecture\n";
  exit;
}
while ($ligne = fgets($fic, 1000)) {
  // on supprime la marque de fin de ligne si elle existe
  $ligne = cutNewLineChar($ligne);
  // on met la ligne dans un tableau
  $infos = explode(":", $ligne);
  // on récupère le login
  $login = array_shift($infos);
  // on néglige le pwd
  array_shift($infos);
  // on crée une entrée dans le dictionnaire
  $dico[$login] = $infos;
}
// on le ferme
fclose($fic);

// exploitation du dictionnaire
afficheInfos($dico, "login10");
afficheInfos($dico, "X");

// fin
exit;

// ------------------------------------------------------------------------
function afficheInfos($dico, $clé) {
  // affiche la valeur associée à clé dans le dictionnaire $dico si elle existe
  if (isset($dico[$clé])) {
    // valeur existe - est-ce un tableau ?
    $valeur = $dico[$clé];
    if (is_array($valeur)) {
      print "[$clé," . join(":", $valeur) . "]\n";
    } else {
      // $valeur n'est pas un tableau
      print "[$clé,$valeur]\n";
    }
  } else {
    // $clé n'est pas une clé du dictionnaire $dico
    print "la clé [$clé] n'existe pas\n";
  }
}

// ------------------------------------------------------------------------
function cutNewLinechar($ligne) {
  // on supprime la marque de fin de ligne de $ligne si elle existe
  $L = strlen($ligne);  // longueur ligne
  while (substr($ligne, $L - 1, 1) == "\n" or substr($ligne, $L - 1, 1) == "\r") {
    $ligne = substr($ligne, 0, $L - 1);
    $L--;
  }
  // fin
  return($ligne);
}

?>
```

Le fichier infos.txt :

```
login0:pwd0:uid0:gid0:infos0:dir0:shell0
login1:pwd1:uid1:gid1:infos1:dir1:shell1
login2:pwd2:uid2:gid2:infos2:dir2:shell2
...
login98:pwd98:uid98:gid98:infos98:dir98:shell98
login99:pwd99:uid99:gid99:infos99:dir99:shell99
```

Les résultats :

```
1.  [login10,uid10:gid10:infos10:dir10:shell10]
2.  la clé [X] n'existe pas
```

Commentaires

- ligne 11 : *fopen(nom_fichier,"w")* ouvre le fichier *nom_fichier* en écriture (*w=write*). Si le fichier n'existe pas, il est créé. S'il existe, il est vidé. Si la création échoue, *fopen* rend une valeur *false*. Dans l'instruction ***if*** *(!$fic = fopen($INFOS, "w"))* {…}, il y a deux opérations successives : 1) *$fic=fopen(..)* 2) *if(! $fic)* {...}
- ligne 18 : *fputs($fic,$chaîne)* écrit *chaîne* dans le fichier $*fic*.
- ligne 20 : *fclose($fic)* ferme le fichier $*fic*.
- ligne 25 : *fopen(nom_fichier,"r")* ouvre le fichier *nom_fichier* en lecture (*r=read*). Si l'ouverture échoue (le fichier n'existe pas), *fopen* rend une valeur *false*.
- ligne 29 : *fgets($fic,1000)* lit la ligne suivante du fichier dans la limite de 1000 caractères. Dans l'opération ***while*** *($ligne = fgets($fic, 1000))* {…}, il y a deux opérations successives 1) $ligne=fgets(...) 2) while (! $ligne). Après que le dernier caractère du fichier a été lu, la fonction *fgets* rend la chaîne vide et la boucle *while* s'arrête.
- ligne 31 : la fonction *fgets* lit une ligne de texte, caractère de fin de ligne inclus. La fonction cutNewLineChar ds lignes 70-79 élimine l'éventuel caractère de fin de ligne.
- ligne 72 : la fonction *strlen($chaîne)* rend le nombre de caractères de $*chaîne*.
- ligne 73 : la fonction *substr($ligne, $position, $taille)* rend $*taille* caractères de $*ligne*, pris à partir du caractère n° $*position*, le 1er caractère ayant le n° 0. Sur les machines windows, la marque de fin de ligne est "\r\n". Sur les machines Unix, c'est la chaîne "\n".
- ligne 35 : la fonction *array_shift($tableau)* élimine le 1er élément de $*tableau* et le rend comme résultat.
- ligne 37 : la fonction *array_shift($tableau)* est utilisée mais on néglige son résultat
- ligne 57 : la fonction *is_array($variable)* rend *true* si $*variable* est un tableau, *false* sinon.
- ligne 58 : la fonction *join* fait la même chose que la fonction *implode* déjà rencontrée

4 Exercice d'application - IMPOTS

4.1 Le problème

On se propose d'écrire un programme permettant de calculer l'impôt d'un contribuable. On se place dans le cas simplifié d'un contribuable n'ayant que son seul salaire à déclarer :

- on calcule le nombre de parts du salarié **nbParts=nbEnfants/2 +1** s'il n'est pas marié, **nbEnfants/2+2** s'il est marié, où *nbEnfants* est son nombre d'enfants.
- on calcule son revenu imposable **R=0.72*S** où S est son salaire annuel
- on calcule son coefficient familial **Q=R/N**
  on calcule son impôt I d'après les données suivantes

```
12620.0     0       0
13190       0.05    631
15640       0.1     1290.5
24740       0.15    2072.5
31810       0.2     3309.5
39970       0.25    4900
48360       0.3     6898.5
55790       0.35    9316.5
92970       0.4     12106
127860      0.45    16754.5
151250      0.50    23147.5
172040      0.55    30710
195000      0.60    39312
0           0.65    49062
```

Chaque ligne a 3 champs. Pour calculer l'impôt I, on recherche la première ligne où *QF<=champ1*. Par exemple, si QF=30000 on trouvera la ligne

```
24740    0.15    2072.5
```

L'impôt I est alors égal à **0.15*R - 2072.5*nbParts**. Si QF est tel que la relation *QF<=champ1* n'est jamais vérifiée, alors ce sont les coefficients de la dernière ligne qui sont utilisés. Ici :

```
0          0.65    49062
```

ce qui donne l'impôt I=0.65*R – 49062*nbParts.

4.2 Version avec tableaux (impots_01)

Nous présentons un premier programme où :

- les données nécessaires au calcul de l'impôt sont dans trois tableaux (limites, coeffR, coeffN)
- les données des contribuables (marié, enfants, salaire) sont dans un premier fichier texte
- les résultats du calcul de l'impôt (marié, enfants, salaire, impôt) sont mémorisés dans un second fichier texte

```
1.  <?php
2.
3.  // définition des constantes
4.  $DATA = "data.txt";
5.  $RESULTATS = "resultats.txt";
6.  $limites = array(12620, 13190, 15640, 24740, 31810, 39970, 48360, 55790, 92970, 127860, 151250, 172040, 195000);
7.  $coeffR = array(0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55, 0.6, 0.65);
8.  $coeffN = array(0, 631, 1290.5, 2072.5, 3309.5, 4900, 6898.5, 9316.5, 12106, 16754.5, 23147.5, 30710, 39312, 49062);
9.
10. // lecture des données
11. $data = fopen($DATA, "r");
12. if (!$data) {
13.   print "Impossible d'ouvrir en lecture le fichier des données [$DATA]\n";
14.   exit;
15. }
```

```
16.
17.  // ouverture fichier des résultats
18.  $résultats = fopen($RESULTATS, "w");
19.  if (!$résultats) {
20.    print "Impossible de créer le fichier des résultats [$RESULTATS]\n";
21.    exit;
22.  }
23.
24.  // on exploite la ligne courante du fichier des données
25.  while ($ligne = fgets($data, 100)) {
26.    // on enlève l'éventuelle marque de fin de ligne
27.    $ligne = cutNewLineChar($ligne);
28.    // on récupère les 3 champs marié,enfants,salaire qui forment $ligne
29.    list($marié, $enfants, $salaire) = explode(",", $ligne);
30.    // on calcule l'impôt
31.    $impôt = calculImpots($marié, $enfants, $salaire, $limites, $coeffR, $coeffN);
32.    // on inscrit le résultat dans le fichier des résultats
33.    fputs($résultats, "$marié:$enfants:$salaire:$impôt\n");
34.    // contribuable suivant
35.  }
36.  // on ferme les fichiers
37.  fclose($data);
38.  fclose($résultats);
39.
40.  // fin
41.  exit;
42.
43.  // ------------------------------------------------------------------------
44.  function cutNewLinechar($ligne) {
45.    // on supprime la marque de fin de ligne de $ligne si elle existe
46.    $L = strlen($ligne);  // longueur ligne
47.    while (substr($ligne, $L - 1, 1) == "\n" or substr($ligne, $L - 1, 1) == "\r") {
48.      $ligne = substr($ligne, 0, $L - 1);
49.      $L--;
50.    }
51.    // fin
52.    return($ligne);
53.  }
54.
55.  // ------------------------------------------------------------------------
56.  function calculImpots($marié, $enfants, $salaire, $limites, $coeffR, $coeffN) {
57.    // $marié : oui, non
58.    // $enfants : nombre d'enfants
59.    // $salaire : salaire annuel
60.    // $limites, $coeffR, $coeffN : les tableaux des données permettant le calcul de l'impôt
61.
62.    // nombre de parts
63.    $marié = strtolower($marié);
64.    if ($marié == "oui")
65.      $nbParts = $enfants / 2 + 2;
66.    else
67.      $nbParts=$enfants / 2 + 1;
68.    // une 1/2 part de plus si au moins 3 enfants
69.    if ($enfants >= 3)
70.      $nbParts+=0.5;
71.    // revenu imposable
72.    $revenuImposable = 0.72 * $salaire;
73.    // quotient familial
74.    $quotient = $revenuImposable / $nbParts;
75.    // est mis à la fin du tableau limites pour arrêter la boucle qui suit
76.    array_push($limites, $quotient);
77.    // calcul de l'impôt
78.    $i = 0;
79.    while ($quotient > $limites[$i])
80.      $i++;
81.    // du fait qu'on a placé $quotient à la fin du tableau $limites, la boucle précédente
```

```
82.    // ne peut déborder du tableau $limites
83.    // maintenant on peut calculer l'impôt
84.    return floor($revenuImposable * $coeffR[$i] - $nbParts * $coeffN[$i]);
85.  }
86.  ?>
```

Le fichier des données *data.txt* (marié, enfants, salaire) :

```
oui,2,200000
non,2,200000
oui,3,200000
non,3,200000
oui,5,50000
non,0,3000000
```

Les fichier *résultats.txt* (marié, enfants, salaire, impôt) des résultats obtenus :

```
oui:2:200000:22504
non:2:200000:33388
oui:3:200000:16400
non:3:200000:22504
oui:5:50000:0
non:0:3000000:1354938
```

Commentaires

- ligne 4 : le nom du fichier texte contenant les données des contribuables (marié, enfants, salaire)
- ligne 5 : le nom du fichier texte contenant les résultats (marié, enfants, salaire, impôt) du calcul de l'impôt
- lignes 6-8 : les trois tableaux des données permettant le calcul de l'impôt
- lignes 11-15 : ouverture en lecture du fichier des données contribuables
- lignes 18-22 : ouverture en écriture du fichier des résultats
- ligne 25 : boucle de lecture des lignes (marié, enfants, salaire) du fichier des contribuables
- ligne 27 : la marque de fin de ligne est enlevée
- ligne 29 : les composantes (marié, enfants, salaire) de la ligne sont récupérées
- ligne 31 : l'impôt est calculé
- ligne 33 : l'impôt est mémorisé dans le fichier des résultats
- lignes 37-38 : une fois le fichier des contribuables exploité totalement, les fichiers sont fermés
- ligne 44 : la fonction qui supprime la marque de fin de ligne d'une ligne $ligne. La marque de fin de ligne est la chaîne "\r\n" sur les systèmes windows, "\n" sur les systèmes Unix. Le résultat est la chaîne d'entrée sans sa marque de fin de ligne.
- lignes 47-48 : *substr($chaîne,$début,$taille)* est la sous-chaîne de $*chaîne* commençant au caractère $*début* et ayant au plus $*taille* caractères.
- ligne 63 : *strtolower($chaîne)* rend $*chaîne* en minuscules
- ligne 76 : *array_push($tableau,$élément)* ajoute $*élément* à la fin du tableau $*tableau*. On notera que le tableau $*limites* est passé par valeur à la fonction *calculerImpot (*ligne 56*)*. Cet ajout d'élément se fait donc sur la copie du paramètre effectif, celui-ci restant inchangé.
- ligne 84 : la fonction *floor($x)* rend la valeur entière immédiatement inférieure à $x.

4.3 Version avec fichiers texte (impots_02)

Cette nouvelle version ne diffère de la précédente que par le fait que les données ($limites, $coeffR, $coeffN) nécessaires au calcul de l'impôt sont maintenant dans un fichier texte sous la forme suivante :

Fichier *impots.txt* :

```
12620:13190:15640:24740:31810:39970:48360:55790:92970:127860:151250:172040:195000:0
0:0.05:0.1:0.15:0.2:0.25:0.3:0.35:0.4:0.45:0.5:0.55:0.6:0.65
0:631:1290.5:2072.5:3309.5:4900:6898.5:9316.5:12106:16754.5:23147.5:30710:39312:49062
```

La nouvelle version se contente de lire les données du fichier [impots.txt] afin de les mettre dans trois tableaux *($limites, $coeff*R*, $coeff*N*)*. On est alors ramené au cas précédent. Nous ne présentons que les changements :

```
1.  <?php
2.
3.  // définition des constantes
4.  $DATA = "data.txt";
```

```
5.  $RESULTATS = "resultats.txt";
6.  $IMPOTS = "impots.txt";
7.
8.  // les données nécessaires au calcul de l'impôt ont été placées dans le fichier $IMPOTS
9.  // à raison d'une ligne par tableau sous la forme
10. // val1:val2:val3,...
11. list($erreur, $limites, $coeffR, $coeffN) = getTables($IMPOTS);
12.
13. // y-a-t-il eu une erreur ?
14. if ($erreur) {
15.   print "$erreur\n";
16.   exit;
17. }//if
18. // lecture des données utilisateur
19. ...
20.
21. // fin
22. exit;
23.
24. // ----------------------------------------------------------------------
25. function cutNewLinechar($ligne) {
26.   // on supprime la marque de fin de ligne de $ligne si elle existe
27. ...
28. }
29.
30. // ----------------------------------------------------------------------
31. function calculImpots($marié, $enfants, $salaire, $limites, $coeffR, $coeffN) {
32.   // $marié : oui, non
33.   // $enfants : nombre d'enfants
34.   // $salaire : salaire annuel
35.   // $limites, $coeffR, $coeffN : les tableaux des données permettant le calcul de l'impôt
36.   ...
37. }
38.
39. // ----------------------------------------------------------------------
40. function getTables($IMPOTS) {
41.   // $IMPOTS : le nom du fichier contenant les données des tables $limites, $coeffR, $coeffN
42.   // le fichier $IMPOTS existe-t-il ?
43.   if (!file_exists($IMPOTS))
44.     return array("Le fichier $IMPOTS n'existe pas","","","");
45.   // lecture en bloc du fichier
46.   $tables = file($IMPOTS);
47.   if (!$tables)
48.     return array("Erreur lors de l'exploitation du fichier $IMPOTS","","","");
49.   // création des 3 tables - on suppose que les lignes sont syntaxiquement correctes
50.   $limites = explode(":", cutNewLineChar($tables[0]));
51.   $coeffR = explode(":", cutNewLineChar($tables[1]));
52.   $coeffN = explode(":", cutNewLineChar($tables[2]));
53.   // fin
54.   return array("", $limites, $coeffR, $coeffN);
55. }
56. ?>
```

Commentaires

- ligne 11 : la fonction *getTables($IMPOTS)* exploite le fichier texte $*IMPOTS* pour en extraire les trois tableaux (*$limites, $coeffR, $coeffN*). Le résultat rendu est ($*erreur*, *$limites, $coeffR, $coeffN*) où $*erreur* est un message d'erreur, vide s'il n'y a pas eu d'erreur.
- ligne 40 : la fonction *getTables($IMPOTS)*.
- ligne 43 : la fonction *files_exists(nom_fichier)* rend le booléen *true* si le fichier *nom_fichier* existe, le booléen *false* sinon.
- ligne 46 : la fonction *file(nom_fichier)* lit tout le fichier texte *nom_fichier*. Elle rend un tableau où chaque élément est une ligne du fichier texte. Comme ici, le fichier *impots.txt* a trois lignes de texte, la fonction va rendre un tableau à trois éléments.
- lignes 50-52 : exploitation des trois lignes récupérées pour créer les trois tableaux (*$limites, $coeffR, $coeffN*)

Les résultats obtenus sont les mêmes qu'obtenus dans la version précédente.

5 Les objets

5.1 Toute variable peut devenir un objet doté d'attributs (exemple_14)

```
1.  <?php
2.
3.  // toute variable peut avoir des attributs par construction
4.  $obj1->attr1 = "un";
5.  $obj1->attr2 = 100;
6.  // affiche l'objet
7.  print "objet1=[$obj1->attr1,$obj1->attr2]\n";
8.  // modifie l'objet
9.  $obj1->attr2+=100;
10. // affiche l'objet
11. print "objet1=[$obj1->attr1,$obj1->attr2]\n";
12. // affecte la valeur de objet1 à objet2
13. $obj2 = $obj1;
14. // modifie obj2
15. $obj2->attr2 = 0;
16. // affiche les deux objets
17. print "objet1=[$obj1->attr1,$obj1->attr2]\n";
18. print "objet2=[$obj2->attr1,$obj2->attr2]\n";
19. // affecte la référence de objet1 à objet3
20. $obj3 = &$obj1;
21. // modifie obj3
22. $obj3->attr2 = 10;
23. // affiche les deux objets
24. print "objet1=[$obj1->attr1,$obj1->attr2]\n";
25. print "objet3=[$obj3->attr1,$obj3->attr2]\n";
26. // un objet est-il un dictionnaire ?
27. print count($obj1)."\n";
28. while (list($attribut, $valeur) = each($obj1))
29.   print "obj1[$attribut]=$valeur\n";
30. // fin
31. exit;
32. ?>
```

Résultats :

```
objet1=[un,100]
objet1=[un,200]
objet1=[un,0]
objet2=[un,0]
objet1=[un,10]
objet3=[un,10]
1
obj1[attr1]=un
obj1[attr2]=10
```

Commentaires

- ligne 4 : la notation *$obj->attr* désigne l'attribut *attr* de la variable $*obj*. S'il n'existe pas il est créé, faisant ainsi de la variable $*obj*, un objet avec attributs.
- ligne 13 : l'expression *$obj2=$obj1*, lorsque $*obj1* est un objet, est une copie d'objets par **référence**. Ainsi $*obj2* et $*obj1* sont des références sur un même objet. L'objet lui-même peut être modifié par l'une ou l'autre des références.
- ligne 20 : l'expression *$obj3=&$obj1* affecte à $*obj3* la référence (l'adresse) de la variable $*obj1*. Si $*obj1* est un objet comme c'est le cas ici, l'expression précédente est équivalente à *$obj3=$obj1*.
- lignes 28-29 : montrent qu'un objet peut être parcouru comme un dictionnaire. Les clés du dictionnaire sont les noms des attributs et les valeurs du dictionnaire, les valeurs de ces mêmes attributs.
- ligne 27 : la fonction *count* peut être appliquée à un objet mais ne donne pas, comme on aurait pu s'y attendre, le nombre d'attributs. Donc un objet présente des similitudes avec un dictionnaire mais n'en est pas un.

5.2 Une classe Personne sans attributs déclarés (exemple_15)

```
1.  <?php
2.
3.  class Personne {
4.
5.    // attributs de la classe
6.    // non déclarés - peuvent être créés dynamiquement
7.
8.    // méthode
9.    function identité() {
10.     // à priori, utilise des attributs inexistants
11.     return "[$this->prénom,$this->nom,$this->âge]";
12.   }
13. }
14.
15. // test
16. // les attributs sont publics et peuvent être créés dynamiquement
17. $p = new Personne();
18. $p->prénom = "Paul";
19. $p->nom = "Langevin";
20. $p->âge = 48;
21. // appel d'une méthode
22. print "personne=" . $p->identité() . "\n";
23. // fin
24. exit;
25. ?>
```

Résultats :

```
personne=[Paul,Langevin,48]
```

Commentaires

- lignes 3-13 : définissent une **classe** *Personne*. Une classe est un moule à partir duquel on crée des objets. Une classe est un ensemble regroupant des **attributs** et des fonctions appelées **méthodes**. Il n'y a pas obligation à déclarer les attributs.
- lignes 9-12 : la méthode *identité* affiche la valeur de trois attributs non déclarés dans la classe. Le mot clé $**this** désigne l'objet auquel on applique la méthode.
- ligne 17 : on crée un objet $*p* de type *Personne*. Le mot clé **new** sert à créer un nouvel objet. L'opération rend une **référence** sur l'objet créé (donc une adresse). Diverses écritures sont possibles : *new Personne()*, *new Personne*, *new personne*. Le nom de la classe est insensible à la casse.
- lignes 18-20 : les trois attributs nécessaires à la méthode *identité* sont créés dans l'objet $p.
- ligne 22 : la méthode *identité* de la classe *Personne* est appliquée sur l'objet $*p*. Dans le code (lignes 9-11) de la méthode *identité*, $*this* référence le même objet que $*p*.

5.3 La classe Personne avec attributs déclarés (exemple_16)

```
1.  <?php
2.
3.  class Personne {
4.
5.    // attributs de la classe
6.    var $prénom;
7.    var $nom;
8.    var $âge;
9.
10.   // méthode
11.   function identité() {
12.     return "[$this->prénom,$this->nom,$this->âge]";
13.   }
14. }
15. // test
16. // les attributs sont publics
17. $p = new Personne();
```

```
18. $p->prénom = "Paul";
19. $p->nom = "Langevin";
20. $p->âge = 48;
21. // appel d'une méthode
22. print "personne=" . $p->identité() . "\n";
23. // fin
24. exit;
25. ?>
```

Résultats :

```
personne=[Paul,Langevin,48]
```

Commentaires

- lignes 6-8 : les attributs de la classe sont explicitement déclarés avec le mot clé **var**.

5.4 La classe Personne avec un constructeur (exemple_17)

Les exemples précédents montraient des classes *Personne* exotiques telles qu'on pouvait les trouver dans PHP 4. Il n'est pas conseillé de suivre ces exemples. Nous présentons maintenant une classe *Personne* correspondant aux bonnes pratiques de PHP 5 :

```
1.  <?php
2.
3.  class Personne {
4.
5.  // attributs de la classe
6.    private $prénom;
7.    private $nom;
8.    private $âge;
9.
10. // getters and setters
11.   public function getPrénom() {
12.     return $this->prénom;
13.   }
14.
15.   public function getNom() {
16.     return $this->nom;
17.   }
18.
19.   public function getAge() {
20.     return $this->âge;
21.   }
22.
23.   public function setPrénom($prénom) {
24.     $this->prénom = $prénom;
25.   }
26.
27.   public function setNom($nom) {
28.     $this->nom = $nom;
29.   }
30.
31.   public function setAge($age) {
32.     $this->âge = $age;
33.   }
34.
35. // constructeur
36.   function __construct($prénom, $nom, $âge) {
37.     // on passe par les set
38.     $this->setPrénom($prénom);
39.     $this->setNom($nom);
40.     $this->setAge($âge);
41.   }
42.
43. // méthode toString
```

```
44.   function __toString() {
45.     return "[$this->prénom,$this->nom,$this->âge]";
46.   }
47.
48. }
49.
50. // test
51. // création d'un objet personne
52. $p = new Personne("Paul", "Langevin", 48);
53. // identité de cette personne
54. print "personne=$p\n";
55. // on change l'âge
56. $p->setAge(14);
57. // identité de la personne
58. print "personne=$p\n";
59. // fin
60. exit;
61. ?>
```

Résultats :

```
personne=[Paul,Langevin,48]
personne=[Paul,Langevin,14]
```

Commentaires

- lignes 3-48 : la classe *Personne*
- lignes 6-8 : les attributs privés (**private**) de la classe. Ces attributs ne sont visibles qu'à l'intérieur de la classe. Les autres mots clés utilisables sont :
  - **public** : fait de l'attribut un attribut public visible de l'extérieur de la classe
  - **protected** : fait de l'attribut un attribut protégé visible de l'intérieur de la classe et des classes dérivées de celle-ci.
- parce que les attributs sont privés, on ne peut y accéder de l'extérieur de la classe. On ne peut donc écrire le code suivant :

```
1. $p=new Personne() ;
2. $p->nom="Landru" ;
```

Ici, on est en-dehors de la classe *Personne*. Comme l'attribut *nom* est privé, la ligne 2 est incorrecte. Pour initialiser les champs privés de l'objet $p, il y a deux moyens :

- utiliser les méthodes publiques **set** et **get** (le nom de ces méthodes peut être quelconque) des lignes 11-33. On pourra alors écrire :

```
1.  $p=new Personne() ;
2.  $p->setNom("Landru") ;
```

- utiliser le constructeur des lignes 36-41. On écrira alors :

```
1. $p=new Personne("Michel","Landru",44) ;
```

L'écriture ci-dessus, appelle automatiquement la méthode de la classe *Personne* appelée *__construct*.

- ligne 54 : cette ligne affiche la personne $*p* sous la forme d'une chaîne de caractères. Pour ce faire, la méthode de la classe *Personne* appelée *__toString* est utilisée.

5.5 La classe Personne avec contrôle de validité dans le constructeur (exemple_18)

Le constructeur d'une classe est le bon endroit pour vérifier que les valeurs d'initialisation de l'objet sont corrects. Mais un objet peut être également initialisé par ses méthodes *set* ou équivalentes. Pour éviter de mettre à deux endroits différents les mêmes vérifications, on mettra ces dernières dans les méthodes *set*. Si une valeur d'initialisation d'un objet se révèle incorrecte, on lancera une **exception**. Voici un exemple :

```
1.  <?php
2.
3.  class Personne {
4.
5.  // attributs de la classe
6.    private $prénom;
```

```
7.    private $nom;
8.    private $âge;
9.
10.  // getters et setters
11.   public function getPrénom() {
12.     return $this->prénom;
13.   }
14.
15.   public function getNom() {
16.     return $this->nom;
17.   }
18.
19.   public function getAge() {
20.     return $this->âge;
21.   }
22.
23.   public function setPrénom($prénom) {
24.     // le prénom doit être non vide
25.     if (!preg_match("/^\s*(.+?)\s*$/", $prénom, $champs)) {
26.       throw new Exception("Le prénom doit être non vide");
27.     } else {
28.       $this->prénom = $champs[1];
29.     }
30.   }
31.
32.   public function setNom($nom) {
33.     // le nom doit être non vide
34.     if (!preg_match("/^\s*(.+?)\s*$/", $nom, $champs)) {
35.       throw new Exception("Le nom doit être non vide");
36.     } else {
37.       $this->nom = $champs[1];
38.     }
39.   }
40.
41.   public function setAge($âge) {
42.     // l'âge doit être valide
43.     if (!preg_match("/^\s*(\d+)\s*$/", $âge, $champs)) {
44.       throw new Exception("L'âge doit être un entier positif ou nul");
45.     } else {
46.       $this->âge = $champs[1];
47.     }
48.   }
49.
50.  // constructeur
51.   function __construct($prénom, $nom, $âge) {
52.     // on passe par les set
53.     $this->setPrénom($prénom);
54.     $this->setNom($nom);
55.     $this->setAge($âge);
56.   }
57.
58.   // méthode toString
59.   function __toString() {
60.     return "[$this->prénom,$this->nom,$this->âge]";
61.   }
62. }
63.
64. // test
65. // création d'un objet Personne
66. $p = new Personne("Paul", "Langevin", 48);
67. // identité de cette personne
68. print "personne=$p\n";
69. // création d'un objet Personne erroné
70. try {
71.   $p = new Personne("xx", "yy", "zz");
72. } catch (Exception $e) {
```

```
73.   print $e->getMessage();
74. }
75. // fin
76. exit;
77. ?>
```

Résultats :

```
personne=[Paul,Langevin,48]
L'âge doit être un entier positif ou nul
```

Commentaires

- lignes 23-30 : initialisation de l'attribut *prénom* et vérification de la valeur d'initialisation
- ligne 25 : utilisation d'une expression régulière. Le prénom est une suite d'un ou plusieurs caractères éventuellement suivi ou précédé d'espaces.
- ligne 26 : si le nom est vide, on lance une exception sinon le prénom est mémorisé (ligne 28)
- ligne 66 : utilisation du constructeur de la classe *Personne*
- ligne 68 : utilisation de la méthode *__toString* de la classe *Personne*
- lignes 70-74 : gestion de l'éventuelle exception lancée par le constructeur de la classe *Personne*.

5.6 Ajout d'une méthode faisant office de second constructeur (exemple_19)

En PHP 5, il n'est pas possible d'avoir plusieurs constructeurs avec des paramètres différents qui permettraient de construire un objet de diverses façons. On peut alors utiliser des méthodes faisant office de constructeur :

```
1.  <?php
2.
3.  class Personne {
4.
5.  // attributs de la classe
6.    private $prénom;
7.    private $nom;
8.    private $âge;
9.
10. // getters et setters
11.   public function getPrénom() {
12.     return $this->prénom;
13.   }
14.
15. ….
16.
17. // constructeur
18.   function __construct($prénom, $nom, $âge) {
19.     // on passe par les set
20.     $this->setPrénom($prénom);
21.     $this->setNom($nom);
22.     $this->setAge($âge);
23.   }
24.
25.   // méthode
26.   public function initWithPersonne($p) {
27.     // initialise l'objet courant avec une personne $p
28.     $this->__construct($p->prénom, $p->nom, $p->âge);
29.   }
30.
31.   // méthode toString
32.   function __toString() {
33.     return "[$this->prénom,$this->nom,$this->âge]";
34.   }
35.
36. }
37.
38. // test
39. // création d'un objet Personne
```

```
40. $p = new Personne("Paul", "Langevin", 48);
41. // identité de cette personne
42. print "personne=$p\n";
43. // création d'une seconde personne
44. $p2 = new Personne("Laure", "Adeline", 67);
45. // initialisation de la première avec les valeurs de la seconde
46. $p->initWithPersonne($p2);
47. // vérification
48. print "personne=$p\n";
49. // fin
50. exit;
51. ?>
```

Résultats :

```
personne=[Paul,Langevin,48]
personne=[Laure,Adeline,67]
```

Commentaires

- lignes 26-29 : la méthode *initWithPersonne* permet d'affecter à l'objet courant les valeurs des attributs d'un autre objet *Personne*. Ici, elle fait appel au constructeur *__construct* mais ce n'est pas obligatoire.

5.7 Un tableau d'objets Personne (exemple_20)

L'exemple suivant montre qu'on peut avoir des tableaux d'objets.

```
1.  <?php
2.
3.  class Personne {
4.
5.  // attributs de la classe
6.    private $prénom;
7.    private $nom;
8.    private $âge;
9.
10. // getters et setters
11.   public function getPrénom() {
12.     return $this->prénom;
13.   }
14.
15. ...
16.
17. // constructeur
18.   function __construct($prénom, $nom, $âge) {
19.     // on passe par les set
20.     $this->setPrénom($prénom);
21.     $this->setNom($nom);
22.     $this->setAge($âge);
23.   }
24.
25.   // méthode
26.   public function initWithPersonne($p) {
27. ….
28.   }
29.
30.   // méthode toString
31.   function __toString() {
32.     ...
33.   }
34.
35. }
36.
37. // test
38. // création d'un tableau d'objets Personne
39. $groupe = array(new Personne("Paul", "Langevin", 48), new Personne("Sylvie", "Lefur", 70));
```

```
40.
41. // identité de ces personnes
42. for ($i = 0; $i < count($groupe); $i++)
43.   print "groupe[$i]=$groupe[$i]\n";
44. // fin
45. exit;
46. ?>
```

Résultats :

```
groupe[0]=[Paul,Langevin,48]
groupe[1]=[Sylvie,Lefur,70]
```

Commentaires

- ligne 39 : création d'un tableau de 2 personnes
- ligne 42 : parcours du tableau
- ligne 43 : *$groupe[$i]* est un objet de type *Personne*. La méthode *__toString* est utilisée pour l'afficher.

5.8 Création d'une classe dérivée de la classe Personne (exemple_21)

```
1.  <?php
2.
3.  class Personne {
4.
5.    // attributs de la classe
6.    private $prénom;
7.    private $nom;
8.    private $âge;
9.
10.   // getters et setters
11.   public function getPrénom() {
12.     return $this->prénom;
13.   }
14. ...
15.
16.   // constructeur
17.   function __construct($prénom, $nom, $âge) {
18.     // on passe par les set
19.     $this->setPrénom($prénom);
20.     $this->setNom($nom);
21.     $this->setAge($âge);
22.   }
23.
24.   // méthode
25.   public function initWithPersonne($p) {
26.     // initialise l'objet courant avec une personne $p
27.     $this->__construct($p->prénom, $p->nom, $p->âge);
28.   }
29.
30.   // méthode toString
31.   function __toString() {
32.     return "[$this->prénom,$this->nom,$this->âge]";
33.   }
34.
35. }
36.
37. // une classe dérivée de Personne
38. class Enseignant extends Personne {
39.
40.   // attributs
41.   private $discipline;   // discipline enseignée
42.
43.   // getter et setter
44.
```

```
45.   public function getDiscipline() {
46.     return $this->discipline;
47.   }
48.
49.   public function setDiscipline($discipline) {
50.     $this->discipline = $discipline;
51.   }
52.
53.   // constructeur
54.
55.   public function __construct($prénom, $nom, $âge, $discipline) {
56.     // attributs parent
57.     parent::__construct($prénom, $nom, $âge);
58.     // autres attributs
59.     $this->setDiscipline($discipline);
60.   }
61.
62.   // surcharge de la fonction __toString de la classe parente
63.   public function __toString() {
64.     return "[" . parent::__toString() . ",$this->discipline]";
65.   }
66.
67. }
68.
69. // test
70. // création d'un tableau d'objets Personne et dérivés
71. $groupe = array(new Enseignant("Paul", "Langevin", 48, "anglais"), new Personne("Sylvie", "Lefur", 70));
72.
73. // identité de ces personnes
74. for ($i = 0; $i < count($groupe); $i++)
75.   print "groupe[$i]=$groupe[$i]\n";
76. // fin
77. exit;
78. ?>
```

Résultats :

```
groupe[0]=[[Paul,Langevin,48],anglais]
groupe[1]=[Sylvie,Lefur,70]
```

Commentaires

- lignes 3-35 : la classe *Personne*
- ligne 38 : la classe *Enseignant* dérive (**extends**) de la classe *Personne*. La classe dérivée *Enseignant* hérite des attributs et des méthodes de sa classe mère.
- ligne 41 : la classe *Enseignant* ajoute un nouvel attribut *discipline* qui lui est propre
- ligne 55 : le constructeur de la classe *Enseignant* reçoit 4 paramètres
  - 3 pour initialiser sa classe parent (prénom, nom, âge)
  - 1 pour sa propre initialisation (discipline)
- ligne 57 : la classe dérivée a accès aux méthodes et constructeurs de sa classe parent via le mot clé **parent ::**. Ici on passe les paramètres (prénom, nom, âge) au constructeur de la classe parent.
- ligne 64 : la méthode *__toString* de la classe dérivée utilise la méthode *__toString* de la classe parent.
- ligne 71 : un tableau contenant un objet *Enseignant* et un objet *Personne*.
- ligne 74 : on boucle sur les éléments du tableau
- ligne 75 : la méthode *__toString* de chaque élément *$groupe[$i]* va être appelée. La classe *Personne* a une méthode *__toString*. La classe *Enseignant* en a deux : celle de sa classe parent et la sienne propre. On peut se demander laquelle va être appelée. L'exécution montre que c'est celle de la classe *Enseignant* qui a été appelée. C'est toujours ainsi : lorsqu'une méthode est appelée sur un objet, on cherche celle-ci dans l'ordre suivant : dans l'objet lui-même, dans sa classe parent s'il en a une, puis dans la classe parent de la classe parent, etc … La recherche s'arrête dès que la méthode a été trouvée.

5.9 Création d'une seconde classe dérivée de la classe Personne (exemple_22)

L'exemple suivant crée une classe *Etudiant* dérivée de la classe *Personne* :

```php
<?php

class Personne {

// attributs de la classe
 private $prénom;
 private $nom;
 private $âge;

// getters et setters
 public function getPrénom() {
  return $this->prénom;
 }
...

// constructeur
 function __construct($prénom, $nom, $âge) {
  // on passe par les set
  $this->setPrénom($prénom);
  $this->setNom($nom);
  $this->setAge($âge);
 }

 // méthode
 public function initWithPersonne($p) {
  // initialise l'objet courant avec une personne $p
  ...
 }

 // méthode toString
 function __toString() {
  return "[$this->prénom,$this->nom,$this->âge]";
 }

}

// une classe dérivée de personne
class Enseignant extends Personne {

 // attributs
 private $discipline;  // discipline enseignée

 // getter et setter
...

 // constructeur

 public function __construct($prénom, $nom, $âge, $discipline) {
  // attributs parent
  parent::__construct($prénom, $nom, $âge);
  // autres attributs
  $this->setDiscipline($discipline);
 }

 // surcharge de la fonction _identité__toString de la classe parente
 public function __toString() {
  return "[" . parent::__toString() . ",$this->discipline]";
 }

}

// une classe dérivée de Personne
class Etudiant extends Personne {

 // attributs
 private $formation;  // discipline enseignée
```

```
67.
68.   // getter et setter
69.
70.   public function getFormation() {
71.     return $this->formation;
72.   }
73.
74.   public function setFormation($formation) {
75.     $this->formation = $formation;
76.   }
77.
78.   // constructeur
79.
80.   public function __construct($prénom, $nom, $âge, $formation) {
81.     // attributs parent
82.     parent::__construct($prénom, $nom, $âge);
83.     // autres attributs
84.     $this->setFormation($formation);
85.   }
86.
87.   // surcharge de la fonction __toString de la classe parente
88.   public function __toString() {
89.     return "[" . parent::__toString() . ",$this->formation]";
90.   }
91.
92. }
93.
94. // test
95. // création d'un tableau d'objets personne et dérivés
96. $groupe = array(new Enseignant("Paul", "Langevin", 48, "anglais"), new Personne("Sylvie", "Lefur", 70), new
    Etudiant("Steve", "Boer", 23, "iup2 qualité"));
97.
98. // identité de ces personnes
99. for ($i = 0; $i < count($groupe); $i++)
100.   print "groupe[$i]=$groupe[$i]\n";
101. // fin
102. exit;
103. ?>
```

Résultats :

```
groupe[0]=[[Paul,Langevin,48],anglais]
groupe[1]=[Sylvie,Lefur,70]
groupe[2]=[[Steve,Boer,23],iup2 qualité]
```

5.10 Les interfaces (exemple_23)

Une interface est une structure définissant des prototypes de méthodes. Une classe implémentant une interface définit le code de toutes les méthodes de l'interface.

```
1.  <?php
2.
3.  // une interface
4.  interface IExemple {
5.    function ajouter($i, $j);
6.
7.    function soustraire($i, $j);
8.  }
9.
10. // implémentation 1 de l'interface
11. class Classe1 implements IExemple {
12.
13.   public function ajouter($a, $b) {
14.     return $a + $b + 10;
15.   }
```

```
16.
17.   public function soustraire($a, $b) {
18.     return $a - $b + 20;
19.   }
20.
21. }
22.
23. // implémentation 2 de l'interface
24. class Classe2 implements IExemple {
25.
26.   public function ajouter($a, $b) {
27.     return $a + $b + 100;
28.   }
29.
30.   public function soustraire($a, $b) {
31.     return $a - $b + 200;
32.   }
33.
34. }
35.
36. // fonction
37. function calculer($a, $b, IExemple $interface) {
38.   print $interface->ajouter($a, $b)."\n";
39.   print $interface->soustraire($a, $b)."\n";
40. }
41.
42. // ------------------- main
43. // création de 2 objets de type Classe1 et Classe2
44. $c1 = new Classe1();
45. $c2 = new Classe2();
46. // appel de la fonction calculer
47. calculer(4, 3, $c1);
48. calculer(14, 13, $c2);
49. ?>
```

Résultats

```
17
21
127
201
```

Commentaires

- lignes 4-8 : l'interface *IExemple* définit deux méthodes : la fonction *ajouter* (ligne 5) et la fonction *soustraire* (ligne 7). L'interface ne définit pas le code de ces méthodes. Ce sont les classes implémentant l'interface qui vont le faire.
- ligne 11 : la classe *Classe1* implémente (**implements**) l'interface *IExemple*. Elle définit donc un code pour les méthodes *ajouter* (ligne 13) et *soustraire* (ligne 17).
- lignes 24-34 : idem pour la classe *Classe2*
- ligne 37 : la méthode *calculer* admet trois paramètres : deux entiers $*a* et $*b* et un objet $*interface* implémentant l'interface *IExemple*. On aurait pu écrire *function calculer($a,$b,$interface)* sans indiquer le type *IExemple* du paramètre $*interface*. En l'indiquant, on permet à l'interpréteur PHP de vérifier que le paramètre effectif est bien un objet implémentant l'interface *IExemple*.
- lignes 38-39 : utilisation des méthodes *ajouter* et *soustraire* de l'objet $interface. Parce que celui-ci implémente l'interface *IExemple*, on sait qu'il possède ces deux méthodes.
- lignes 44-45 : création de deux objets de types différents mais implémentant tous deux l'interface *IExemple*.
- ligne 47 : on passe à la fonction *calculer*, l'objet $*c1* de type *Classe1* implémentant l'interface *IExemple*.
- ligne 48 : on passe à la fonction *calculer*, l'objet $*c2* de type *Classe2* implémentant l'interface *IExemple*.

6 Exercice d'application - IMPOTS avec objets (impots_03)

On reprend l'exercice déjà étudié précédemment (pages 24 et 26) pour le résoudre avec un code PHP utilisant une classe.

```php
<?php

// définition d'une classe Impôts
class Impôts {

 // attributs : les 3 tableaux de données
  private $limites;
  private $coeffR;
  private $coeffN;

  // constructeur

  public function __construct($IMPOTS) {
    // $IMPOTS : le nom du fichier contenant les données des tables $limites, $coeffR, $coeffN
    // le fichier $IMPOTS existe-t-il ?
    if (!file_exists($IMPOTS)) {
      throw new Exception("Le fichier $IMPOTS n'existe pas");
    }//if
    // lecture en bloc du fichier
    $tables = file($IMPOTS);
    if (!$tables) {
      throw new Exception("Erreur lors de l'exploitation du fichier $IMPOTS");
    }//if
    // création des 3 tables - on suppose que les lignes sont syntaxiquement correctes
    $u = new Utilitaires(); // pour disposer de fonctions utilitaires
    $this->limites = explode(":", $u->cutNewLineChar($tables[0]));
    $this->coeffR = explode(":", $u->cutNewLineChar($tables[1]));
    $this->coeffN = explode(":", $u->cutNewLineChar($tables[2]));
    }

  // ------------------------------------------------------------------
  public function calculer($marié, $enfants, $salaire) {
    // $marié : oui, non
    // $enfants : nombre d'enfants
    // $salaire : salaire annuel

    // nombre de parts
    $marié = strtolower($marié);
    if ($marié == "oui")
      $nbParts = $enfants / 2 + 2;
    else
      $nbParts=$enfants / 2 + 1;
    // une 1/2 part de plus si au moins 3 enfants
    if ($enfants >= 3)
      $nbParts+=0.5;
    // revenu imposable
    $revenuImposable = 0.72 * $salaire;
    // quotient familial
    $quotient = $revenuImposable / $nbParts;
    // est mis à la fin du tableau limites pour arrêter la boucle qui suit
    $this->limites[count($this->limites) - 1] = $quotient;
    // calcul de l'impôt
    $i = 0;
    while ($quotient > $this->limites[$i])
      $i++;
    // du fait qu'on a placé $quotient à la fin du tableau $limites, la boucle précédente
    // ne peut déborder du tableau $limites
    // maintenant on peut calculer l'impôt
    return floor($revenuImposable * $this->coeffR[$i] - $nbParts * $this->coeffN[$i]);
```

```
60.    }
61.
62.  }
63.
64.  // une classe de fonctions utilitaires
65.  class Utilitaires {
66.
67.    public function cutNewLinechar($ligne) {
68.      // on supprime la marque de fin de ligne de $ligne si elle existe
69.      $L = strlen($ligne);  // longueur ligne
70.      while (substr($ligne, $L - 1, 1) == "\n" or substr($ligne, $L - 1, 1) == "\r") {
71.        $ligne = substr($ligne, 0, $L - 1);
72.        $L--;
73.      }//while
74.      // fin
75.      return($ligne);
76.    }
77.
78.  }
79.
80.  // test ----------------------------------------------------
81.  // définition des constantes
82.  $DATA = "data.txt";
83.  $RESULTATS = "resultats.txt";
84.  $IMPOTS = "impots.txt";
85.  // les données nécessaires au calcul de l'impôt ont été placées dans le fichier IMPOTS
86.  // à raison d'une ligne par tableau sous la forme
87.  // "val1":"val2":"val3",...
88.  // on crée un objet Impôts
89.  try {
90.    $I = new Impôts($IMPOTS);
91.  } catch (Exception $e) {
92.    print $e->getMessage();
93.    exit;
94.  }
95.  // on crée un objet utilitaires
96.  $u = new Utilitaires();
97.
98.  // lecture des données utilisateur
99.  $data = fopen($DATA, "r");
100. if (!$data) {
101.   print "Impossible d'ouvrir en lecture le fichier des données [$DATA]\n";
102.   exit;
103. }
104.
105. // ouverture fichier des résultats
106. $résultats = fopen($RESULTATS, "w");
107. if (!$résultats) {
108.   print "Impossible de créer le fichier des résultats [$RESULTATS]\n";
109.   exit;
110. }
111.
112. // on exploite la ligne courante du fichier des données
113. while ($ligne = fgets($data, 100)) {
114.   // on enlève l'éventuelle marque de fin de ligne
115.   $ligne = $u->cutNewLineChar($ligne);
116.   // on récupère les 3 champs marié,enfants,salaire qui forment $ligne
117.   list($marié, $enfants, $salaire) = explode(",", $ligne);
118.   // on calcule l'impôt
119.   $impôt = $I->calculer($marié, $enfants, $salaire);
120.   // on inscrit le résultat
121.   fputs($résultats, "$marié:$enfants:$salaire:$impôt\n");
122.   // donnée suivante
123. }// while
124. // on ferme les fichiers
```

```
125. fclose($data);
126. fclose($résultats);
127.
128. // fin
129. print "Terminé\n";
130. exit;
131. ?>
```

Le fichier [impots.txt] :

```
12620:13190:15640:24740:31810:39970:48360:55790:92970:127860:151250:172040:195000:0
0:0.05:0.1:0.15:0.2:0.25:0.3:0.35:0.4:0.45:0.5:0.55:0.6:0.65
0:631:1290.5:2072.5:3309.5:4900:6898.5:9316.5:12106:16754.5:23147.5:30710:39312:49062
```

Le fichier [data.txt]

```
oui,2,200000
non,2,200000
oui,3,200000
non,3,200000
oui,5,50000
non,0,3000000
```

Résultats : ceux déjà obtenus dans les versions avec tableaux et fichiers.

Le fichier [resultats.txt]

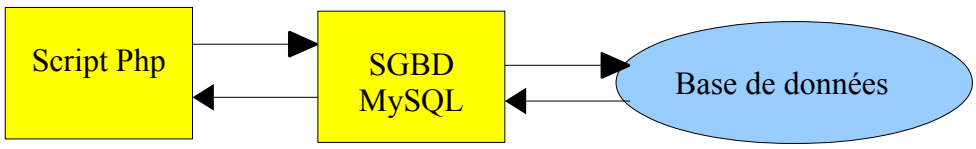
```
oui:2:200000:22504
non:2:200000:33388
oui:3:200000:16400
non:3:200000:22504
oui:5:50000:0
non:0:3000000:1354938
```

Commentaires

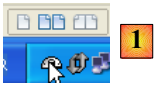
- ligne 4 : la classe **Impôts**. Elle encapsule des données et des méthodes permettant le calcul de l'impôt :
  - **données** : lignes 7-9 – ce sont les trois tableaux de données permettant le calcul de l'impôt. Ces trois tableaux sont initialisés par le constructeur de la ligne 13 à partir d'un fichier texte.
  - **méthodes** : ligne 32 – la méthode *calculer* permet de calculer l'impôt.
- lignes 17, 22 : lorsque le constructeur ne peut pas construire l'objet *Impôts*, il lance une exception.
- ligne 25 : la classe *Utilitaires* a été définie pour encapsuler la fonction *cutNewLineChar*.
- lignes 26-28 : initialisation des trois champs privés de la classe *Impôts*
- ligne 32 : la méthode *calculer* de la classe *Impôts* permet de calculer l'impôt d'un contribuable.
- lignes 65-78 : la classe *Utilitaires*
- lignes 89-94 : construction d'un objet *Impôts* avec gestion de l'exception éventuelle. Cette construction va exploiter le fichier texte $IMPOTS pour mettre son contenu dans les trois champs privés de l'objet *Impôts*.
- lignes 96 : construction d'un objet *Utilitaires* pour disposer de la fonction *cutNewLineChar*.

7 Utilisation du Sgbd MySql

Nous allons écrire des scripts PHP utilisant une base de données MySQL :



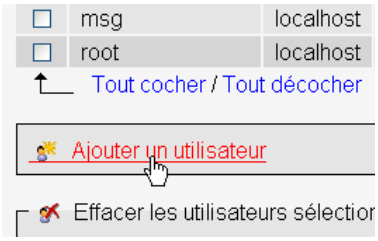
Le Sgbd MySQL est inclus dans le paquetage WampServer (cf page 5). Nous montrons comment créer une base de données ainsi qu'un utilisateur MySQL.

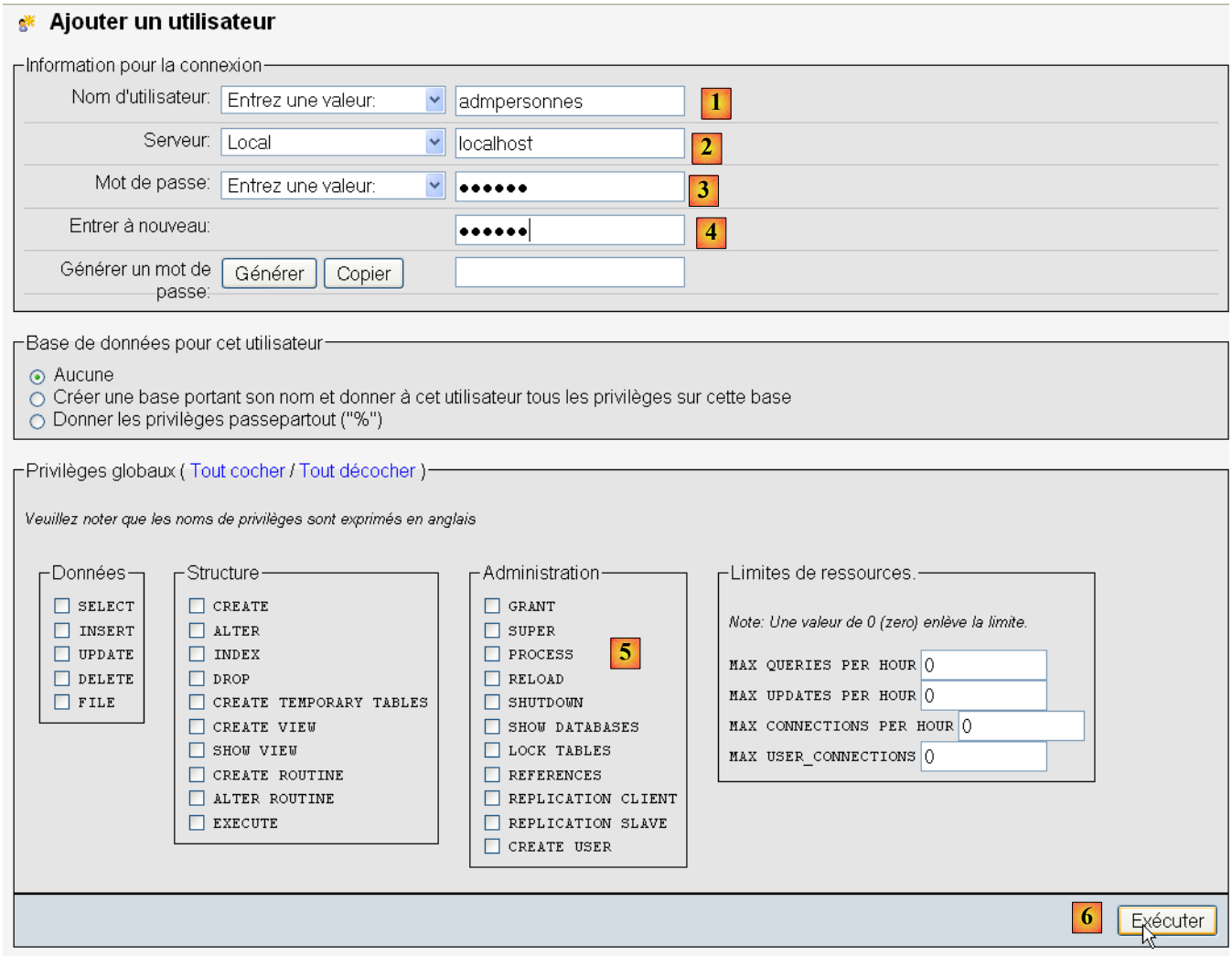


- une fois lancé, WampServer peut être administré à partir d'une icône [1] placée en bas à droite de la barre des tâches.
- en [2], on lance l'outil d'administration de MySQL

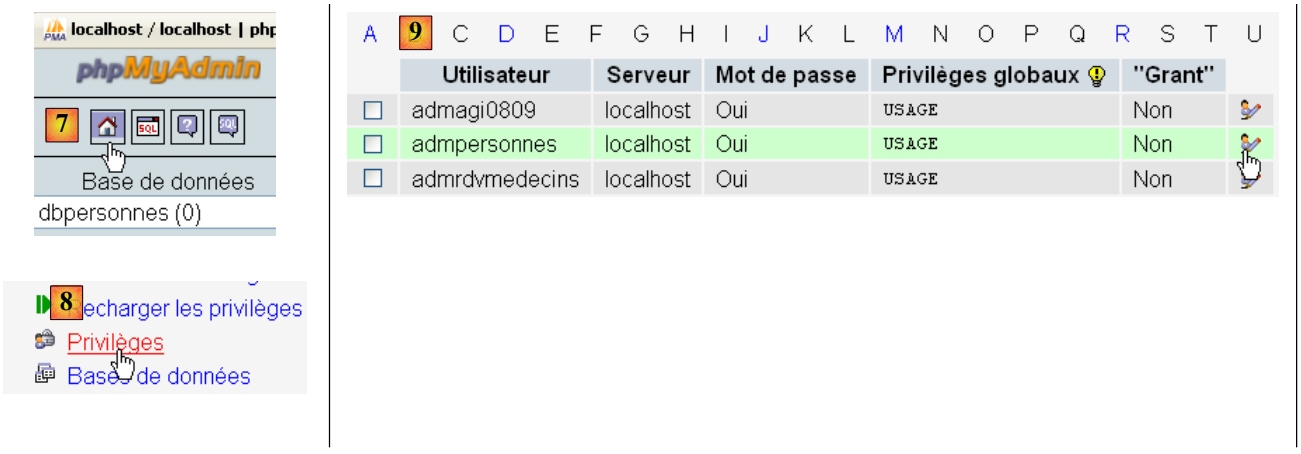
On crée une base de données [dbpersonnes] :

On crée un utilisateur [admpersonnes] avec le mot de passe [nobody] :

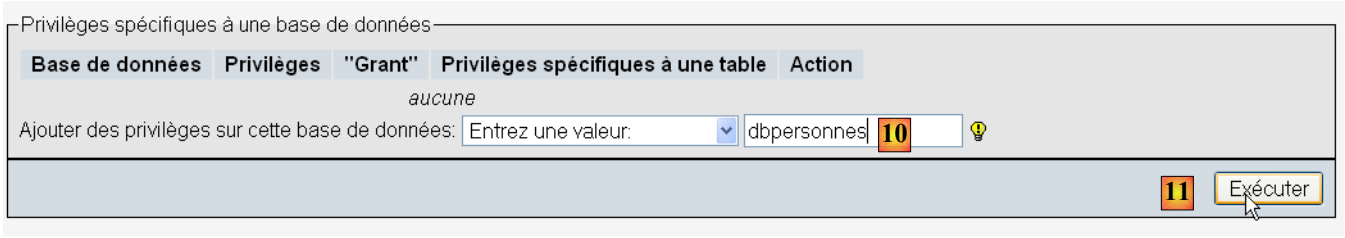




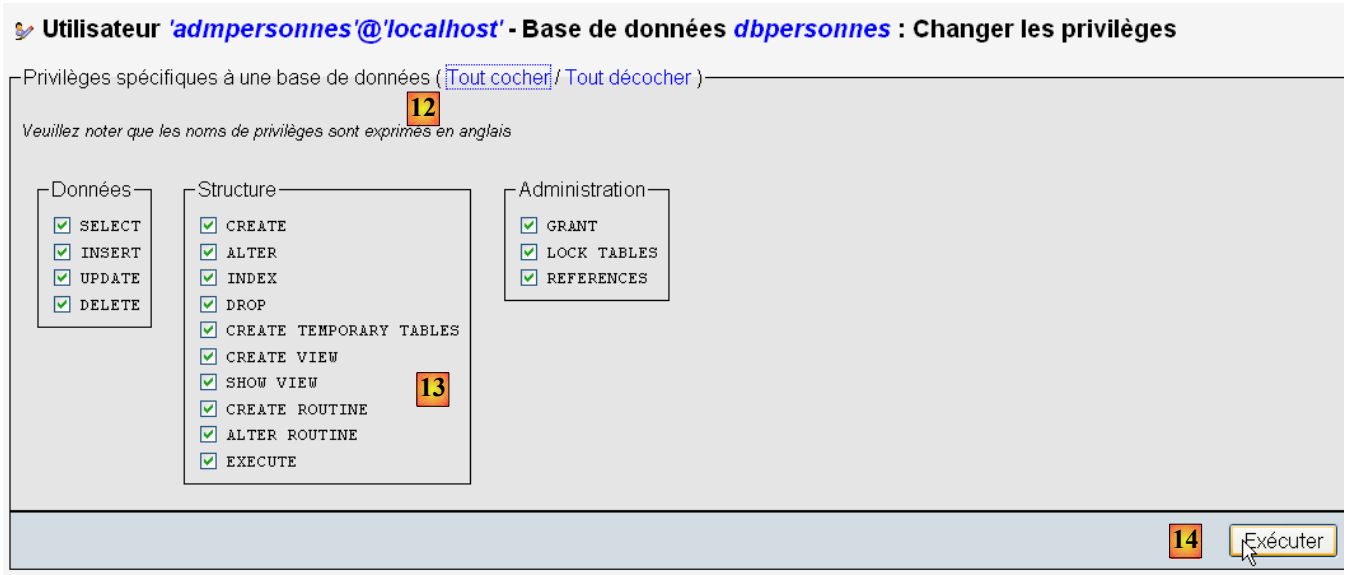
- en [1], le nom de l'utilisateur
- en [2], la machine du SGBD sur lequel on lui donne des droits
- en [3], son mot de passe
- en [4], idem
- en [5], on ne donne aucun droit à cet utilisateur
- en [6], on le crée



- en [7], on revient sur la page d'accueil de *phpMyAdmin*
- en [8], on utilise le lien [Privileges] de cette page pour aller modifier ceux de l'utilisateur [admpersonnes] [9].



- en [10], on indique qu'on veut donner à l'utilisateur [admpersonnes] des droits sur la base de données [dbpersonnes]
- en [11], on valide le choix



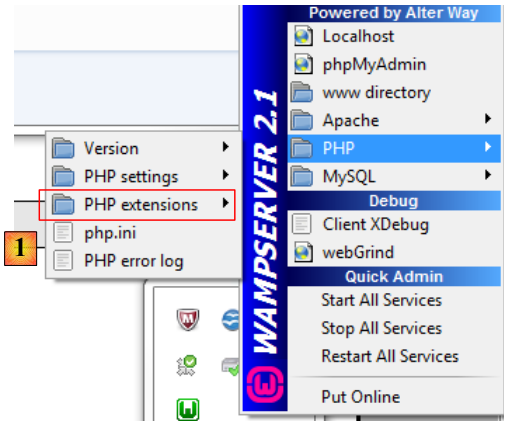
- avec le lien [12] [Tout cocher], on accorde à l'utilisateur [admpersonnes] tous les droits sur la base de données [dbpersonnes] [13]
- on valide en [14]

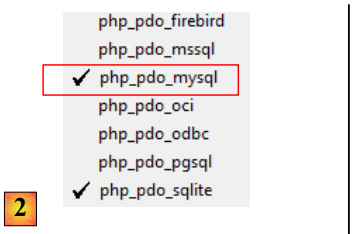
Désormais nous avons :

- une base de données MySQL [dbpersonnes]
- un utilisateur [admpersonnes / nobody] qui a tous les droits sur cette base de données

Nous allons écrire des scripts Php pour exploiter la base de données. PHP dispose de diverses bibliothèques pour gérer les bases de données. Nous utiliserons la bibliothèque PDO (Php Data Objects) qui s'interface entre le code PHP et le SGBD :

La bibliothèque PDO permet au script Php de s'abstraire de la nature exacte du Sgbd utilisé. Ainsi ci-dessus, le Sgbd MySQL peut être remplacé par le Sgbd Postgres avec un impact minimum sur le code du script Php. Cette bibliothèque n'est pas disponible par défaut. On peut vérifier sa disponibilité de la façon suivante :





- 1 : à partir de l'icône d'administration de WampServer, on sélectionne l'option [PHP / PHP extensions]
- 2 : on voit les différentes extensions PDO disponibles et celles qui sont actives : [php_pdo_mysql] pour le Sgbd MySQL, [php_pdo_sqlite] pour le Sgbd SQL Lite. Pour activer une extension, il suffit de cliquer dessus. L'interpréteur Php est alors relancé avec la nouvelle extension activée.

7.1 Connexion à une base MySQL – 1 (mysql_01)

La connexion à un Sgbd se fait par la construction d'un objet PDO. Le constructeur admet différents paramètres :

```
$dbh=new PDO($dsn,$user,$passwd,$driver_options)
```

La signification des paramètres est la suivante :

$dsn (Data Source Name) est une chaîne précisant la nature du Sgbd et sa localisation sur internet. La chaîne "*mysql:host=localhost*" indique qu'on a affaire à un Sgbd MySQL opérant sur le serveur local. Cette chaîne peut comprendre d'autres paramètres, notamment le port d'écoute du Sgbd et le nom de la base à laquelle on veut se connecter : "*mysql:host=localhost:port=3306:dbname=dbpersonnes*".
$user identifiant de l'utilisateur qui se connecte
$passwd son mot de passe
$driver_options un tableau d'options pour le pilote du Sgbd

Seul le premier paramètre est obligatoire. L'objet ainsi construit sera ensuite le support de toutes les opérations faites sur la base de données à laquelle on s'est connecté. Si l'objet PDO n'a pu être construit, une exception de type PDOException est lancée.

Voici un exemple de connexion :

```
1.  <?php
2.
3.  // connexion à une base MySql
4.  // l'identité de l'utilisateur est (admpersonnes,nobody)
5.  $ID = "admpersonnes";
6.  $PWD = "nobody";
7.  $HOTE = "localhost";
8.
9.  try {
10.   // connexion
11.   $dbh = new PDO("mysql:host=$HOTE", $ID, $PWD);
12.   print "Connexion réussie\n";
13.   // fermeture
14.   $dbh = null;
15. } catch (PDOException $e) {
16.   print "Erreur : " . $e->getMessage() . "\n";
17.   exit();
18. }
```

Résultats :

```
Connexion réussie
```

Commentaires

- ligne 11 : la connexion à un Sgbd se fait par la construction d'un objet PDO. Le constructeur est ici utilisé avec les paramètres suivants :
  - une chaîne précisant la nature du Sgbd et sa localisation sur internet. La chaîne "*mysql:host=localhost*" indique qu'on a affaire à un Sgbd MySQL opérant sur le serveur local. Le port n'a pas été précisé. Le port 3306 est alors utilisé par défaut. Le nom de la base de données n'est pas indiqué non plus. Il y aura alors connexion au SGBD MySQL, la sélection d'une base précise se faisant plus tard.
  - un identifiant d'utilisateur
  - son mot de passe
- ligne 14 : la fermeture de la connexion se fait par destruction de l'objet PDO créé initialement.
- ligne 15 : la connexion à un Sgbd peut échouer. Dans ce cas, une exception de type PDOException est lancée.

7.2 Création d'une table MySQL (mysql_02)

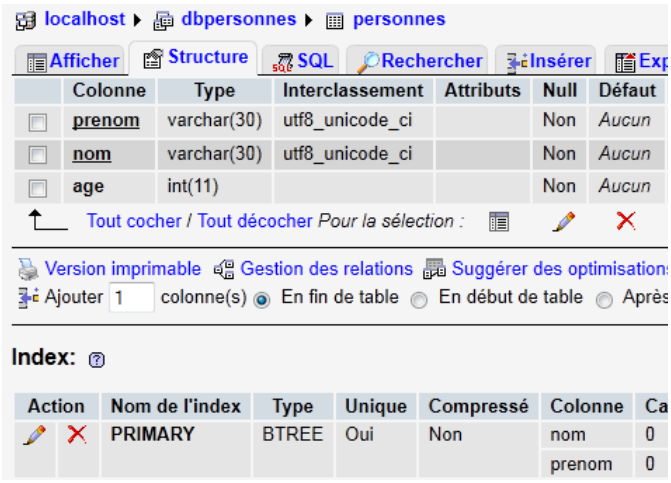
```
1.  <?php
2.
3.  // connexion à la base MySql
4.  // l'identité de l'utilisateur
5.  $ID = "admpersonnes";
6.  $PWD = "nobody";
7.  // identité de la base
8.  $DSN = "mysql:host=localhost;dbname=dbpersonnes";
9.
10. // connexion
11. list($erreur, $connexion) = connecte($DSN, $ID, $PWD);
12. if ($erreur) {
13.   print "Erreur lors de la connexion à la base [$DSN] sous l'identité ($ID,$PWD) : $erreur\n";
14.   exit;
15. }
16.
17. // suppression de la table personnes si elle existe
18. $requête = "drop table personnes";
19. $erreur = exécuteRequête($connexion, $requête);
20. //y-a-t-il eu une erreur ?
21. if ($erreur)
22.   print "$requête : Erreur ($erreur)\n";
23. else
24.   print "$requête: Exécution réussie\n";
25. // création de la table personnes
26. $requête = "create table personnes (prenom varchar(30) NOT NULL, nom varchar(30) NOT NULL, age integer NOT
    NULL, primary key(nom,prenom))";
27. $erreur = exécuteRequête($connexion, $requête);
28. //y-a-t-il eu une erreur ?
29. if ($erreur)
30.   print "$requête : Erreur ($erreur)\n";
31. else
32.   print "$requête: Exécution réussie\n";
33. // on se déconnecte et on quitte
34. déconnecte($connexion);
35. exit;
36.
37. // ----------------------------------------------------------------------------
38. function connecte($dsn, $login, $pwd) {
39.   // connecte ($login,$pwd) à la base $dsn
40.   // rend l'id de la connexion ainsi qu'un code d'erreur
41.   try {
42.     // connexion
43.     $dbh = new PDO($dsn, $login, $pwd);
44.     // retour sans erreur
45.     return array("", $dbh);
46.   } catch (PDOException $e) {
```

```
47.     // retour avec erreur
48.     return array($e->getMessage(), null);
49.   }
50. }
51.
52. // ------------------------------------------------------------------------
53. function déconnecte($connexion) {
54.   // ferme la connexion identifiée par $connexion
55.   $connexion = null;
56. }
57.
58. // ------------------------------------------------------------------------
59. function exécuteRequête($connexion, $sql) {
60.   // exécute la requête $sql sur la connexion $connexion
61.   try {
62.     $connexion->exec($sql);
63.     // retour sans erreur
64.     return "";
65.   } catch (PDOException $e) {
66.     // retour avec erreur
67.     return $e->getMessage();
68.   }
69. }
```

Résultats :

```
drop table personnes: Exécution réussie
create table personnes (prenom varchar(30) NOT NULL, nom varchar(30) NOT NULL, age integer NOT NULL,
primary key(nom,prenom)): Exécution réussie
```

Dans PHPMyAdmin, on peut voir la présence de la table :



Commentaires

- lignes 38-50 : la fonction *connecte* crée une connexion à un Sgbd. Elle rend un tableau (*$erreur, $connexion*) où *$connexion* est la connexion créée ou *null* s'il n'a pu être créée. Dans ce dernier cas, *$erreur* est un message d'erreur.
- lignes 53-56 : la fonction *déconnecte* ferme une connexion
- ligne 59 : la fonction *exécuteRequête* permet d'exécuter un ordre SQL sur une connexion. La connexion est un objet PDO. La méthode utilisée pour exécuter un ordre SQL sur un objet PDO est la méthode **exec** (ligne 63). L'exécution de la requête peut lancer une PDOException. Aussi celle-ci est-elle gérée. La fonction rend un message d'erreur en cas d'erreur, une chaîne vide sinon.

7.3 Remplissage de la table personnes (mysql_03)

Le script suivant exécute des ordres SQL trouvés dans le fichier texte [creation.txt] suivant :

```
drop table personnes
create table personnes (prenom varchar(30) not null, nom varchar(30) not null, age integer not null,
primary key (nom,prenom))
insert into personnes values('Paul','Langevin',48)
insert into personnes values ('Sylvie','Lefur',70)
insert into personnes values ('Pierre','Nicazou',35)
insert into personnes values ('Geraldine','Colou',26)
insert into personnes values ('Paulette','Girond',56)
```

```php
<?php

// connexion à la base MySql
// l'identité de l'utilisateur
$ID = "admpersonnes";
$PWD = "nobody";
// identité de la base
$DSN = "mysql:host=localhost;dbname=dbpersonnes";
// identité du fichier texte des commandes SQL à exécuter
$TEXTE = "creation.txt";

// connexion
list($erreur, $connexion) = connecte($DSN, $ID, $PWD);
if ($erreur) {
  print "Erreur lors de la connexion à la base [$DSN] sous l'identité ($ID,$PWD) : $erreur\n";
  exit;
}

// création et remplissage de la table
$erreurs = exécuterCommandes($connexion, $TEXTE, 1, 0);
//affichage nombre d'erreurs
print "il y a eu $erreurs[0] erreurs\n";
for ($i = 1; $i < count($erreurs); $i++)
  print "$erreurs[$i]\n";

// on se déconnecte et on quitte
déconnecte($connexion);
exit;

// ------------------------------------------------------------------------
function connecte($dsn, $login, $pwd) {
...
}

// ------------------------------------------------------------------------
function déconnecte($connexion) {
 ...
}

// ------------------------------------------------------------------------
function exécuteRequête($connexion, $sql) {
  // exécute la requête $sql sur la connexion $connexion
  // rend 1 msg d'erreur si erreur, la chaîne vide sinon
  ...
}

// ------------------------------------------------------------------------
function exécuterCommandes($connexion, $SQL, $suivi=0, $arrêt=1) {
  // utilise la connexion $connexion
  // exécute les commandes SQL contenues dans le fichier texte $SQL
  // ce fichier est un fichier de commandes SQL à exécuter à raison d'une par ligne
  // si $suivi=1 alors chaque exécution d'un ordre SQL fait l'objet d'un affichage indiquant sa réussite ou son échec
  // si $arrêt=1, la fonction s'arrête sur la 1ère erreur rencontrée sinon elle exécute ttes les commandes sql
  // la fonction rend un tableau (nb d'erreurs, erreur1, erreur2, ...)

```

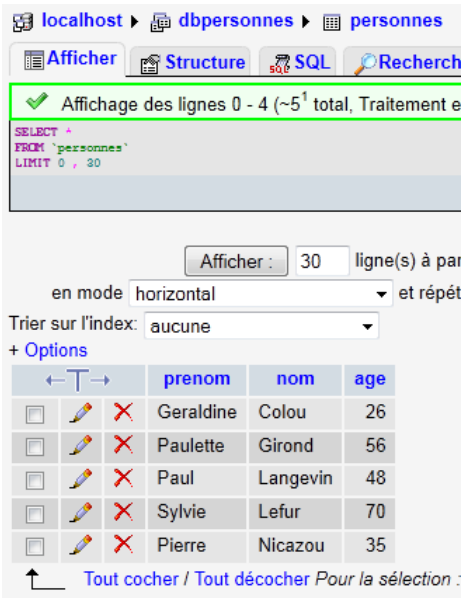
```php
  // on vérifie la présence du fichier $SQL
  if (! file_exists($SQL))
    return array(1, "Le fichier $SQL n'existe pas");

  // exécution des requêtes SQL contenues dans $SQL
  // on les met dans un tableau
  $requêtes = file($SQL);
  // on les exécute - au départ pas d'erreurs
  $erreurs = array(0);
  for ($i = 0; $i < count($requêtes); $i++) {
    //a-t-on une requête vide
    if (preg_match("/^\s*$/", $requêtes[$i]))
      continue;
    // exécution de la requête $i
    $erreur = exécuteRequête($connexion, $requêtes[$i]);
    //y-a-t-il eu une erreur ?
    if ($erreur) {
      // une erreur de plus
      $erreurs[0]++;
      // msg d'erreur
      $msg = "$requêtes[$i] : Erreur ($erreur)\n";
      $erreurs[] = $msg;
      // suivi écran ou non ?
      if ($suivi)
        print "$msg\n";
      // on s'arrête ?
      if ($arrêt)
        return $erreurs;
    } else
    if ($suivi)
      print "$requêtes[$i] : Exécution réussie\n";
  }//for
  // retour
  return $erreurs;
}
```

Les **résultats** écran :

```
drop table personnes : Exécution réussie
create table personnes (prenom varchar(30) not null, nom varchar(30) not null, age integer not null,
primary key (nom,prenom)) : Exécution réussie
insert into personnes values('Paul','Langevin',48) : Exécution réussie
insert into personnes values ('Sylvie','Lefur',70) : Exécution réussie
insert into personnes values ('Pierre','Nicazou',35) : Exécution réussie
insert into personnes values ('Geraldine','Colou',26) : Exécution réussie
insert into personnes values ('Paulette','Girond',56) : Exécution réussie
il y a eu 0 erreurs
```

Les insertions faites sont visibles avec PhpMyAdmin :



Commentaires

La nouveauté réside dans la fonction *exécuterCommandes* des lignes 48-90. Cette fonction exécute sur la connexion $*connexion* les ordres SQL trouvés dans le fichier texte de nom $*SQL*. Elle rend un tableau d'erreurs (*$nbErreurs, $msg1, $msg2,* …) où $*nbErreurs* est le nombre d'erreurs, $*msgi* le message d'erreur n° *i*. S'il n'y a pas d'erreurs, le tableau rendu est le tableau *array(0)*.

7.4 Exécution de requêtes SQL quelconques (mysql_04)

Le script suivant montre l'exécution des ordres SQL du fichier texte [sql.txt] suivant :

```
select * from personnes
select nom,prenom from personnes order by nom asc, prenom desc
select * from personnes where age between 20 and 40 order by age desc, nom asc, prenom asc
insert into personnes values('Josette','Bruneau',46)
update personnes set age=47 where nom='Bruneau'
select * from personnes where nom='Bruneau'
delete from personnes where nom='Bruneau'
select * from personnes where nom='Bruneau'
xselect * from personnes where nom='Bruneau'
```

Parmi ces ordres SQL, il y a l'ordre **select** qui ramène des résultats de la base de données, les ordres **insert**, **update**, **delete** qui modifient la base sans ramener de résultats et enfin des ordres erronés tels que le dernier (**xselect**).

```
1.  <?php
2.
3.  // connexion à la base MySql
4.  // l'identité de l'utilisateur
5.  $ID = "admpersonnes";
6.  $PWD = "nobody";
7.  // identité de la base
8.  $DSN = "mysql:host=localhost;dbname=dbpersonnes";
9.  // identité du fichier texte des commandes SQL à exécuter
10. $TEXTE = "sql.txt";
11.
12. // connexion
13. list($erreur, $connexion) = connecte($DSN, $ID, $PWD);
14. if ($erreur) {
15.   print "Erreur lors de la connexion à la base [$DSN] sous l'identité ($ID,$PWD) : $erreur\n";
16.   exit;
17. }
18.
19. // exécution des ordres SQL
```

```php
$erreurs = exécuterCommandes($connexion, $TEXTE, 1, 0);
//affichage nombre d'erreurs
print "il y a eu $erreurs[0] erreur(s)\n";
for ($i = 1; $i < count($erreurs); $i++)
  print "$erreurs[$i]\n";

// on se déconnecte et on quitte
déconnecte($connexion);
exit;

// ------------------------------------------------------------------------
function connecte($dsn, $login, $pwd) {
  // connecte ($login,$pwd) à la base $dsn
  // rend l'id de la connexion ainsi qu'un msg d'erreur
  ...
}

// ------------------------------------------------------------------------
function déconnecte($connexion) {
  ...
}

// ------------------------------------------------------------------------
function exécuteRequête($connexion, $sql) {
  // exécute la requête $sql sur la connexion $connexion
  // rend un tableau de 2 éléments ($erreur,$résultat)

  // on détermine si c'est un select ou non
  $commande = "";
  if (preg_match("/^\s*(\S+)/", $sql, $champs)) {
    $commande = $champs[0];
  }
  // exécution de la commande
  try {
    if (strtolower($commande) == "select") {
      $res = $connexion->query($sql);
    } else {
      $res = $connexion->exec($sql);
      if($res===FALSE){
        $info=$connexion->errorInfo();
        return array($info[2],null);
      }
    }
    // retour sans erreur
    return array("", $res);
  } catch (PDOException $e) {
    // retour avec erreur
    return array($e->getMessage(), null);
  }
}

// ------------------------------------------------------------------------
function exécuterCommandes($connexion, $SQL, $suivi=0, $arrêt=1) {
  // utilise la connexion $connexion
  // exécute les commandes SQL contenues dans le fichier texte $SQL
  // ce fichier est un fichier de commandes SQL à exécuter à raison d'une par ligne
  // si $suivi=1 alors chaque exécution d'un ordre SQL fait l'objet d'un affichage indiquant sa réussite ou son échec
  // si $arrêt=1, la fonction s'arrête sur la 1ère erreur rencontrée sinon elle exécute ttes les commandes sql
  // la fonction rend un tableau (nb d'erreurs, erreur1, erreur2, ...)
  // on vérifie la présence du fichier $SQL
  if (!file_exists($SQL))
    return array(1, "Le fichier $SQL n'existe pas");

  // exécution des requêtes SQL contenues dans $TEXTE
  // on les met dans un tableau
  $requêtes = file($SQL);
```

```
86.   // on les exécute - au départ pas d'erreurs
87.   $erreurs = array(0);
88.   for ($i = 0; $i < count($requêtes); $i++) {
89.     //a-t-on une requête vide
90.     if (preg_match("/^\s*$/", $requêtes[$i]))
91.       continue;
92.     // exécution de la requête $i
93.     list($erreur, $res) = exécuteRequête($connexion, $requêtes[$i]);
94.     //y-a-t-il eu une erreur ?
95.     if ($erreur) {
96.       // une erreur de plus
97.       $erreurs[0]++;
98.       // msg d'erreur
99.       $msg = "$requêtes[$i] : Erreur ($erreur)\n";
100.      $erreurs[] = $msg;
101.      // suivi écran ou non ?
102.      if ($suivi)
103.        print "$msg\n";
104.      // on s'arrête ?
105.      if ($arrêt)
106.        return $erreurs;
107.    } else
108.    if ($suivi) {
109.      print "$requêtes[$i] : Exécution réussie\n";
110.      // infos sur le résultat de la requête exécutée
111.      afficherInfos($res);
112.    }
113.  }//for
114.  // retour
115.  return $erreurs;
116. }
117.
118. // ----------------------------------------------------------------------------
119. function afficherInfos($résultat) {
120.   // affiche le résultat $résultat d'une requête sql
121.   // s'agissait-il d'un select ?
122.   if ($résultat instanceof PDOStatement) {
123.     // on affiche les noms des champs
124.     $titre = "";
125.     $nbColonnes = $résultat->columnCount();
126.     for ($i = 0; $i < $nbColonnes; $i++) {
127.       $infos = $résultat->getColumnMeta($i);
128.       $titre.=$infos['name'] . ",";
129.     }
130.     // on enlève le dernier caractère ,
131.     $titre = substr($titre, 0, strlen($titre) - 1);
132.     // on affiche la liste des champs
133.     print "$titre\n";
134.     // ligne séparatrice
135.     $séparateurs = "";
136.     for ($i = 0; $i < strlen($titre); $i++) {
137.       $séparateurs.="-";
138.     }
139.     print "$séparateurs\n";
140.     // données
141.     foreach ($résultat as $ligne) {
142.       $data = "";
143.       for ($i = 0; $i < $nbColonnes; $i++) {
144.         $data.=$ligne[$i] . ",";
145.       }
146.
147.       // on enlève le dernier caractère ,
148.       $data = substr($data, 0, strlen($data) - 1);
149.       // on affiche
150.       print "$data\n";
151.     }
```

```
152.   } else {
153.     // ce n'était pas un select
154.     print " $résultat lignes(s) a (ont) été modifiée(s)\n";
155.   }
156. }
```

Les **résultats** écran :

```
select * from personnes
 : Exécution réussie
prenom,nom,age
--------------
Geraldine,Colou,26
Paulette,Girond,56
Paul,Langevin,48
Sylvie,Lefur,70
Pierre,Nicazou,35
select nom,prenom from personnes order by nom asc, prenom desc : Exécution réussie
nom,prenom
----------
Colou,Geraldine
Girond,Paulette
Langevin,Paul
Lefur,Sylvie
Nicazou,Pierre
select * from personnes where age between 20 and 40 order by age desc, nom asc, prenom asc : Exécution
réussie
prenom,nom,age
--------------
Pierre,Nicazou,35
Geraldine,Colou,26
insert into personnes values('Josette','Bruneau',46) : Exécution réussie
 1 lignes(s) a (ont) été modifiée(s)
update personnes set age=47 where nom='Bruneau' : Exécution réussie
 1 lignes(s) a (ont) été modifiée(s)
select * from personnes where nom='Bruneau' : Exécution réussie
prenom,nom,age
--------------
Josette,Bruneau,47
delete from personnes where nom='Bruneau' : Exécution réussie
 1 lignes(s) a (ont) été modifiée(s)
select * from personnes where nom='Bruneau' : Exécution réussie
prenom,nom,age
--------------
xselect * from personnes where nom='Bruneau' : Erreur (You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right syntax to use near 'xselect * from
personnes where nom='Bruneau'' at line 1)

il y a eu 1 erreur(s)
xselect * from personnes where nom='Bruneau' : Erreur (You have an error in your SQL syntax; check the
manual that corresponds to your MySQL server version for the right syntax to use near 'xselect * from
personnes where nom='Bruneau'' at line 1)
```

Commentaires

Chacune des commandes du fichier texte [sql.txt] est exécutée par la fonction *exécuteRequête* de la ligne 43.

- ligne 43 : les deux paramètres de la fonction sont la connexion ($*connexion*) sur laquelle doivent être exécutés les ordres SQL et l'ordre sql ($*sql*) à exécuter. La fonction rend un tableau de deux valeurs ($*erreur*,$*résultat*) où
  - $*erreur* est un message d'erreur éventuellement vide s'il n'y a pas eu d'erreur
  - $*résultat* : le résultat rendu par l'exécution de l'ordre SQL. Ce résultat est différent selon que l'ordre est un ordre *select* ou bien un ordre *insert*, *update*, *delete*.
- lignes 48-51 : on récupère le 1er élément de l'ordre SQL pour savoir si on affaire à un ordre *select* ou bien à un ordre *insert*, *update*, *delete*.
- ligne 55 : dans le cas d'un ordre *select*, celui-ci est exécuté avec la méthode *[PDO]->query("ordre select")*. Le résultat rendu est un objet de type *PDOStatement*.
- ligne 57 : dans le cas d'un ordre *insert, update, delete*, celui-ci est exécuté avec la méthode *[PDO]->exec("ordre SQL")*. Le résultat rendu est le nombre de lignes modifiées par l'ordre SQL. Ainsi si un ordre SQL *delete* supprime deux lignes, le résultat rendu est l'entier 2. S'il y a une erreur à l'exécution, le résultat rendu est le booléen *false*. Dans ce cas, la méthode *[PDO]->errorinfo()* donne des informations sur l'erreur sous la forme d'un tableau de valeurs. L'élément d'indice 2 de ce tableau est le message d'erreur.
- lignes 58-60 : traitement de l'éventuelle erreur de l'opération *[PDO]->exec("ordre SQL")*.
- lignes 65-68 : traitement de l'éventuelle exception

- ligne 72 : la fonction *exécuterCommandes* exécute sur la connexion $*connexion* les commandes SQL stockées dans le fichier texte $*SQL*. C'est un code que nous avons déjà rencontré à un détail près : la ligne 111.
- ligne 111 : la fonction *exécuteRequête* a rendu un tableau ($*erreur*,$*résultat*) ou $*résultat* est le résultat de l'exécution d'une commande SQL. Ce résultat est différent selon que l'ordre SQL était un ordre *select* ou bien un ordre *insert*, *update*, *delete*. La fonction *afficherInfos* affiche des informations sur ce résultat.
- ligne 122 : si l'ordre SQL était un ordre *select*, le résultat est de type *PDOStatement*. Ce type représente une table faite de lignes et de colonnes.
- ligne 125 : la méthode *[PDOStatement]->getColumnCount()* rend le nombre de colonnes de la table résultat du *select*
- ligne 127 : la méthode *[PDOStatement]->getMeta(i)* rend un dictionnaire d'informations sur la colonne n° *i* de la table résultat du *select*. Dans ce dictionnaire, la valeur associée à la clé '*name*' est le nom de la colonne.
- lignes 127-129 : les noms des colonnes de la table résultat du *select* sont concaténées dans une chaîne de caractères.
- lignes 141-145 : un objet de type *PDOStatement* peut être parcouru par une boucle *foreach*. A chaque itération, l'élément obtenu est une ligne de la table résultat du *select* sous la forme d'un tableau de valeurs représentant les valeurs des différentes colonnes de la ligne. On affiche toutes ces valeurs avec une boucle *for*.
- ligne 154 : le résultat de l'exécution d'un ordre *insert*, *update*, *delete* est le nombre de lignes modifiées par l'ordre.

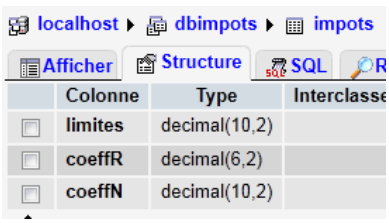
8 Exercice IMPOTS avec MySQL

Nous avons déjà écrit trois versions de cet exercice. La dernière version utilisait une classe de calcul de l'impôt. Cette classe prenait dans un fichier texte, les données permettant ce calcul. Elle va désormais les prendre dans une base de données. Pour cela, nous écrivons un premier code qui va transférer les données du fichier texte dans une base de données.

Le fichier texte *impots.txt* est le suivant:

```
12620:13190:15640:24740:31810:39970:48360:55790:92970:127860:151250:172040:195000:0
0:0.05:0.1:0.15:0.2:0.25:0.3:0.35:0.4:0.45:0.5:0.55:0.6:0.65
0:631:1290.5:272.5:3309.5:4900:6898.5:9316.5:12106:16754.5:23147.5:30710:39312:49062
```

La base de données à construire est la suivante :



limites	coeffR	coeffN
12620.00	0.00	0.00
13190.00	0.05	631.00
15640.00	0.10	1290.50
24740.00	0.15	2072.50
31810.00	0.20	3309.50
39970.00	0.25	4900.00
48360.00	0.30	6898.50
55790.00	0.35	9316.50
92970.00	0.40	12106.00
127860.00	0.45	16754.50
151250.00	0.50	23147.50
172040.00	0.55	30710.00
195000.00	0.60	39312.00
0.00	0.65	49062.00

La base s'appelle [dbimpots] et a une unique table [impots]. Elle est exploitée par l'utilisateur [root] sans mot de passe.

8.1 Transfert d'un fichier texte dans une table MySQL (txt2mysql)

```
1.  <?php
2.
3.  // transfère le fichier texte des données nécessaires au calcul des impôts
4.  // dans une table mysql
5.  // les données
6.  $IMPOTS = "impots.txt";
7.  $BASE = "dbimpots";
8.  $TABLE = "impots";
9.  $USER = "root";
10. $PWD = "";
11. $HOTE = "localhost";
12.
13. // les données nécessaires au calcul de l'impôt ont été placées dans le fichier $IMPOTS
14. // à raison d'une ligne par tableau sous la forme
15. // val1:val2:val3,...
16. list($erreur, $limites, $coeffR, $coeffN) = getTables($IMPOTS);
17. // y-a-t-il eu une erreur ?
18. if ($erreur) {
19.   print "$erreur\n";
20.   exit;
21. }//if
22. // on transfère ces tableaux dans une table mysql
23. $erreur = copyToMysql($limites, $coeffR, $coeffN, $HOTE, $USER, $PWD, $BASE, $TABLE);
```

```
24. if ($erreur)
25.   print "$erreur\n";
26. else
27.   print "Transfert opéré\n";
28. // fin
29. exit;
30.
31. // ------------------------------------------------------------------------
32. function copyToMysql($limites, $coeffR, $coeffN, $HOTE, $USER, $PWD, $BASE, $TABLE) {
33.   // copie les 3 tableaux numériques $limites, $coeffR, $coeffN
34.   // dans la table $TABLE de la base mysql $BASE
35.   // la base mysql est sur la machine $HOTE
36.   // l'utilisateur est identifié par $USER et $PWD
37.   // connexion
38.   list($erreur, $connexion) = connecte("mysql:host=$HOTE;dbname=$BASE", $USER, $PWD);
39.   if ($erreur)
40.     return "Erreur lors de la connexion à MySql sous l'identité ($HOTE,$USER,$PWD) : $erreur\n";
41.   // suppression de la table
42.   $requête = "drop table $TABLE";
43.   exécuteRequête($connexion, $requête);
44.   // création de la table
45.   $requête = "create table $TABLE (limites decimal(10,2), coeffR decimal(6,2), coeffN decimal(10,2))";
46.   list($erreur, $res) = exécuteRequête($connexion, $requête);
47.   if ($erreur)
48.     return "$requête : erreur ($erreur)";
49.   // remplissage
50.   for ($i = 0; $i < count($limites); $i++) {
51.     // requête d'insertion
52.     $requête = "insert into $TABLE (limites,coeffR,coeffN) values ($limites[$i],$coeffR[$i],$coeffN[$i])";
53.     // exécution de la requête
54.     list($erreur, $res) = exécuteRequête($connexion, $requête);
55.     // retour si erreur
56.     if ($erreur)
57.       return "$requête : erreur ($erreur)";
58.   }//for
59.   // c'est fini - on se déconnecte
60.   déconnecte($connexion);
61.   // retour sans erreur
62.   return "";
63. }
64. // ------------------------------------------------------------------------
65. function getTables($IMPOTS) {
66.   // $IMPOTS : le nom du fichier contenant les données des tables $limites, $coeffR, $coeffN
67.   ...
68.   // fin
69.   return array("", $limites, $coeffR, $coeffN);
70. }
71. // ------------------------------------------------------------------------
72. function cutNewLinechar($ligne) {
73.   // on supprime la marque de fin de ligne de $ligne si elle existe
74.   ...
75.   // fin
76.   return($ligne);
77. }
78. // ------------------------------------------------------------------------
79. function connecte($dsn, $login, $pwd) {
80.   // connecte ($login,$pwd) à la base $dsn
81.   // rend l'id de la connexion ainsi qu'un msg d'erreur
82.   ...
83. }
84.
85. //connecte
86. // ------------------------------------------------------------------------
87. function déconnecte($connexion) {
88.   // ferme la connexion identifiée par $connexion
89.   ...
```

```
90.  }
91.
92.  // ----------------------------------------------------------------------------
93.  function exécuteRequête($connexion, $sql) {
94.    // exécute la requête $sql sur la connexion $connexion
95.    // rend un tableau de 2 éléments ($erreur,$résultat)
96.   ...
97.  }
```

Les **résultats** écran :

```
Transfert opéré
```

8.2 Le programme de calcul de l'impôt (impots_04)

Cette version utilise une classe *Impôts* qui va chercher dans une base de données les valeurs nécessaires au calcul de l'impôt. Nous introduisons ici une nouvelle notion, celle de **requête préparée**. L'exécution d'un ordre SQL par un Sgbd est faite en deux temps :

1. la requête est préparée : le Sgbd va préparer un plan d'exécution optimisé pour la requête. Il s'agit de l'exécuter le plus efficacement possible.
2. la requête est exécutée.

Si une même requête est exécutée N fois, les deux étapes précédentes sont faites N fois. Il est cependant possible de préparer la requête 1 fois et de l'exécuter N fois. Pour cela il faut utiliser des requêtes préparées. Si $*requête* est l'ordre SQL à exécuter et $*connexion* l'objet PDO représentant la connexion :

$statement=$connexion->prepare($requête) prépare une requête et rend la requête "préparée"
$statement->execute() exécute la requête préparée.

Si la requête préparée est un ordre *select* alors
$statement->fetchAll() ramène toutes les lignes de la table résultat du *select* sous la forme d'un tableau T où *T[i,j]* est la valeur de la colonne *j* de la ligne *i* de la table
$statement->fetch() ramène la ligne courante de la table sous la forme d'un tbaleau T où *T[j]* est la valeur de la colonne *j* de la ligne

La requête préparée apporte d'autres avantages que celui d'une meilleure efficacité. Elle amène notamment plus de sécurité. Aussi devrait-on l'utiliser systématiquement.

```
1.  <?php
2.
3.  // test ----------------------------------------------------
4.  // définition des constantes
5.  $DATA = "data.txt";
6.  $RESULTATS = "resultats.txt";
7.  $TABLE = "impots";
8.  $BASE = "dbimpots";
9.  $USER = "root";
10. $PWD = "";
11. $HOTE="localhost";
12.
13. // les données nécessaires au calcul de l'impôt ont été placées dans la table mysqL $TABLE
14. // appartenant à la base $BASE. La table a la structure suivante
15. // limites decimal(10,2), coeffR decimal(6,2), coeffN decimal(10,2)
16. // les paramètres des personnes imposables (statut marital, nombre d'enfants, salaire annuel)
17. // ont été placés dans le fichier texte $DATA à raison d'une ligne par contribuable
18. // les résultats (statut marital, nombre d'enfants, salaire annuel, impôt à payer) sont placés dans
19. // le fichier texte $RESULTATS à raison d'un résultat par ligne
20.
21. // on crée un objet Impôts
22. $I = new Impôts($HOTE, $USER, $PWD, $BASE, $TABLE);
23. // y-a-t-il eu une erreur ?
24. $erreur=$I->getErreur();
25. if ($erreur) {
26.   print "$I->erreur\n";
```

```
27.   exit;
28. }//if
29. // on crée un objet utilitaires
30. $u = new Utilitaires();
31.
32. // ouverture fichier des données des contribuables
33. $data = fopen($DATA, "r");
34. if (!$data) {
35.   print "Impossible d'ouvrir en lecture le fichier des données [$DATA]\n";
36.   exit;
37. }
38.
39. // ouverture fichier des résultats
40. $résultats = fopen($RESULTATS, "w");
41. if (!$résultats) {
42.   print "Impossible de créer le fichier des résultats [$RESULTATS]\n";
43.   exit;
44. }
45.
46. // on exploite la ligne courante du fichier des données contribuables
47. while ($ligne = fgets($data, 100)) {
48.   // on enlève l'éventuelle marque de fin de ligne
49.   $ligne = $u->cutNewLineChar($ligne);
50.   // on récupère les 3 champs marié,enfants,salaire qui forment $ligne
51.   list($marié, $enfants, $salaire) = explode(",", $ligne);
52.   // on calcule l'impôt
53.   $impôt = $I->calculer($marié, $enfants, $salaire);
54.   // on inscrit le résultat
55.   fputs($résultats, "$marié:$enfants:$salaire:$impôt\n");
56.   // donnée suivante
57. }
58. // on ferme les fichiers
59. fclose($data);
60. fclose($résultats);
61.
62. // fin
63. exit;
64.
65. // ------------------------------------------------------------------------
66. // définition d'une classe Impôts
67. class Impôts {
68.
69.   // attributs : les 3 tableaux de données
70.   private $limites;
71.   private $coeffR;
72.   private $coeffN;
73.   private $erreur;
74.   private $nbLimites;
75.
76.   // getter
77.   public function getErreur(){
78.     return $this->erreur;
79.   }
80.   // constructeur
81.
82.   function __construct($HOTE, $USER, $PWD, $BASE, $TABLE) {
83.     // initialise les attributs $limites, $coeffR, $coeffN
84.     // les données nécessaires au calcul de l'impôt ont été placées dans la table mysql $TABLE
85.     // appartenant à la base $BASE. La table a la structure suivante
86.     // limites decimal(10,2), coeffR decimal(6,2), coeffN decimal(10,2)
87.     // la connexion à la base mysql de la machine $HOTE se fait sous l'identité ($USER,$PWD)
88.     // initialise le champ $erreur avec une éventuelle erreur
89.     // vide si pas d'erreur
90.     //
91.     // connexion à la base mysql
```

```
92.     $DSN = "mysql:host=$HOTE;dbname=$BASE";
93.     list($erreur, $connexion) = connecte($DSN, $USER, $PWD);
94.     if ($erreur) {
95.       $this->erreur = "Erreur lors de la connexion à MySql sous l'identité ($HOTE,$USER,$PWD) : $erreur\n";
96.       return;
97.     }
98.     // lecture de la table $TABLE
99.     $requête = "select limites,coeffR,coeffN from $TABLE";
100.    // exécute la requête $requête sur la connexion $connexion
101.    try {
102.      $statement = $connexion->prepare($requête);
103.      $statement->execute();
104.      // exploitation du résultat de la requête
105.      while ($colonnes = $statement->fetch()) {
106.        $this->limites[] = $colonnes[0];
107.        $this->coeffR[] = $colonnes[1];
108.        $this->coeffN[] = $colonnes[2];
109.      }
110.      // pas d'erreur
111.      $this->erreur = "";
112.      // nombre d'élements du tableau limites
113.      $this->nbLimites = count($this->limites);
114.    } catch (PDOException $e) {
115.      $this->erreur = $e->getMessage();
116.    }
117.    // déconnexion
118.    déconnecte($connexion);
119.  }
120.  // ----------------------------------------------------------------
121.  function calculer($marié, $enfants, $salaire) {
122.    // $marié : oui, non
123.    // $enfants : nombre d'enfants
124.    // $salaire : salaire annuel
125.
126.  // l'objet est-il dans un état correct ?
127.    if ($this->erreur)
128.      return -1;
129.
130.    // nombre de parts
131.    ...
132.  }
133.
134. }
135.
136. // ------------------------------------------------------------------------------
137. // une classe de fonctions utilitaires
138. class Utilitaires {
139.
140.   function cutNewLinechar($ligne) {
141.     // on supprime la marque de fin de ligne de $ligne si elle existe
142.     ...
143.   }
144.
145. }
146. // ------------------------------------------------------------------------------
147. function connecte($dsn, $login, $pwd) {
148.   // connecte ($login,$pwd) à la base $dsn
149.   // rend l'id de la connexion ainsi qu'un msg d'erreur
150.   ...
151. }
152.
153. //connecte
154. // ------------------------------------------------------------------------------
155. function déconnecte($connexion) {
156.   // ferme la connexion identifiée par $connexion
```

```
157.  ...
158. }
```

Résultats : les mêmes qu'avec les versions précédentes.

Commentaires

Les nouveautés sont lignes 98-109 :

- ligne 99 : l'ordre SQL *select* qui va permettre de récupérer les données nécessaires au calcul de l'impôt.
- ligne 102 : préparation de l'ordre SQL *select*
- ligne 103 : exécution de l'ordre préparé
- lignes 105-109 : exploitation ligne par ligne de la table résultat du *select*

9 Les fonctions réseau de PHP

Nous abordons maintenant les fonctions réseau de Php qui nous permettent de faire de la programmation TCP / IP (Transfer Control Protocol / Internet Protocol).

9.1 Obtenir le nom ou l'adresse IP d'une machine de l'Internet (inet_01)

```
1.  <?php
2.
3.  // fonctions nom Machine <--> adresse IP machine
4.  ini_set("display_errors", "off");
5.  // constantes
6.  $HOTES = array("istia.univ-angers.fr", "www.univ-angers.fr", "www.ibm.com", "localhost", "", "xx");
7.  // adresses IP des machines de $HOTES
8.  for ($i = 0; $i < count($HOTES); $i++) {
9.    getIPandName($HOTES[$i]);
10. }
11. // fin
12. exit;
13.
14. //-----------------------------------------------
15. function getIPandName($nomMachine) {
16. //$nomMachine : nom de la machine dont on veut l'adresse IP
17. // nomMachine-->adresse IP
18.   $ip = gethostbyname($nomMachine);
19.   if ($ip != $nomMachine) {
20.     print "ip[$nomMachine]=$ip\n";
21. // adresse IP --> nomMachine
22.     $name = gethostbyaddr($ip);
23.     if ($name != $ip) {
24.       print "name[$ip]=$name\n";
25.     } else {
26.       print "Erreur, machine[$ip] non trouvée\n";
27.     }
28.   } else {
29.     print "Erreur, machine[$nomMachine] non trouvée\n";
30.   }
31. }
```

Résultats :

```
ip[istia.univ-angers.fr]=193.49.146.171
name[193.49.146.171]=istia.istia.univ-angers.fr
ip[www.univ-angers.fr]=193.49.144.40
name[193.49.144.40]=ametys-fo.univ-angers.fr
ip[www.ibm.com]=129.42.56.216
Erreur, machine[129.42.56.216] non trouvée
ip[localhost]=127.0.0.1
name[127.0.0.1]=localhost127.0.0.1
ip[]=192.168.1.11
name[192.168.1.11]=st-PC.home
Erreur, machine[xx] non trouvée
```

Commentaires

- ligne 4 : on demande à ce que les erreurs d'exécution ne soient pas affichées.

Les fonctions réseau de Php sont utilisées dans la fonction *getIpandName* de la ligne 15.

- ligne 18 : la fonction *gethostbyname($nom)* permet d'obtenir l'adresse IP "ip3.ip2.ip1.ip0" de la machine s'appelant $*nom*. Si la machine $*nom* n'existe pas, la fonction rend $*nom* comme résultat.
- ligne 22 : la fonction *gethostbyaddr($ip)* permet d'obtenir le nom de la machine d'adresse $ip de la forme "ip3.ip2.ip1.ip0". Si la machine $*ip* n'existe pas, la fonction rend $*ip* comme résultat.

9.2 Un client web (inet_02)

Un script permettant d'avoir le contenu de la page index d'un site web.

```
1.  <?php
2.
3.  // gestion des erreurs
4.  ini_set("display_errors","off");
5.  // otenir le texte HTML d'URL
6.  // liste de sites web
7.  $SITES = array("istia.univ-angers.fr", "www.univ-angers.fr", "www.ibm.com", "xx");
8.  // lecture des pages index des sites du tableau $SITES
9.  for ($i = 0; $i < count($SITES); $i++) {
10. // lecture page index du site $SITES[$i]
11.   $résultat = getIndex($SITES[$i]);
12. // affichage résultat
13.   print "$résultat\n";
14. }//for
15. // fin
16. exit;
17.
18. //---------------------------------------------------------------------
19. function getIndex($site) {
20. // lit l'URL $site/ et la stocke dans le fichier $site.html
21. // création du fichier $site.html
22.   $html = fopen("$site.html", "w");
23.   if (!$html)
24.     return "Erreur lors de la création du fichier $site.html";
25.
26. // ouverture d'une connexion sur le port 80 de $site
27.   $connexion = fsockopen($site, 80);
28. // retour si erreur
29.   if (!$connexion)
30.     return "Echec de la connexion au site ($site,80) : $erreur";
31. // $connexion représente un flux de communication bidirectionnel
32. // entre le client (ce programme) et le serveur web contacté
33. // ce canal est utilisé pour les échanges de commandes et d'informations
34. // le protocole de dialogue est HTTP
35. // le client envoie la commande get pour demander l'URL /
36. // syntaxe get URL HTTP/1.0
37. // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
38.   fputs($connexion, "GET / HTTP/1.0\n\n");
39. // le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
40. // ces données puis fermer le canal. Le client lit donc tout ce qui arrive de $connexion
41. // jusqu'à la fermeture du canal
42.   while ($ligne = fgets($connexion, 1000))
43.     fputs($html, $ligne);
44. // le client ferme la connexion à son tour
45.   fclose($connexion);
46. // fermeture du fichier $html
47.   fclose($html);
48. // retour
```

```
49.    return "Transfert réussi de la page index du site $site";
50.  }
```

Résultats : par exemple, le fichier reçu pour le site [www.ibm.com] :

```
1.  HTTP/1.1 302 Found
2.  Date: Wed, 08 Jun 2011 15:43:56 GMT
3.  Server: IBM_HTTP_Server
4.  Content-Type: text/html
5.  Expires: Fri, 01 Jan 1990 00:00:00 GMT
6.  Pragma: no-cache
7.  Cache-Control: no-cache, must-revalidate
8.  Location: http://www.ibm.com/us/en/
9.  Content-Length: 209
10. Kp-eeAlive: timeout=10, max=14
11. Connection: Keep-Alive
12.
13. <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
14. <html><head>
15. <title>302 Found</title>
16. </head><body>
17. <h1>Found</h1>
18. <p>The document has moved <a href="http://www.ibm.com/us/en/">here</a>.</p>
19. </body></html>
```

- les lignes 1-11 sont les entêtes HTTP de la réponse du serveur
- ligne 1 : le serveur demande au client de se rediriger vers l'Url indiquée ligne 8
- ligne 2 : date et heure de la réponse
- ligne 3 : identité du serveur web
- ligne 4 : contenu envoyé par le serveur. Ici une page HTML qui commence ligne 13
- ligne 12 : la ligne vide qui termine les entêtes HTTP
- lignes 13-19 : la page HTML envoyée par le serveur web.

Commentaires du code :

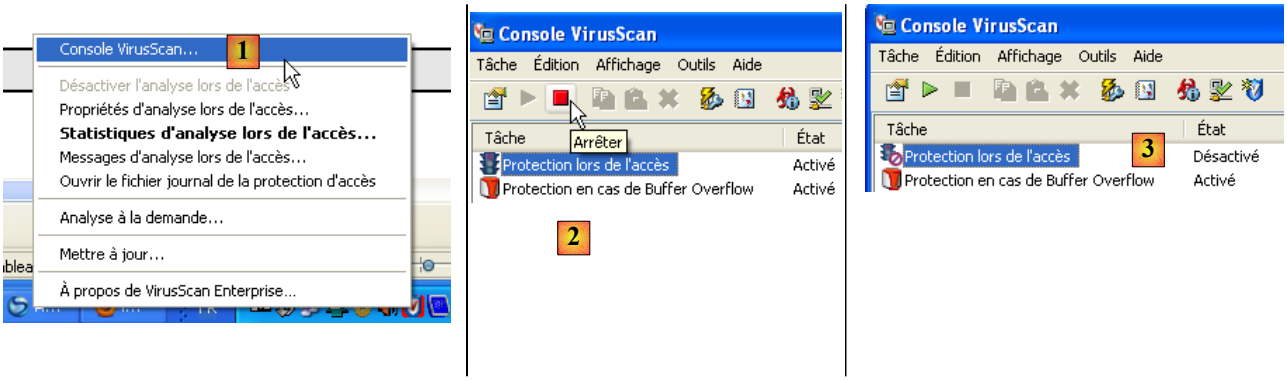
- ligne 7 : la liste des Url des sites web dont on veut la page index. Celle-ci sera stockée dans le fichier texte [nomsite.html].
- ligne 11 : la fonction *getIndex* fait le travail
- ligne 19 : la fonction *getIndex($site)* télécharge la page racine (ou page index) du site web $*site* et la stocke dans le fichier texte $*site.html*.
- ligne 27 : la fonction *fsockopen($site,$port)* permet de créer une connexion avec un service TCP / IP travaillant sur le port $*port* de la machine $*site*. Une fois la connexion client / serveur ouverte, de nombreux services TCP / IP échangent des lignes de texte. C'est le cas ici du protocole HTTP (HyperText Transfer Protocol). Le flux du serveur parvenant au client peut alors être traité comme un fichier texte. Il en est de même pour le flux partant du client vers le serveur.
- ligne 38 : la fonction *fputs* permet au client d'envoyer des données au serveur. Ici la ligne de texte envoyée a la signification suivante : "Je veux (GET) la page racine (/) du site web auquel je suis connecté. Je travaille avec le protocole HTTP version 1.0". La version actuelle de ce protocole est 1.1.
- ligne 42 : les lignes de texte de la réponse du serveur peuvent être lues ligne par ligne avec une boucle *while* et enregistrées dans le fichier texte [$site.html]. Lorsque le serveur web a envoyé la page qu'on lui a demandée, il ferme sa connexion avec le client. Côté client, cela sera détecté comme une fin de fichier.

9.3 Un client smtp (inet_03)

Parmi les protocoles TCP / IP, SMTP (SendMail Transfer Protocol) est le protocole de communication du service d'envoi de messages.

Notes :

- sur une machine Windows possédant un antivirus, ce dernier empêchera probablement au script Python de se connecter au port 25 d'un serveur SMTP. Il faut alors désactiver l'antivirus. Pour McAfee par exemple, on peut procéder ainsi :



- en [1], on active la console VirusScan
- en [2], on arrête le service [Protection lors de l'accès]
- en [3], il est arrêté

Le script :

```
1.  <?php
2.
3.  // client SMTP (SendMail Transfer Protocol) permettant d'envoyer un message
4.  // les infos sont prises dans un fichier $INFOS contenant les lignes suivantes
5.  // ligne 1 : smtp, expéditeur, destinataire
6.  // lignes suivantes : le texte du message
7.  // expéditeur: email expéditeur
8.  // destinataire: email destinataire
9.  // smtp: nom du serveur smtp à utiliser
10. // protocole de communication SMTP client-serveur
11. // -> client se connecte sur le port 25 du serveur smtp
12. // <- serveur lui envoie un message de bienvenue
13. // -> client envoie la commande EHLO: nom de sa machine
14. // <- serveur répond OK ou non
15. // -> client envoie la commande mail from: <expéditeur>
16. // <- serveur répond OK ou non
17. // -> client envoie la commande rcpt to: <destinataire>
18. // <- serveur répond OK ou non
19. // -> client envoie la commande data
20. // <- serveur répond OK ou non
21. // -> client envoie ttes les lignes de son message et termine avec une ligne contenant le
22. // seul caractère .
23. // <- serveur répond OK ou non
24. // -> client envoie la commande quit
25. // <- serveur répond OK ou non
26. // les réponses du serveur ont la forme xxx texte où xxx est un nombre à 3 chiffres. Tout nombre xxx >=500
27. // signale une erreur. La réponse peut comporter plusieurs lignes commençant toutes par xxx sauf la dernière
28. // de la forme xxx(espace)
29. // les lignes de texte échangées doivent se terminer par les caractères RC(#13) et LF(#10)
30.
31. // données
32. $INFOS = "mail.txt"; // les paramètres de l'envoi du courrier
33. // on récupère les paramètres du courrier
34. list($erreur, $smtpServer, $expéditeur, $destinataire, $message) = getInfos($INFOS);
35. // erreur ?
36. if ($erreur) {
37.   print "$erreur\n";
38.   exit;
39. }
40. print "Envoi du message [$smtpServer,$expéditeur,$destinataire]\n";
41. // envoi du courrier en mode verbeux
42. $résultat = sendmail($smtpServer, $expéditeur, $destinataire, $message, 1);
43. print "Résultat de l'envoi : $résultat\n";
44. // fin
45. exit;
```

```
46.
47.  //---------------------------------------------------------------------
48.  function getInfos($fichier) {
49.  // rend les informations ($smtp,$expéditeur,$destinataire,$message) prises dans le fichier texte $fichier
50.  // ligne 1 : smtp, expéditeur, destinataire
51.  // lignes suivantes : le texte du message
52.
53.  // ouverture de $fichier
54.    $infos = fopen($fichier, "r");
55.  // le fichier $fichier existe-t-il
56.    if (!$infos)
57.      return array("Le fichier $fichier n'a pu être ouvert en lecture");
58.  // lecture de la 1ère ligne
59.    $ligne = fgets($infos, 1000);
60.  // suppression de la marque de fin de ligne
61.    $ligne = cutNewLineChar($ligne);
62.  // récupération des champs smtp,expéditeur,destinataire
63.    $champs = explode(",", $ligne);
64.  // a-t-on le bon nombre de champs ?
65.    if (count($champs) != 3)
66.      return "La ligne 1 du fichier $fichier (serveur smtp, expéditeur, destinataire) a un
67.  nombre de champs incorrect";
68.  // "traitement" des informations récupérées
69.    for ($i = 0; $i < count($champs); $i++)
70.      $champs[$i] = trim($champs[$i]);
71.  // récupération des champs
72.    list($smtpServer, $expéditeur, $destinataire) = $champs;
73.  // lecture message
74.    $message = "";
75.    while ($ligne = fgets($infos, 1000))
76.      $message.=$ligne;
77.    fclose($infos);
78.  // retour
79.    return array("", $smtpServer, $expéditeur, $destinataire, $message);
80.  }
81.
82.  //---------------------------------------------------------------------
83.
84.  function sendmail($smtpServer, $expéditeur, $destinataire, $message, $verbose) {
85.  // envoie $message au serveur smtp $smtpserver de la part de $expéditeur
86.  // pour $destinataire. Si $verbose=1, fait un suivi des échanges client-serveur
87.  // on récupère le nom du client
88.    $client = gethostbyaddr(gethostbyname(""));
89.  // ouverture d'une connexion sur le port 25 de $smtpServer
90.    $connexion = fsockopen($smtpServer, 25);
91.  // retour si erreur
92.    if (!$connexion)
93.      return "Echec de la connexion au site ($smtpServer,25)";
94.  // $connexion représente un flux de communication bidirectionnel
95.  // entre le client (ce programme) et le serveur smtp contacté
96.  // ce canal est utilisé pour les échanges de commandes et d'informations
97.
98.  // après la connexion le serveur envoie un message de bienvenue qu'on lit
99.    $erreur = sendCommand($connexion, "", $verbose, 1);
100.   if ($erreur) {
101.     fclose($connexion);
102.     return $erreur;
103.   }
104.
105. // cmde ehlo:
106.   $erreur = sendCommand($connexion, "EHLO $client", $verbose, 1);
107.   if ($erreur) {
108.     fclose($connexion);
109.     return $erreur;
110.   }
111.
```

```
112. // cmde mail from:
113.  $erreur = sendCommand($connexion, "MAIL FROM: <$expéditeur>", $verbose, 1);
114.  if ($erreur) {
115.    fclose($connexion);
116.    return $erreur;
117.  }
118.
119. // cmde rcpt to:
120.  $erreur = sendCommand($connexion, "RCPT TO: <$destinataire>", $verbose, 1);
121.  if ($erreur) {
122.    fclose($connexion);
123.    return $erreur;
124.  }
125.
126. // cmde data
127.  $erreur = sendCommand($connexion, "DATA", $verbose, 1);
128.  if ($erreur) {
129.    fclose($connexion);
130.    return $erreur;
131.  }
132.
133. // préparation message à envoyer
134.  // il doit contenir les lignes
135.  // From: expéditeur
136.  // To: destinataire
137.  // ligne vide
138.  // Message
139.  // .
140.  $data = "From: $expéditeur\r\nTo: $destinataire\r\n$message\r\n.\r\n";
141.  $erreur = sendCommand($connexion, $data, $verbose, 0);
142.  if ($erreur) {
143.    fclose($connexion);
144.    return $erreur;
145.  }
146.
147. // cmde quit
148.  $erreur = sendCommand($connexion, "QUIT", $verbose, 1);
149.  if ($erreur) {
150.    fclose($connexion);
151.    return $erreur;
152.  }
153.
154. // fin
155.  fclose($connexion);
156.  return "Message envoyé";
157. }
158.
159. // ------------------------------------------------------------------------
160.
161. function sendCommand($connexion, $commande, $verbose, $withRCLF) {
162. // envoie $commande dans le canal $connexion
163. // mode verbeux si $verbose=1
164. // si $withRCLF=1, ajoute la séquence RCLF à échange
165.
166. // données
167.  if ($withRCLF)
168.    $RCLF = "\r\n"; else
169.    $RCLF="";
170.
171. // envoi cmde si $commande non vide
172.  if ($commande) {
173.    fputs($connexion, "$commande$RCLF");
174.
175. // écho éventuel
176.    if ($verbose)
177.      affiche($commande, 1);
```

```
178.   }//if
179.
180. // lecture réponse
181.   $réponse = fgets($connexion, 1000);
182.
183. // écho éventuel
184.   if ($verbose)
185.     affiche($réponse, 2);
186.
187. // récupération code erreur
188.   $codeErreur = substr($réponse, 0, 3);
189.
190. // dernière ligne de la réponse ?
191.   while (substr($réponse, 3, 1) == "-") {
192.
193.     // lecture réponse
194.     $réponse = fgets($connexion, 1000);
195.
196.     // écho éventuel
197.     if ($verbose)
198.       affiche($réponse, 2);
199.   }//while
200. // réponse terminée
201.
202. // erreur renvoyée par le serveur ?
203.   if ($codeErreur >= 500)
204.     return substr($réponse, 4);
205.
206. // retour sans erreur
207.   return "";
208. }
209.
210. // ------------------------------------------------------------------------
211.
212. function affiche($échange, $sens) {
213. // affiche $échange à l'écran
214. // si $sens=1 affiche -->$echange
215. // si $sens=2 affiche <-- $échange sans les 2 derniers caractères RCLF
216.   switch ($sens) {
217.     case 1:
218.       print "--> [$échange]\n";
219.       return;
220.     case 2:
221.       $L = strlen($échange);
222.       print "<-- [" . substr($échange, 0, $L - 2) . "]\n";
223.       return;
224.   }//switch
225. }
226.
227. // ------------------------------------------------------------------------
228.
229. function cutNewLinechar($ligne) {
230. // on supprime la marque de fin de ligne de $ligne si elle existe
231.   ...
232. }
```

Le fichier *infos.txt* :

```
1.  smtp.orange.fr, serge.tahe@univ-angers.fr , serge.tahe@istia.univ-angers.fr
2.  Subject: test
3.
4.  ligne1
5.  ligne2
6.
7.  ligne3
```

- ligne 1 : [smtp.orange.fr] le serveur utilisé pour l'envoi du courrier, [serge.tahe@univ-angers.fr] l'adresse de l'expéditeur, [serge.tahe@istia.univ-angers.fr] l'adresse du destinataire
- ligne 2 : entêtes du message. Ici il n'y en a qu'un, celui du sujet du message.
- ligne 3 : la ligne vide termine les entêtes du message
- lignes 4-7 : le texte du message

Les **résultats** écran :

```
1.  Envoi du message [smtp.orange.fr,serge.tahe@univ-angers.fr,serge.tahe@univ-angers.fr]
2.  <-- [220 mwinf5d05 ME ESMTP server ready]
3.  --> [EHLO st-PC.home]
4.  <-- [250-mwinf5d05 hello [2.1.22.82], pleased to meet you]
5.  <-- [250-HELP]
6.  <-- [250-AUTH LOGIN PLAIN]
7.  <-- [250-SIZE 44000000]
8.  <-- [250-ENHANCEDSTATUSCODES]
9.  <-- [250-8BITMIME]
10. <-- [250 OK]
11. --> [MAIL FROM: <serge.tahe@univ-angers.fr>]
12. <-- [250 2.1.0 <serge.tahe@univ-angers.fr> sender ok]
13. --> [RCPT TO: <serge.tahe@univ-angers.fr>]
14. <-- [250 2.1.5 <serge.tahe@univ-angers.fr> recipient ok]
15. --> [DATA]
16. <-- [354 enter mail, end with "." on a line by itself]
17. --> [From: serge.tahe@univ-angers.fr
18. To: serge.tahe@univ-angers.fr
19. Subject: test
20.
21. ligne1
22. ligne2
23. ligne3
24. .
25. ]
26. <-- [250 2.0.0 Ag3i1h0051mFoG203g3ip1 mail accepted for delivery]
27. --> [QUIT]
28. <-- [221 2.0.0 mwinf5d05 ME closing connection]
29. Résultat de l'envoi : Message envoyé
```

- ligne 1 : message de suivi d'exécution du script
- ligne 2 : première réponse du serveur smtp. Elle fait suite à la connexion du client sur le port 25 du serveur smtp. Les réponses du serveur sont des lignes de la forme [xxx message] ou [xxx-message]. La première syntaxe indique que la réponse est terminée. xxx est un code de résultat. Une valeur supérieure ou égale à 500 signale une erreur. La seconde syntaxe indique que la réponse n'est pas terminée et qu'une autre ligne va suivre.
- ligne 2 : le serveur smtp indique qu'il est prêt à recevoir des commandes
- ligne 3 : le client envoie la commande [EHLO nomDeMachine] où nomDeMachine est le nom internet de la machine sur laquelle s'exécute le client
- lignes 4-10 : réponse du serveur smtp
- ligne 11 : le client envoie la commande [MAIL FROM: <expéditeur>] qui indique l'adresse mail de l'expéditeur.
- ligne 12 : le serveur smtp indique qu'il accepte cette adresse. Il aurait pu la refuser si sa syntaxe avait été incorrecte. Ceci dit, il ne pas vérifie que l'adresse électronique existe vraiment.
- ligne 13 : le client envoie la commande [RCPT TO: <destinataire>] qui indique l'adresse mail du destinataire du message.
- ligne 14 : le serveur smtp répond qu'il accepte cette adresse. Là encore, une vérification syntaxique est faite.
- ligne 15 : le client envoie la commande [DATA] qui indique à l'utilisateur que les lignes qui vont suivre sont celles du message.
- ligne 16 : le serveur répond que le message peut être envoyé. Celui-ci est une suite de lignes de texte qui doit se terminer par une ligne formée d'un seul caractère, un point.
- lignes 17-25 : le message envoyé par le client
- lignes 17- 19 : les entêtes du message [From:, To:, Subject:] servent à indiquer respectivement l'expéditeur, le destinataire et le sujet du message.
- ligne 20 : ligne vide qui signale la fin des entêtes
- lignes 21-23 : le corps du message
- ligne 24 : la ligne formée d'un unique point qui signale la fin du message.
- ligne 26 : le serveur smtp répond qu'il accepte le message
- ligne 27 : le client envoie la commande [QUIT] pour indiquer qu'il a terminé
- ligne 28 : le serveur smtp lui répond qu'il va fermer la connexion qui le lie au client

Commentaires du code

Nous détaillerons peu le code du script car il a été abondamment commenté.

- lignes 48-80 : la fonction qui exploite le fichier [infos.txt] qui contient le message à envoyer ainsi que les informations nécessaires à cet envoi. Elle rend un tableau ($erreur, $smtpServer, $expéditeur, $destinataire, $messge) avec :
    - $*erreur* : un message d'erreur éventuel, vide sinon.
    - $*smtpServer* : le nom du serveur smtp auquel il faut se connecter
    - $*expéditeur* : l'adresse mail de l'expéditeur
    - $*destinataire* : l'adresse mail du destinataire
    - $*message* : le message à envoyer. Outre le corps du message, il peut y avoir des entêtes.
- lignes 84-157 : la fonction *sendMail* se charge d'envoyer le message. Ses paramètres sont les suivants :
    - $*smtpServer* : le nom du serveur smtp auquel il faut se connecter
    - $*expéditeur* : l'adresse mail de l'expéditeur
    - $*destinataire* : l'adresse mail du destinataire
    - $*message* : le message à envoyer.
    - $*verbose* : à 1 indique que les échanges avec le serveur smtp doivent être reproduits sur la console.

  La fonction *sendMail* rend un message d'erreur, vide s'il n'y a pas eu d'erreur.
- ligne 88 : permet d'obtenir le nom windows d'un ordinateur opérant avec l'OS windows.
- ligne 99 : nous avons vu que le dialogue client / serveur était une suite de la forme :
    - envoi par le client d'une commande sur une ligne
    - réponse du serveur sur une ou plusieurs lignes

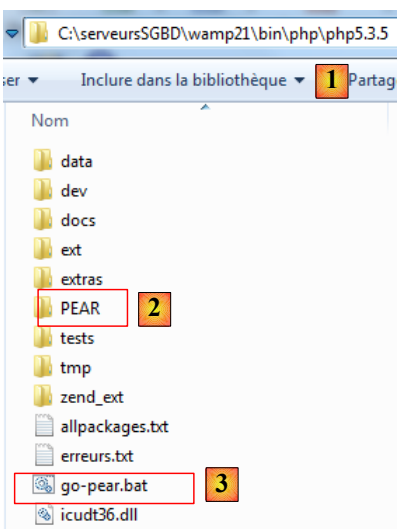
  La fonction *sendCommand* permet de faire un échange client / serveur. Elle rend un message d'erreur qui est vide s'il n'y a pas d'erreur.
- lignes 161-208 : la fonction *sendCommand* admet les paramètres suivants :
    - $connexion : le canal TCP / IP qui lie le client au serveur
    - $commande : la commande à envoyer sur ce canal. La réponse du serveur à cette commande sera lue.
    - $verbose : à 1 indique que les échanges avec le serveur smtp doivent être reproduits sur la console.
    - $withRCLF : à 1 indique qu'il faut ajouter la marque de fin de ligne "\r\n" à la fin de la commande
- ligne 173 : envoi de la commande par le client
- ligne 181 : lecture de la première ligne de la réponse du serveur smtp de la forme xxx texte ou xxx-texte. Ce dernier cas indique que le serveur a une autre ligne à envoyer. xxx est le code d'erreur envoyé par le serveur.
- ligne 188 – récupération du code d'erreur de la réponse
- lignes 191-199 : lecture des autres lignes de la réponse
- lignes 203-204 : si le code d'erreur est >=500, alors c'est que le serveur smtp signale une erreur.

9.4 Un second programme d'envoi de mail (inet_04)

Ce script a la même fonctionnalité que le précédent : envoyer un mail. Nous utilisons pour cela des modules de la bibliothèque PEAR. Cette bibliothèque comporte des dizaines de modules couvrant différents domaines. Nous allons utiliser les suivants :

- *Net/SMTP* : un module permettant de dialoguer avec un serveur SMTP
- *Mail* : un module permettant de gérer l'envoi d'un mail selon différents protocoles.
- *Mail/Mime* : un module permettant de créer un message qui peut comporter des documents attachés.

Pour disposer de ces modules, il faut d'abord les installer sur la machine exécutant le script Php. L'installation du paquetage logiciel *WampServer* a installé un interpréteur PHP. Dans l'arborescence de celui-ci, il est possible d'installer des modules PEAR.



- en [1], le dossier d'installation de l'interpréteur Php
- en [2], le dossier PEAR qui contiendra les modules PEAR qui vont être installés
- en [3], le fichier de commandes [go-pear.bat] qui initialise la bibliothèque PEAR

Pour initialiser la bibliothèque PEAR, on ouvre une fenêtre DOS et on exécute le fichier [go-pear.bat]. Ce script va se connecter au site internet de la bibliothèque PEAR. Il faut donc une connexion internet.

```
C:\serveursSGBD\wamp21\bin\php\php5.3.5>go-pear.bat
```

Connecté au site internet de la bibliothèque PEAR, le script va télécharger un certain nombre d'éléments. Parmi ceux-ci, un nouveau script [pear.bat]. C'est avec ce script qu'on va installer les différents modules PEAR dont nous avons besoin. Ce script s'appelle avec des arguments. Parmi ceux-ci l'argument [help] permet d'avoir une liste des commandes acceptées par le script :

```
C:\serveursSGBD\wamp21\bin\php\php5.3.5>pear help
Commands:
build                Build an Extension From C Source
bundle               Unpacks a Pecl Package
channel-add          Add a Channel
channel-alias        Specify an alias to a channel name
channel-delete       Remove a Channel From the List
channel-discover     Initialize a Channel from its server
channel-info         Retrieve Information on a Channel
channel-login        Connects and authenticates to remote channel server
channel-logout       Logs out from the remote channel server
channel-update       Update an Existing Channel
clear-cache          Clear Web Services Cache
config-create        Create a Default configuration file
config-get           Show One Setting
config-help          Show Information About Setting
config-set           Change Setting
config-show          Show All Settings
convert              Convert a package.xml 1.0 to package.xml 2.0 format
cvsdiff              Run a "cvs diff" for all files in a package
cvstag               Set CVS Release Tag
download             Download Package
download-all         Downloads each available package from the default channel

info                 Display information about a package
install              Install Package
list                 List Installed Packages In The Default Channel
list-all             List All Packages
list-channels        List Available Channels
list-files           List Files In Installed Package
list-upgrades        List Available Upgrades
login                Connects and authenticates to remote server [Deprecated i
n favor of channel-login]
logout               Logs out from the remote server [Deprecated in favor of c
hannel-logout]
makerpm              Builds an RPM spec file from a PEAR package
package              Build Package
package-dependencies Show package dependencies
package-validate     Validate Package Consistency
```

```
pickle                  Build PECL Package
remote-info             Information About Remote Packages
remote-list             List Remote Packages
run-scripts             Run Post-Install Scripts bundled with a package
run-tests               Run Regression Tests
search                  Search remote package database
shell-test              Shell Script Test
sign                    Sign a package distribution file
svntag                  Set SVN Release Tag
uninstall               Un-install Package
update-channels         Update the Channel List
upgrade                 Upgrade Package
upgrade-all             Upgrade All Packages [Deprecated in favor of calling upgr
ade with no parameters]
Usage: pear [options] command [command-options] <parameters>
Type "pear help options" to list all options.
Type "pear help shortcuts" to list all command shortcuts.
Type "pear help <command>" to get the help for the specified command.
```

La commande [install] permet d'installer des modules PEAR. On peut demander de l'aide sur la commande [install] :

```
C:\serveursSGBD\wamp21\bin\php\php5.3.5>pear help install
pear install [options] [channel/]<package> ...
Installs one or more PEAR packages.  You can specify a package to install in four ways:

"Package-1.0.tgz" : installs from a local file

"http://example.com/Package-1.0.tgz" : installs from anywhere on the net.

"package.xml" : installs the package described in package.xml.  Useful for testing, or for wrapping a
PEAR package in another package manager such as RPM.

"Package[-version/state][.tar]" : queries your default channel's server(pear.php.net) and downloads the
newest package with the preferred quality/state (stable).

To retrieve Package version 1.1, use "Package-1.1," to retrieve Package state beta, use "Package-beta."
To retrieve an uncompressed file, append .tar (make sure there is no file by the same name first)

To download a package from another channel, prefix with the channel name like "channel/Package"

More than one package may be specified at once.  It is ok to mix these four ways of specifying packages.

Options:
  -f, --force
        will overwrite newer installed packages
  -l, --loose
        do not check for recommended dependency version
  -n, --nodeps
        ignore dependencies, install anyway
  -r, --register-only
        do not install files, only register the package as installed
  -s, --soft
        soft install, fail silently, or upgrade if already installed
  -B, --nobuild
        don't build C extensions
  -Z, --nocompress
        request uncompressed files when downloading
  -R DIR, --installroot=DIR
        root directory used when installing files (ala PHP's INSTALL_ROOT), use packagingroot for RPM
  -P DIR, --packagingroot=DIR
        root directory used when packaging files, like RPM packaging
  --ignore-errors
        force install even if there were errors
  -a, --alldeps
        install all required and optional dependencies
  -o, --onlyreqdeps
        install all required dependencies
  -O, --offline
        do not attempt to download any urls or contact channels
  -p, --pretend
        Only list the packages that would be downloaded
```

Les modules PEAR à installer sont les suivants : [Mail], [Mail_Mime], [Net_SMTP]. Dans la fenêtre Dos, on tapera successivement les commandes suivantes :

```
<php_installDir>pear install Mail
...
<php_installDir>pear install Mail_Mime
…
<php_installDir>pear install Net_SMTP
```

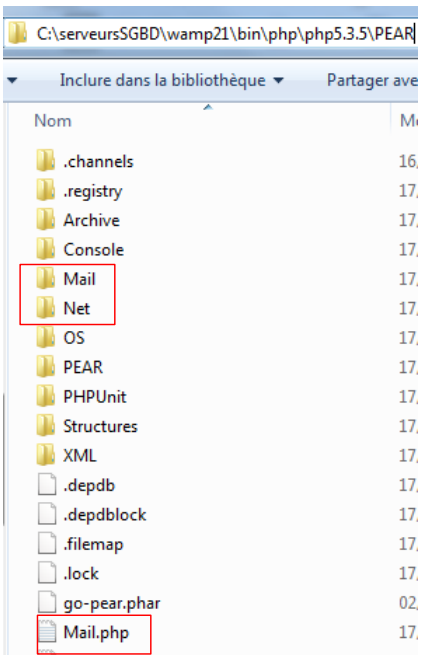
```
...
```

où <php_installDir> est le dossier d'installation de l'interpréteur Php (C:\serveursSGBD\wamp21\bin\php\php5.3.5 dans cet exemple)

On peut voir les modules installés :

```
C:\serveursSGBD\wamp21\bin\php\php5.3.5>pear list
INSTALLED PACKAGES, CHANNEL PEAR.PHP.NET:
=========================================
PACKAGE           VERSION STATE
Archive_Tar       1.3.7   stable
Console_Getopt    1.3.1   stable
Mail              1.2.0   stable
Mail_Mime         1.8.1   stable
Net_SMTP          1.6.0   stable
Net_Socket        1.0.10  stable
PEAR              1.9.4   stable
PHPUnit           1.3.2   stable
Structures_Graph  1.0.4   stable
XML_Util          1.2.1   stable
```

Les modules sont installés dans le dossier <php_installDir>/PEAR :



Avec les modules PEAR installés, le script Php d'envoi de mail devient le suivant :

```php
<?php

// gestion des erreurs
ini_set("display_errors", "off");
// modules
ini_set("include_path", ".;C:\serveursSGBD\wamp21\bin\php\php5.3.5\PEAR");
require_once "Mail.php";
require_once "Mail/Mime.php";
require_once "Net/SMTP.php";


// client SMTP (SendMail Transfer Protocol) permettant d'envoyer un message
// les infos sont prises dans un fichier $INFOS contenant les lignes suivantes
// ligne 1 : smtp, expéditeur, destinataire, attachement
// lignes suivantes : le texte du message
// expéditeur:email expéditeur
// destinataire: email destinataire
```

```php
// smtp: nom du serveur smtp à utiliser
// attachement : nom du document à attacher
//
// données
$INFOS = "mail2.txt"; // les paramètres de l'envoi du courrier
// on récupère les paramètres du courrier
list($erreur, $smtpServer, $expéditeur, $destinataire, $message, $sujet, $attachement) = getInfos($INFOS);
// erreur ?
if ($erreur) {
  print "$erreur\n";
  exit;
}
print "Envoi du message [$smtpServer,$expéditeur,$destinataire,$sujet, $attachement]\n";
// envoi du courrier en mode verbeux
$résultat = sendmail($smtpServer, $expéditeur, $destinataire, $message, $sujet, $attachement);
print "Résultat de l'envoi : $résultat\n";
// fin
exit;

//---------------------------------------------------------------------
function getInfos($fichier) {
// rend les informations ($smtp,$expéditeur,$destinataire,$message, $sujet, $attachement) prises dans le fichier texte $fichier
// ligne 1 : smtp, expéditeur, destinataire, sujet, attachement
// lignes suivantes : le texte du message
...
// retour
  return array("", $smtpServer, $expéditeur, $destinataire, $message, $sujet, $attachement);
}

//getInfos
//---------------------------------------------------------------------

function sendmail($smtpServer, $expéditeur, $destinataire, $message, $sujet, $attachement) {
// envoie $message au serveur smtp $smtpserver de la part de $expéditeur
// pour $destinataire. Le document $attachement est joint au message
// le message a le sujet $sujet
//
  // message
  $msg = new Mail_Mime();
  $msg->setTXTBody($message);
  $msg->addAttachment($attachement);
  $headers = $msg->headers(array("From" => $expéditeur, "To" => $destinataire, "Subject" => $sujet));
  // envoi
  $mailer = &Mail::factory("smtp", array("host" => $smtpServer, "port" => 25));
  $envoi = $mailer->send($expéditeur, $headers, $msg->get());
  if ($envoi === TRUE) {
    return "Message envoyé";
  } else {
    return $envoi;
  }
}

// ---------------------------------------------------------------------

function cutNewLinechar($ligne) {
// on supprime la marque de fin de ligne de $ligne si elle existe
  …
}
```

Le fichier [mail2.txt] :

```
smtp.orange.fr, serge.tahe@univ-angers.fr , serge.tahe@univ-angers.fr, test, document.pdf
ligne1
ligne2
ligne3
```

- ligne 1 : dans l'ordre, le serveur SMTP, l'adresse de l'expéditeur, l'adresse du destinataire, le sujet du message, le document à attacher.
- lignes 2-4 : le texte du message

Résultats écran

```
Envoi du message [smtp.orange.fr,serge.tahe@univ-angers.fr,serge.tahe@univ-angers.fr,test, document.pdf]
Résultat de l'envoi : Message envoyé
```

Commentaires

Le script ne diffère du précédent que par sa fonction *sendmail*. Nous décrivons celle-ci :

- ligne 6 : les script Php des modules PEAR ont été installés dans un répertoire qui par défaut n'est pas exploré par l'interpréteur Php. Afin que celui-ci trouve les modules PEAR, nous définissons nous-mêmes les dossiers que doit explorer l'interpréteur Php. Il est possible de modifier à l'exécution certains paramètres de configuration de l'interpréteur Php. Cette modification n'est visible que du script qui la fait et seulement pendant la durée de son exécution. La configuration par défaut de l'interpréteur Php peut être trouvée dans le fichier *<php_installdir>/php.ini*. Ce fichier contient des lignes de la forme
  clé=valeur
  La valeur de la clé peut être modifiée à l'exécution par la fonction *ini_set(clé, nouvelle_valeur)*. La clé pour spécifier le chemin de recherche par l'interpréteur des fonctions et classes Php référencées par le script est '*include_path*'. Ici, nous mettons dans le chemin de recherche à la fois le dossier du script (.) et le dossier PEAR du dossier d'installation de l'interpréteur Php.
- lignes 7-9 : les scripts Php d'envoi de mail sont chargés.
- ligne 50 : la fonction *sendmail* qui se charge d'envoyer le courrier. Ses paramètres sont les suivants :

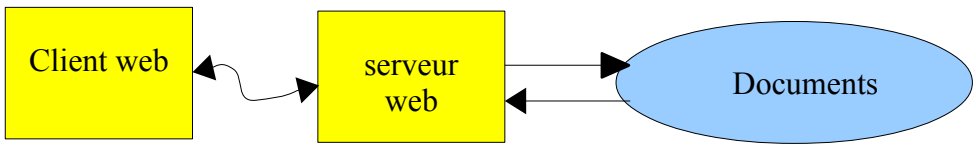
  - $*smtpServer* : le nom du serveur smtp auquel il faut se connecter
  - $*expéditeur* : l'adresse mail de l'expéditeur
  - $*destinataire* : l'adresse mail du destinataire
  - $*message* : le message à envoyer.
  - $*sujet* : le sujet du message
  - $*attachement* : le nom du document à attacher au message

  La fonction *sendMail* rend un message d'erreur, vide s'il n'y a pas eu d'erreur.
- ligne 56 : création d'un message de type *Mail_Mime*. Ce message est composé d'entêtes (From, To, Subject) et d'un corps (le message lui-même)
- ligne 57 : on fixe le corps du message *Mail_Mime*.
- ligne 58 : on attache un document au message
- ligne 59 : on fixe les entêtes (From, To, Subject) du message *Mail_Mime*
- ligne 61 : on crée la classe chargée d'envoyer le message *Mail_Mime*. Le premier paramètre de la méthode est le nom du protocole à utiliser, ici le protocole SMTP. Le second paramètre est un tableau fixant le nom et le port du service smtp à utiliser
- ligne 62 : envoi du message. La méthode send reçoit trois paramètres : l'adresse de l'expéditeur, les entêtes du message *Mail_Mime*, le corps du message *Mail_Mime*. La fonction send renvoie le booléen TRUE si l'envoi s'est bien passé, un message d'erreur sinon.

10 Des serveurs en PHP

Les programmes PHP pouvant être exécutés par un serveur WEB, un tel programme devient un programme serveur pouvant servir plusieurs clients. Du point de vue du client, appeler un service WEB revient à demander l'URL de ce service. Le client peut être écrit avec n'importe quel langage, notamment en PHP. Dans ce dernier cas, on utilise alors les fonctions réseau que nous venons de voir. Il nous faut par ailleurs savoir "converser" avec un service WEB, c'est à dire comprendre le protocole *http* de communication entre un serveur Web et ses clients. C'est le but des programmes qui suivent.

Le client web décrit au paragraphe 9.2, page 64, nous a permis de découvrir une partie du protocole HTTP.

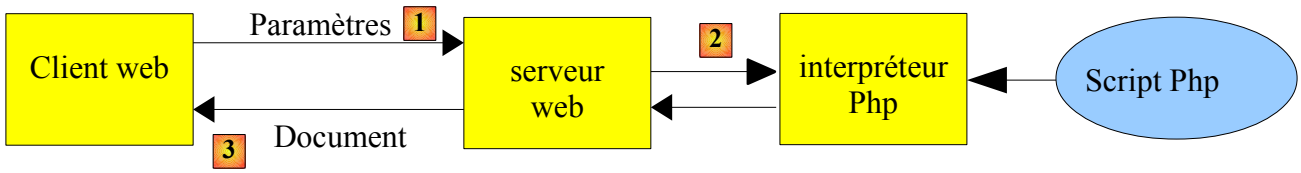


Dans leur version la plus simple, les échanges client / serveur sont les suivants :

- le client ouvre une connexion avec le port 80 du serveur web
- il fait une requête concernant un document
- le serveur web envoie le document demandé et ferme la connexion
- le client ferme à son tour la connexion

Le client peut être de nature diverse : un texte au format Html, une image, une vidéo, ... Ce peut être un document existant (document statique) ou bien un document généré à la volée par un script (document dynamique). Dans ce dernier cas, on parle de programmation web. Le script de génération dynamique de documents peut être écrit dans divers langages : Php, Python, Perl, Java, Ruby, C#, VB.net, ...

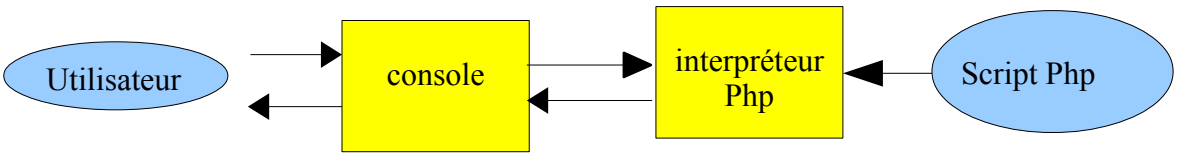
Nous utilisons ici Php pour générer dynamiquement des documents texte.



- en [1], le client ouvre une connexion avec le serveur, demande un script Php, envoie ou non des paramètres à destination de ce script
- en [2], le serveur web fait exécuter le script Php par l'interpréteur Php. Ce script génère un document qui est envoyé au client [3]
- le serveur clôt la connexion. Le client en fait autant.

Le serveur web peut traiter plusieurs clients à la fois. Avec le paquetage logiciel WampServer, le serveur web est un serveur Apache, un serveur open source de l'Apache Foundation (http://www.apache.org/). Dans les applications qui suivent, WampServer doit être lancé. Cela active trois logiciels : le serveur web Apache, le Sgbd MySQL, l'interpréteur Php.

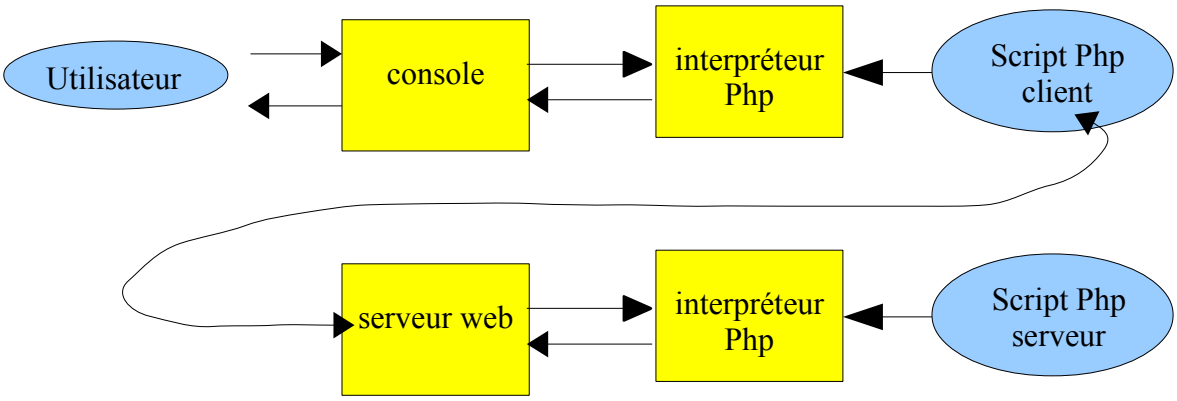
Les scripts exécutés par le serveur web seront écrits avec l'outil Netbeans. Nous avons écrit jusqu'à maintenant des scripts Php exécutés dans un contexte console :



L'utilisateur utilise la console pour demander l'exécution d'un script Php et en recevoir les résultats.

Dans les applications client /serveur qui vont suivre,

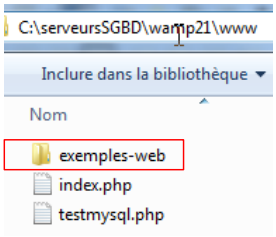
- le script du client est exécuté dans un contexte console
- le script du serveur est exécuté dans un contexte web



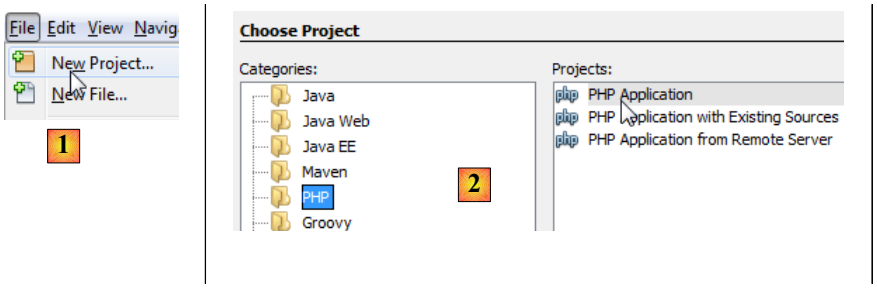
Le script Php du serveur ne peut pas être n'importe où dans le système de fichiers. En effet, le serveur web cherche dans des endroits précisés par configuration, les documents statiques et dynamiques qu'on lui demande. La configuration par défaut de WampServer fait que les documents sont cherchés dans le dossier *<WampServer>/www* où *<WampServer>* est le dossier d'installation

de WampServer. Ainsi un client web demande un document D avec l'Url [http://localhost/D], le serveur web lui servira le document D de chemin [<WampServer>/www/D].

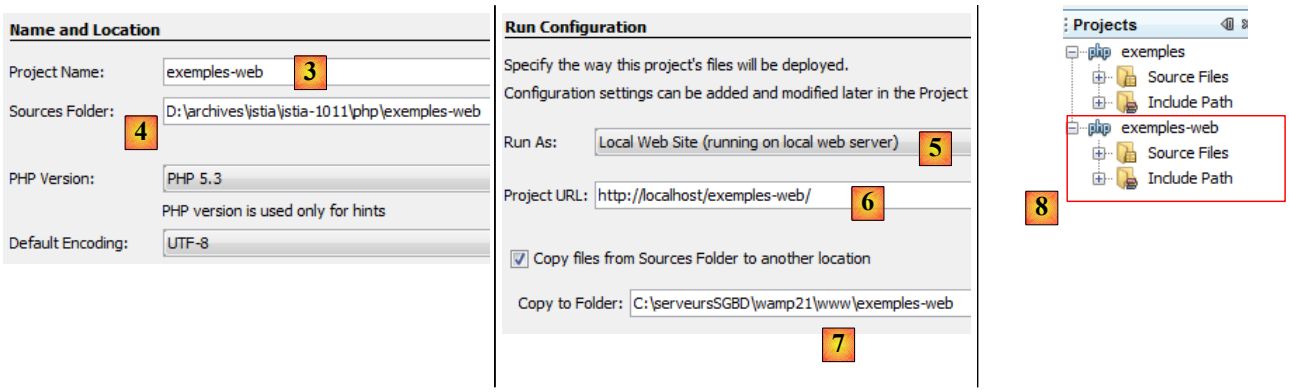
Dans les exemples qui suivent, nous mettrons les scripts serveur dans le dossier [www/exemples-web]. Si un script serveur s'appelle *S.php*, il sera demandé au serveur web avec l'Url [*http://localhost/exemples-web/S.php*]. Le document [*<WampServer>/www/exemples-web/S.php*] lui sera alors servi.



Pour créer un script serveur avec Netbeans, nous procéderons de la façon suivante :

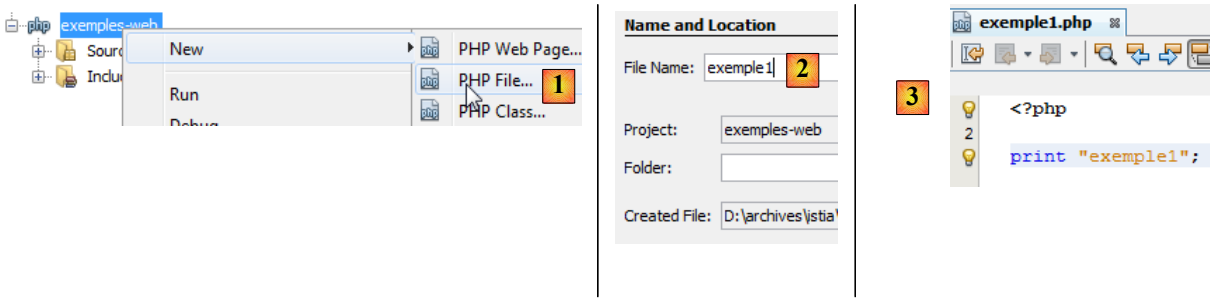


- en [1], nous créons un nouveau projet
- en [2], nous prenons la catégorie [PHP] et le projet [PHP Application]



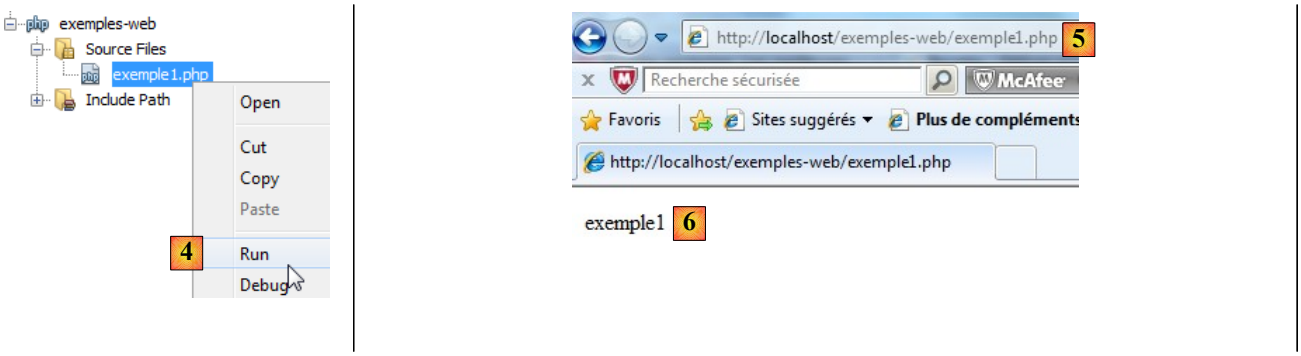
- en [3], nous donnons un nom au projet
- en [4], nous choisissons un dossier pour le projet
- en [5], nous indiquons que le script doit être exécuté par un serveur web local (l'Url du script sera de la forme http://localhost/...). Le serveur web local sera le serveur web Apache de WampServer.
- en [6], nous donnons l'Url du projet. Ici, nous décidons qu'un script S.php du projet sera demandé avec l'Url [http://localhost/exemples-web/S.php]. D'après ce qui a été dit, cela veut dire que le chemin du script S.php dans le système de fichiers sera [<WampServer>/www/exemples-web/S.php]. C'est ce qui est indiqué en [7]. Nous demandons ici que tout script S.php du projet soit recopié dans l'arborescence du serveur web Apache.
- en [8], le nouverau projet.

Ecrivons un script de test :



- en [1], nous créons dans le projet [exemples-web] un premier script Php
- en [2], nous lui donnons un nom
- en [3], après l'avoir créé, nous lui donnons le contenu suivant

Pour la suite, il faut que WampServer soit lancé.



- en [4], nous exécutons le script web [exemple1.php]. Netbeans va alors lancer le navigateur par défaut de la machine et va lui demander d'afficher l'Url [http://localhost/exemples-web/exemple1.php] [5]
- en [6], le navigateur affiche ce que le script serveur a écrit au client.

Par la suite, nous rencontrerons deux types de clients web :

- un navigateur comme ci-dessus. Nous avons écrit que le serveur web envoyait une réponse de la forme : *entêtes Http, ligne vide, texte*. Le navigateur n'affiche que *texte*.
- un script Php qui lui affichera la totalité de la réponse *entêtes Http, ligne vide, texte*.

Dans la suite,
- les scripts serveur seront écrits comme [exemple1.php] ci-dessus
- les scripts client seront écrits comme les scripts console que nous avons écrits jusqu'à maintenant.

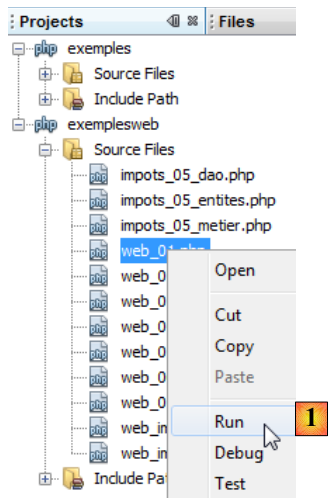
10.1 Application client/ serveur de date/heure

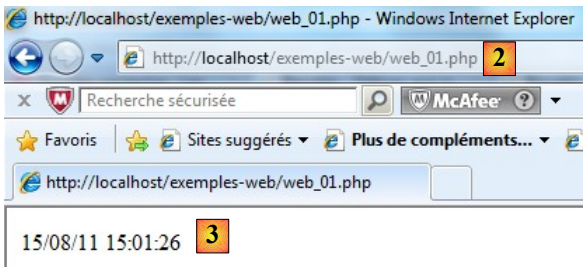
10.1.1 Le serveur (web_01)

```
1. <?php
2. // time : nb de millisecondes depuis 01/01/1970
3. // format affichage date-heure
4. // d: jour sur 2 chiffres
5. // m: mois sur 2 chiffres
6. // y : année sur 2 chiffres
7. // H : heure 0,23
8. // i : minutes
9. // s: secondes
10. print date("d/m/y H:i:s",time());
```

Basiquement, le script Php ci-dessus écrit l'heure courante sur l'écran. Cependant, lorsqu'il est exécuté par un serveur web, le flux n° 1 qui est habituellement associé à l'écran est redirigé vers la connexion qui lie le serveur à son client. Donc, dans un contexte web, le script ci-dessus envoie l'heure courante sous forme de texte au client.

Exécutons ce script au sein de Netbeans :



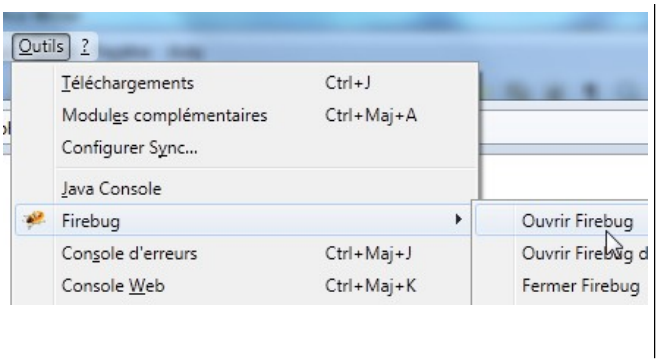


- en [1], on exécute le script. Un navigateur web est alors lancé.
- en [2], l'Url demandée par le navigateur web
- en [3], le texte envoyé par le script serveur

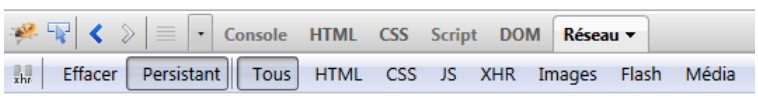
Le navigateur client utilise le protocole Http pour dialoguer avec le serveur web. Nous avons déjà décrit la forme de ce protocole.

Le client envoie des lignes de texte qu'on peut décomposer en trois parties : *entêtes Http, ligne vide, document*. Le document envoyé au serveur web est le plus souvent vide ou bien c'est un ensemble de paramètres *parami=vali* où *vali* est une valeur saisie par l'utilisateur dans un formulaire Html.
La réponse du serveur a la même forme : *entêtes Http, ligne vide, document* où *document* est cette fois le document demandé par le navigateur client. Si le client a transmis des paramètres, le document délivré dépend en général de ces paramètres.

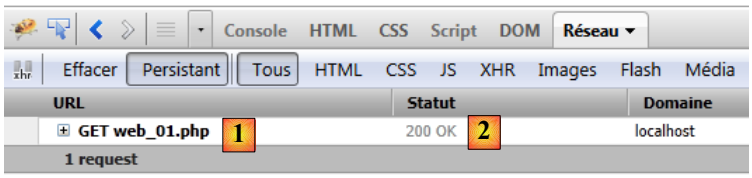
Avec le navigateur **Firefox**, il est possible de découvrir les échanges réels entre client et serveur web. Il existe un plugin pour *Firefox*, appelé **Firebug** qui permet de tracer ces échanges. Firebug est disponible à l'Url [https://addons.mozilla.org/fr/firefox/addon/firebug/]. Si on utilise le navigateur *Firefox* pour visiter cette Url, on peut alors télécharger le plugin *Firebug*. Nous supposons dans la suite que le plugin Firebug a été téléchargé et installé. Il est disponible via une option du menu de Firefox :



Une fenêtre Firebug s'ouvre dans la fenêtre du navigateur Firefox. Cette fenêtre présente elle-même un menu :



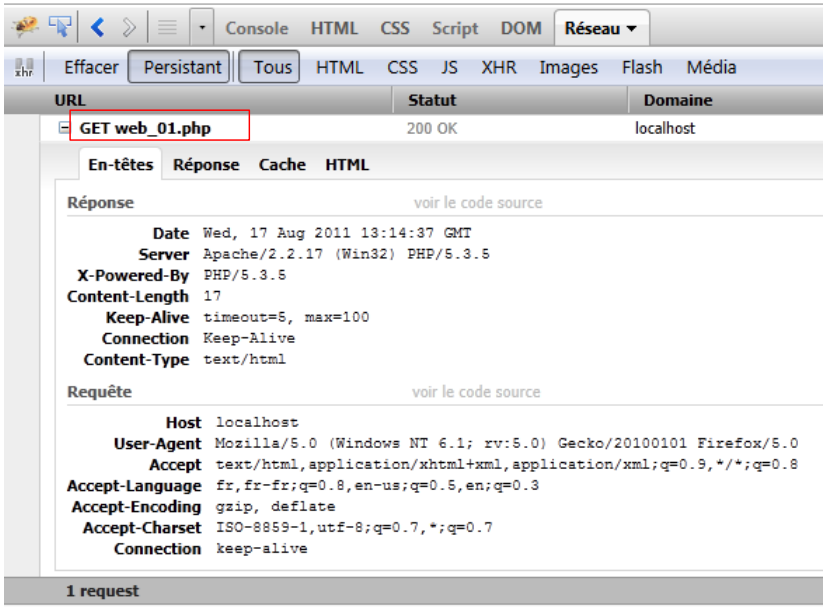
Pour découvrir les échanges client / serveur lors d'une requête Http, nous demandons l'Url [http://localhost/exemples-web/web_01.php] avec le navigateur Firefox. La fenêtre de Firebug se remplit alors d'informations :

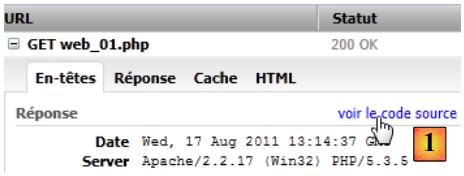


Ci-dessus, on a un résumé des échanges client / serveur :

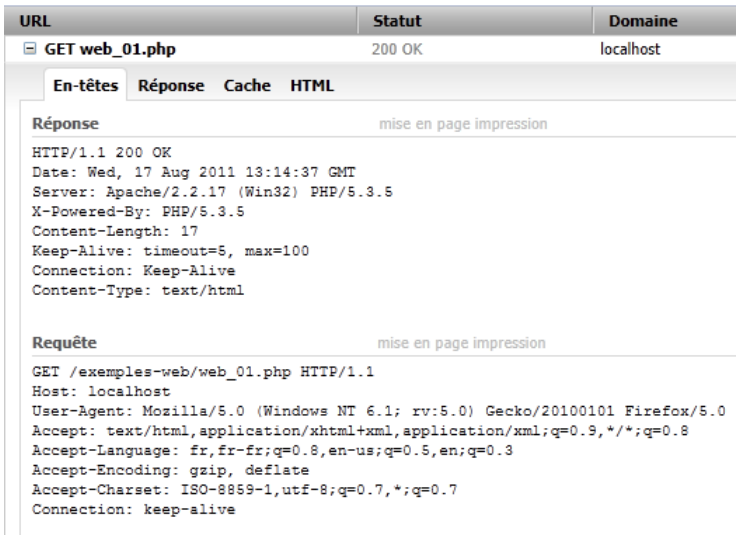
- [1] : le client a envoyé la commande HTTP **: GET /exemples-web/web_01.php HTTP/1.1** pour demander le document [web01.php]
- [2] : le serveur a envoyé la réponse : **HTTP/1.1 200 OK** indiquant qu'il a trouvé le document demandé.

Firebug permet d'obtenir les échanges complets. Il suffit de "déplier" l'Url :





Nous voyons ci-dessus les en-têtes Http échangés entre le client (Requête) et le serveur (Réponse). Il est possible d'avoir le code source des échanges, c.a.d. les lignes de texte réellement échangées [1]. Nous obtenons alors le code source suivant :

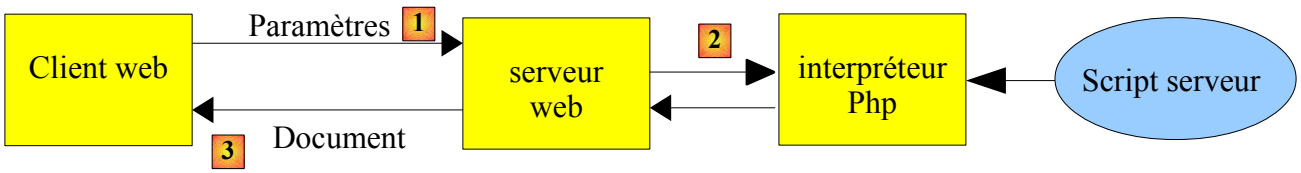


Pour écrire un script client du serveur web, il nous suffit de reproduire le comportement du navigateur. Après avoir ouvert une connexion avec le serveur, le script client pourrait envoyer les 8 lignes de la requête ci-dessus. En fait tout n'est pas indispensable et nous n'enverrons que les trois lignes suivantes :

```
1.  GET /exemples-web/web01.php HTTP/1.1
2.  Host: localhost
3.  Connection: close
```

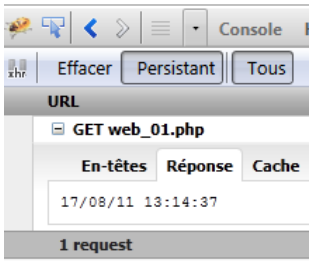
- ligne 1 : précise le document demandé et le protocole Http utilisé
- ligne 2 : donne le nom de la machine du script client
- ligne 3 : indique qu'après l'échange, le client fermera la connexion au serveur

Intéressons-nous maintenant à la réponse du serveur. Nous savons qu'elle a été produite par le script Php [web_01.php]. Ci-dessus, nous voyons les entêtes Http de la réponse. Le code du script [web01.php] montre que ce n'est pas lui qui les a générés. Rappelons-nous la configuration du script serveur :



C'est le serveur web qui a généré les entêtes Http de la réponse. Le script serveur peut les générer lui-même. Nous en verrons un exemple un peu plus loin.

Nous avons dit que la réponse du serveur web était de la forme : *entêtes Http, ligne vide, document*. Si le document est un document texte, on peut voir celui-ci dans l'onglet [Réponse] de Firebug :



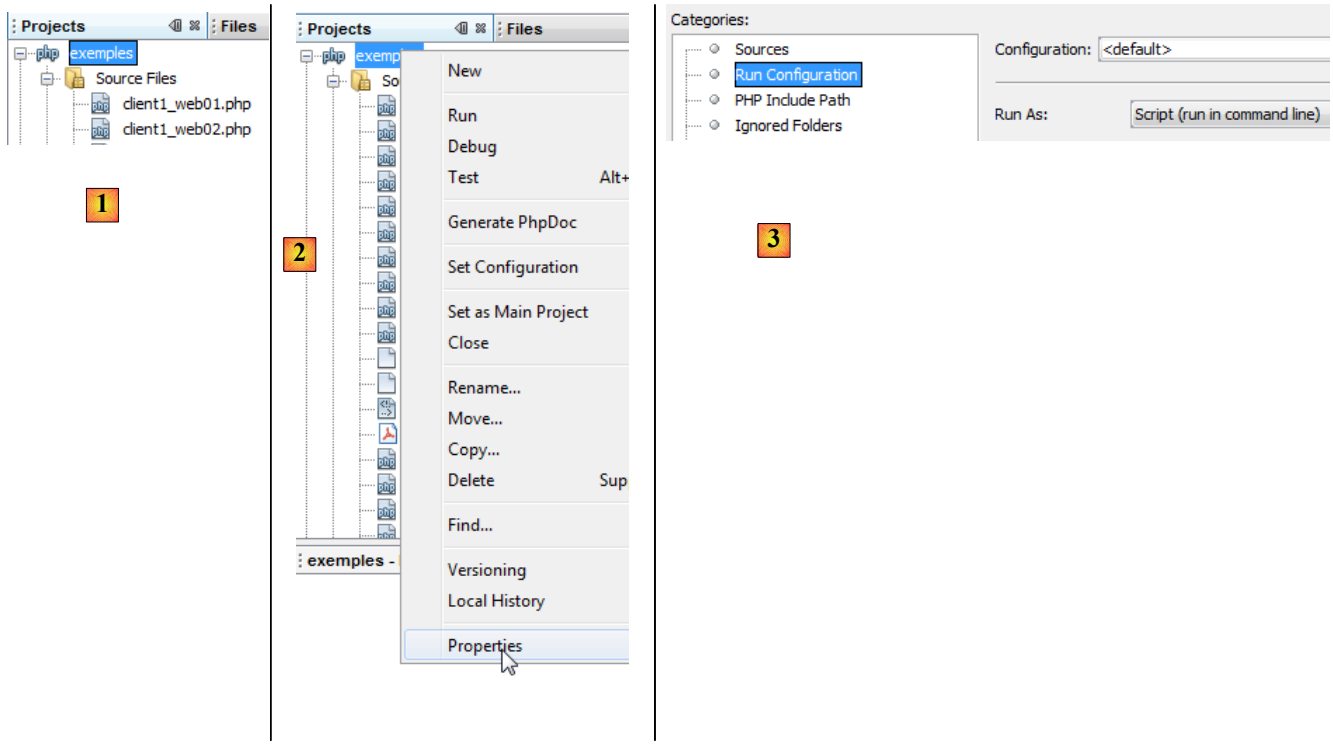
Cette réponse a été générée par le script [web_01.php].

10.1.2 Un client (client1_web_01)

Nous écrivons maintenant un script client pour le service précédent. Nous savons que le client doit :

- ouvrir une connexion avec le serveur web
- envoyer le texte : *entêtes Http, ligne vide*
- lire la réponse complète du serveur jusqu'à ce que celui-ci ferme sa connexion avec le client
- fermer sla connexion avec le serveur

Le script client s'exécute dans un environnement console de Netbeans :



- en [1], le script client [client1_web_01.php] est inclus dans le projet Netbeans [exemples]
- en [2], les propriétés du projet Netbeans [exemples]
- en [3], le projet Netbeans [exemples] s'exécute en mode "ligne de commande", ce que nous avons appelé aussi le mode "console".

Le code du script client est le suivant :

```
1.  <?php
2.
3.  // données
4.  $HOTE = "localhost";
5.  $PORT = 80;
6.  $urlServeur = "/exemples-web/web_01.php";
7.  // ouverture d'une connexion sur le port 80 de $HOTE
8.  $connexion = fsockopen($HOTE, $PORT);
9.  // erreur ?
10. if (!$connexion) {
11.   print "Erreur : $erreur\n";
12.   exit;
13. }
14. // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
15. // GET
16. fputs($connexion, "GET $urlServeur HTTP/1.1\n");
17. // Host
18. fputs($connexion, "Host: localhost\n");
19. // Connection
20. fputs($connexion,"Connection: close\n");
21. // ligne vide
```

```
22. fputs($connexion,"\n");
23. // le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
24. // ses données puis fermer le canal. Le client lit donc tout ce qui arrive de $connexion
25. // jusqu'à la fermeture du canal
26. while ($ligne = fgets($connexion, 1000)) {
27.   print "$ligne";
28. }//while
29. // le client ferme la connexion à son tour
30. fclose($connexion);
31. // fin
32. exit;
```

Commentaires

- ligne 8 : ouverture d'une connexion vers le serveur
- ligne 16 : commande Http GET
- ligne 18 : commande Http Host
- ligne 20 : commande Http Connection
- ligne 22 : ligne vide
- lignes 26-28 : lecture de toutes les lignes de texte envoyées par le serveur jusqu'à ce qu'il ferme la connexion.
- ligne 30 : le client ferme à son tour la connexion

Résultats

L'exécution du script client produit les résultats suivants :

```
1. HTTP/1.1 200 OK
2. Date: Wed, 17 Aug 2011 13:35:00 GMT
3. Server: Apache/2.2.17 (Win32) PHP/5.3.5
4. X-Powered-By: PHP/5.3.5
5. Content-Length: 17
6. Connection: close
7. Content-Type: text/html
8.
9. 17/08/11 13:35:00
```

Commentaires

- lignes 1-7 : la réponse Http du serveur web.
- ligne 8 : la ligne vide qui signale la fin des entêtes Http
- lignes 9 et au-delà : le document. Ici, c'est un simple texte représentant la date et l'heure courante. C'est le texte écrit par le script Php sur la sortie n° 1.

- ligne 1 : le serveur répond qu'il a trouvé le document demandé.
- ligne 2 : date et heure courantes du serveur
- ligne 3 : identité du serveur web
- ligne 4 : indique que le document qui va suivre est un document généré par un script Php
- ligne 5 : nombre de caractères du document
- ligne 6 : le serveur indique qu'après avoir envoyé le document, il va fermer la connexion
- ligne 7 : indique que le document envoyé par le serveur est un texte au format Html. C'est inexact ici. Le document est du texte sans format particulier. Lorsque le document n'est pas du texte au format Html, c'est au script Php de l'indiquer. Nous ne l'avons pas fait ici.

10.1.3 Un deuxième client (client2_web_01)

Le client précédent affichait tout ce que lui envoyait le serveur web. Dans la pratique, on ignore généralement les entêtes Http de la réponse et on exploite la partie document. Nous cherchons ici à récupérer la date et l'heure envoyées par le script PHP serveur. Nous allons récupérer ces informations au moyen d'une expression régulière.

```
1. <?php
2.
3. // récupérer les informations envoyées par un serveur web
4. // données
5. $HOTE = "localhost";
6. $PORT = 80;
7. $urlServeur = "/exemples-web/web_01.php";
8. // ouverture d'une connexion sur le port 80 de $HOTE
```

```php
$connexion = fsockopen($HOTE, $PORT);
// erreur ?
if (!$connexion) {
  print "Erreur : $erreur\n";
  exit;
}
// les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
// GET
fputs($connexion, "GET $urlServeur HTTP/1.1\n");
// Host
fputs($connexion, "Host: localhost\n");
// Connection
fputs($connexion,"Connection: close\n");
// ligne vide
fputs($connexion,"\n");
// le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
// ces données puis fermer le canal. Le client lit donc tout ce qui arrive de $connexion
// jusqu'à trouver la ligne qu'il cherche de la forme jj/mm/aa hh:mm:ss
while ($ligne = fgets($connexion, 1000)) {
  print "$ligne";
  if (preg_match("/(\d\d)\/(\d\d)\/(\d\d) (\d\d):(\d\d):(\d\d)/", $ligne, $champs)) {
    // on récupère les # champs
    array_shift($champs); // enlève le 1er élément du tableau champs
    // on récupère les 6 champs dans 6 variables
    list($j, $m, $a, $h, $i, $s) = $champs;
    // affichage résultat
    print "\ndateheure=[$j,$m,$a,$h,$i,$s]\n";
  }//if
}//while
// le client ferme la connexion à son tour
fclose($connexion);
// fin
exit;
```

Résultats

```
HTTP/1.1 200 OK
Date: Wed, 17 Aug 2011 14:14:51 GMT
Server: Apache/2.2.17 (Win32) PHP/5.3.5
X-Powered-By: PHP/5.3.5
Content-Length: 17
Connection: close
Content-Type: text/html

17/08/11 14:14:51
dateheure=[17,08,11,14,14,51]
```

10.2 Récupération par le serveur des paramètres envoyés par le client

Dans le protocole Http, un client a deux méthodes pour passer des paramètres au serveur Web :

- il demande l'URL du service sous la forme
  GET *url?param1=val1&param2=val2&param3=val3… HTTP/1.0*

où les valeurs *vali* doivent au préalable subir un encodage afin que certains caractères réservés soient remplacés par leur valeur hexadécimale.

- il demande l'URL du service sous la forme
  POST *url HTTP/1.0*

puis parmi les entêtes Http envoyés au serveur place l'entête suivant :

Content-length=N

La suite des entêtes envoyés par le client se terminent par une ligne vide. Il peut alors envoyer ses données sous la forme

val1&param2=val2&param3=val3…

où les valeurs *vali* doivent, comme pour la méthode GET, être préalablement encodées. Le nombre de caractères envoyés au serveur doit être N où N est la valeur déclarée dans l'entête

Content-length=N

Le script PHP qui récupère les paramètres *parami* précédents envoyés par le client obtient leurs valeurs dans le tableau :

- $_GET["*param*"] pour une commande GET
- $_POST["*param*"] pour une commande POST

10.2.1 Le client GET (client1_web_02)

Le script Php ci-dessous envoie trois paramètres [nom, prenom, age] au serveur.

```
1.  <?php
2.
3.  // client : envoie prenom,nom,age au serveur par la méthode GET
4.  // données
5.  $HOTE = "localhost";
6.  $PORT = 80;
7.  $URL = "/exemples-web/web_02.php";
8.  list($prenom, $nom, $age) = array("jean-paul", "de la hûche", 45);
9.  // connexion au serveur web
10. $connexion = fsockopen($HOTE, $PORT);
11. // retour si erreur
12. if (!$connexion) {
13.   print "Echec de la connexion au site ($HOTE,$PORT) : $erreur";
14.   exit;
15. }//if
16. // envoi des informations au serveur php
17. // on encode les informations
18. $infos = "prenom=" . urlencode(utf8_decode($prenom)) . "&nom=" . urlencode(utf8_decode($nom)) . "&age=" .
    urlencode("$age");
19. // suivi console
20. print "infos envoyées au serveur (GET)=$infos\n";
21. print "URL demandée=[$URL?$infos]\n\n";
22. // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
23. // GET
24. fputs($connexion, "GET $URL?$infos HTTP/1.1\n");
25. // Host
26. fputs($connexion, "Host: localhost\n");
27. // Connection
28. fputs($connexion,"Connection: close\n");
29. // ligne vide
30. fputs($connexion,"\n");
31. // le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
32. // ses données puis fermer le canal. Le client lit tout ce qui arrive de $connexion jusqu'à la fermeture du canal
33. while ($ligne = fgets($connexion, 1000))
34.   print "$ligne";
35. // le client ferme la connexion à son tour
36. fclose($connexion);
```

Commentaires

- ligne 7 : Url du script serveur
- ligne 8 : les valeurs des 3 paramètres
- ligne 10 : ouverture d'une connexion avec le serveur web
- ligne 18 : encodage des 3 paramètres. Nous sommes dans un script écrit sous Netbeans avec un encodage des caractères en UTF-8. Donc les 3 valeurs des paramètres de la ligne 8 sont encodés en UTF-8. La fonction utf8_decode passe leur encodage en ISO-8859-1. Ceci fait, ils peuvent être encodés pour l'Url. Tous les caractères non alphabétiques sont remplacés par %xx où xx est la valeur hexadécimale du caractère. Les espaces sont eux remplacés par le signe +.
- ligne 24 : l'Url demandée est **$URL?$infos** où $*infos* est de la forme *nom=val1&prenom=val2&age=val3*.

10.2.2 Le serveur (web_02)

Le serveur se contente d'afficher ce qu'il reçoit.

```
1.  <?php
2.
```

```
3.  // gestion des erreurs
4.  ini_set("display_errors", "off");
5.
6.  // récupération par le serveur des informations envoyées par le client
7.  // ici prenom=P&nom=N&age=A
8.  // ces informations sont automatiquement disponibles dans les variables
9.  // $_GET['prenom'], $_GET['nom'], $_GET['age']
10. // on les renvoie au client
11.
12. // entête UTF-8
13. header("Content-Type: text/plain; charset=utf-8");
14.
15. // paramètres envoyés au serveur
16. $prenom = isset($_GET['prenom']) ? $_GET['prenom'] : "";
17. $nom = isset($_GET['nom']) ? $_GET['nom'] : "";
18. $age = isset($_GET['age']) ? $_GET['age'] : "";
19.
20. // réponse au client
21. $réponse = "informations reçues du client [" .
22.         utf8_encode(htmlspecialchars($prenom, ENT_QUOTES)) .
23.         "," . utf8_encode(htmlspecialchars($nom, ENT_QUOTES)) .
24.         "," . utf8_encode(htmlspecialchars($age, ENT_QUOTES)) . "]\n";
25. print $réponse;
```

Commentaires

- ligne 13 : fixe l'entête Http "Content-Type". Par défaut, le serveur web envoie l'entête

```
Content-Type: text/html
```

qui dit que la réponse est du texte au format Html. Ici, la réponse sera du texte sans formatage avec des caractères encodés en UTF-8 :

```
Content-Type: text/plain; charset=utf-8
```

Les entêtes Http doivent être envoyés avant la réponse du serveur. Aussi ci-dessus, l'appel de la fonction *header* se trouvera-t-elle avant toute instruction *print*.

- lignes 16-18 : on récupère les 3 paramètres dans le tableau $_GET.
- ligne 21 : on construit la chaîne de caractères qui va être envoyée en réponse au client. Certains caractères ont des significations spéciales en HTML, et doivent être remplacés par des entités HTML pour être affichés. **htmlspecialchars($string)** remplace tous ces caractères par leur équivalent dans la chaîne *$string*. Par exemple, le caractère $ devient &amp. Ensuite comme nous avons dit en ligne 13 que la réponse serait du texte UTF-8, nous encodons en UTF-8 les valeurs récupérées.
- ligne 25 : la réponse est envoyée au client

Test 1

Exécutons le script [web_02] à partir de Netbeans. Un navigateur est alors lancé pour afficher l'Url [http://localhost/exemples-web/web_02.php] :

- en [1], la navigateur affiche l'Url [http://localhost/exemples-web/web_02.php]. Parce que nous n'avons pas suffixé cette Url avec des paramètres, le serveur a répondu avec des paramètres vides. Rappelons que la réponse du serveur est celle écrite avec l'instruction *print*.
- en [2], nous suffixons l'Url avec des paramètres. Cette fois-ci le script serveur les renvoie bien.

Notons que Netbeans n'est pas nécessaire à l'exécution d'un script serveur. Il suffit de taper l'Url du script serveur dans un navigateur pour que ce script soit exécuté.

Test 2

On exécute dans Netbeans le client [client1_web_02.php]. On reçoit la réponse suivante :

```
1.  infos envoyées au serveur (GET)=prenom=jean-paul&nom=de+la+h%FBche&age=45
2.  URL demandée=[/exemples-web/web_02.php?prenom=jean-paul&nom=de+la+h%FBche&age=45]
3.
4.  HTTP/1.1 200 OK
5.  Date: Wed, 17 Aug 2011 14:31:01 GMT
6.  Server: Apache/2.2.17 (Win32) PHP/5.3.5
7.  X-Powered-By: PHP/5.3.5
8.  Content-Length: 59
9.  Connection: close
10. Content-Type: text/plain; charset=utf-8
11.
12. informations reçues du client [jean-paul,de la hûche,45]
```

- ligne 1 : encodage des 3 paramètres. On voit que le caractère û est devenu %FB.
- ligne 12 : la réponse du serveur

10.2.3 Le client POST (client2_web_03)

Un client Http envoie au serveur web la séquence de texte suivante : *entêtes Http, ligne vide, document*. Dans le client précédent, cette séquence était la suivante :

```
GET /url?paramètres HTTP/1.1
… autres entêtes Http
ligne vide
```

Il n'y avait pas de document. Il existe une autre façon de transmettre des paramètres, la méthode dite POST. Dans ce cas, la séquence de texte envoyée au serveur web est la suivante :

```
POST /url HTTP/1.1
… autres entêtes Http
ligne vide
paramètres
```

Cette fois-ci, les paramètres qui pour le client GET étaient inclus dans les entêtes Http, font partie, dans le client POST, du document envoyé derrière les entêtes.

Le script du client POST est le suivant :

```
1.  <?php
2.
3.  // client : envoie prenom,nom,age au serveur par la méthode POST
4.  // données
5.  $HOTE = "localhost";
6.  $PORT = 80;
7.  $URL = "/exemples-web/web_03.php";
8.  list($prenom, $nom, $age) = array("jean-paul", "de la hûche", 45);
9.  // connexion au serveur web
10. $connexion = fsockopen($HOTE, $PORT);
11. // retour si erreur
12. if (!$connexion) {
13.   print "Echec de la connexion au site ($HOTE,$PORT) : $erreur";
14.   exit;
15. }//if
16. // envoi des informations au serveur php
17. // on encode les informations
18. $infos = "prenom=" . urlencode(utf8_decode($prenom)) . "&nom=" . urlencode(utf8_decode($nom)) . "&age=" .
    urlencode("$age");
19. print "client : infos envoyées au serveur (POST) : $infos\n";
20. // on se connecte à l'URL $URL en lui postant (POST) des paramètres
21. // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
22. // POST
23. fputs($connexion, "POST $URL HTTP/1.1\n");
24. // Host
```

```
25. fputs($connexion, "Host: localhost\n");
26. // Connection
27. fputs($connexion,"Connection: close\n");
28. // Content-type
29. fputs($connexion, "Content-type: application/x-www-form-urlencoded\n");
30. // Content-length
31. // on envoie la taille (nombre de caractères) des infos qui vont être envoyées
32. fputs($connexion, "Content-length: " . strlen($infos) . "\n");
33. // on envoie une ligne vide
34. fputs($connexion, "\n");
35. // on envoie les infos
36. fputs($connexion, $infos);
37. // le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
38. // ses données puis fermer le canal. Le client lit tout ce qui arrive de $connexion
39. // jusqu'à la fermeture du canal
40. while ($ligne = fgets($connexion, 1000))
41.   print "$ligne";
42. // le client ferme la connexion à son tour
43. fclose($connexion);
```

Commentaires

- ligne 7 : l'Url du service web auquel le client POST va se connecter. Ce service web va être prochainement décrit.
- ligne 8 : les paramètres à transmettre au service web
- ligne 10 : connexion au serveur web
- ligne 18 : encodage des paramètres à envoyer au service web
- ligne 23 : commande Http POST
- ligne 25 : commande Http Host
- ligne 27 : commande Http Connection
- ligne 29 : commande Http **Content-type**. Nous avons déjà rencontré cet entête Http. Il est présent à chaque fois qu'un document est envoyé. Un serveur web qui envoie un document Html, utilise l'entête Http
  ```
  Content-type : text/html
  ```
  S'il envoie du texte non formaté, il utilise l'entête Http
  ```
  Content-type : text/plain
  ```
  Notre client POST envoie un document qui est un texte de la forme *param1=val1&param2=val2&...*. Ce type de document a le type **application/x-www-form-urlencoded**. Nous n'expliquerons pas pourquoi car cela nous obligerait à expliquer ce qu'est un formulaire web.
- ligne 32 : commande **Content-length**. Nous avons déjà rencontré cet entête Http. Il est présent à chaque fois qu'un document est envoyé. Il indique le nombre d'octets du document.
- ligne 34 : la ligne vide signalant la fin des entêtes Http
- ligne 36 : l'envoi des paramètres
- lignes 40-41 : lecture de la réponse complète du serveur
- ligne 43 : fermeture de la connexion

10.2.4 Le serveur (web_03)

Le service web [web_03] fait la même chose que le service web [web_02]. Il lit les paramètres envoyés par le client POST et les erenvoie au client. Son code est le suivant :

```
1. <?php
2.
3. // gestion des erreurs
4. ini_set("display_errors", "off");
5. // entête UTF-8
6. header("Content-Type: text/plain; charset=utf-8");
7.
8. // récupération par le serveur des informations envoyées par le client
9. // ici prenom=P&nom=N&age=A
10. // ces informations sont automatiquement disponibles dans les variables
11. // $_POST['prenom'], $_POST['nom'], $_POST['age']
12. // on les renvoie au client
13. // paramètres envoyés au serveur
14. $prenom = isset($_POST['prenom']) ? $_POST['prenom'] : "";
15. $nom = isset($_POST['nom']) ? $_POST['nom'] : "";
```

```
16. $age = isset($_POST['age']) ? $_POST['age'] : "";
17. // réponse au client
18. $réponse = "informations reçues du client [" .
19.         utf8_encode(htmlspecialchars($prenom, ENT_QUOTES)) .
20.         "," . utf8_encode(htmlspecialchars($nom, ENT_QUOTES)) .
21.         "," . utf8_encode(htmlspecialchars($age, ENT_QUOTES)) . "]\n";
22. print $réponse;
```

Commentaires

- lignes 14-16 : les paramètres envoyés par un client POST deviennent disponibles dans le tableau $_POST pour le service web qui les reçoit.
- ligne 6 : entête Http *Content-Type*. On peut s'étonner de ne pas trouver dans les entêtes Http l'entête Http *Content-Length* indiquant la taille du document renvoyé au client. Nous avons vu que le serveur web envoyait des entêtes Http par défaut. L'entête *Content-Length* en fait partie.

Résultats

Une fois le script serveur écrit sous Netbreans, il devient immédiatement disponible via le serveur Apache de WampServer. Rappelons-nous que cela est obtenu par configuration (cf page 78). On lance le client qui interroge le serveur et on reçoit alors la réponse suivante :

```
1.  client : infos envoyées au serveur (POST) : prenom=jean-paul&nom=de+la+h%FBche&age=45
2.  HTTP/1.1 200 OK
3.  Date: Sat, 20 Aug 2011 12:55:19 GMT
4.  Server: Apache/2.2.17 (Win32) PHP/5.3.5
5.  X-Powered-By: PHP/5.3.5
6.  Content-Length: 59
7.  Connection: close
8.  Content-Type: text/plain; charset=utf-8
9.
10. informations reçues du client [jean-paul,de la hûche,45]
```

- lignes 2-10 : la réponse du serveur
- lignes 2-8 : les en-têtes Http
- ligne 10 : le document
- ligne 6 : l'en-tête Http *Content-Length*. Comme ce n'est pas le script serveur qui a généré cet en-tête, il a donc été généré par le serveur web.
- ligne 8 : le seul en-tête généré par le script serveur

10.3 Récupération des variables d'environnement du serveur WEB

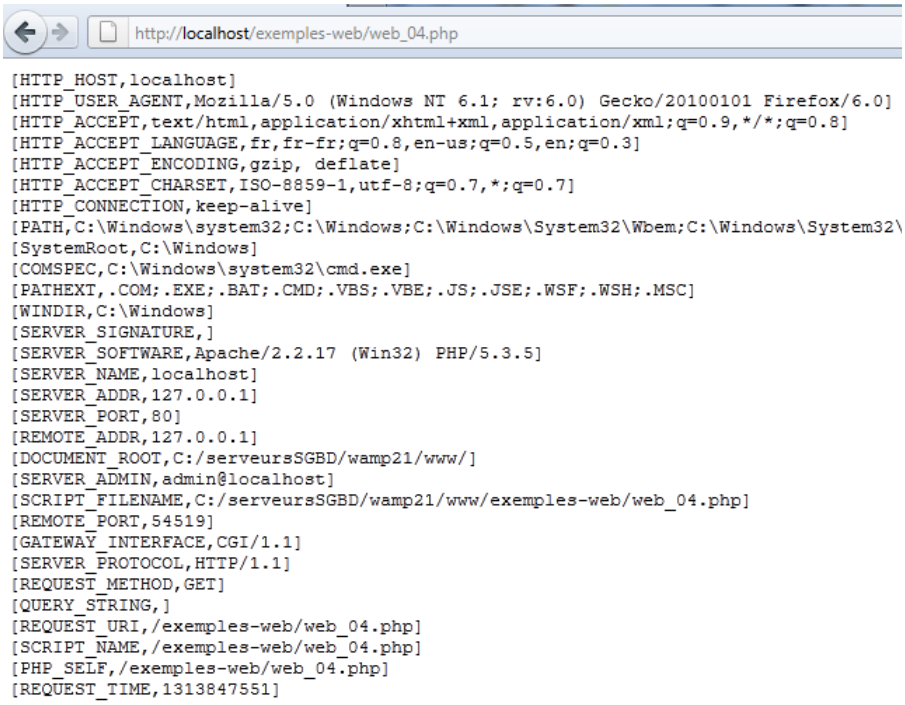
Un script Serveur s'exécute dans un environnement web qu'il peut connaître. Cet environnement est stocké dans le dictionnaire $_SERVER. Nous écrivons tout d'abord une application serveur qui envoie à ses clients le contenu de ce dictionnaire.

10.3.1 Le serveur (web_04)

```
1.  <?php
2.
3.  // gestion des erreurs
4.  ini_set("display_errors", "off");
5.  // entête UTF-8
6.  header("Content-Type: text/plain; charset=utf-8");
7.
8.  // on renvoie au client la liste des variables disponibles dans l'environnement du serveur
9.  foreach ($_SERVER as $clé => $valeur) {
10.   print "[$clé,$valeur]\n";
11. }
```

- les couples (clé, valeur) du dictionnaire $_SERVER sont envoyés aux clients.

Le résultat obtenu lorsque le client est un navigateur web est le suivant :



Voici la signification de certaines des variables (pour windows. Sous Linux, elles seraient différentes) :

HTTP_CMDE	CMDE représente l'entête Http envoyé par le client. On a accès à tous ces entêtes.
PATH	le chemin des exécutables sur la machine sur laquelle s'exécute le script serveur
COMSPEC	le chemin de l'interpréteur de commandes Dos
PATHEXT	les extensions des fichiers exécutables
WINDIR	le dossier d'installation de Windows
SERVER_SIGNATURE	la signature du serveur web. Ici rien.
SERVER_SOFTWARE	le type du serveur web
SERVER_NAME	le nom Internet de la machine du serveur web
SERVER_PORT	le port d'écoute du serveur web
SERVER_ADDR	l'adresse IP de la machine du serveur web
REMOTE_ADDR	l'adresse IP du client. Ici le client était sur la même machine que le serveur.
REMOTE_PORT	le port de communication du client
DOCUMENT_ROOT	la racine de l'arborescence des documents servis par le serveur web
SERVER_ADMIN	l'adresse électronique de l'administrateur du serveur web
SCRIPT_FILENAME	le chemin complet du scrit serveur
SERVER_PROTOCOL	la version du protocole Http utilisée par le serveur web
REQUEST_METHOD	l'ordre Http utilisé par le client. Il y en a quatre : GET, POST, PUT, DELETE
QUERY_STRING	les paramètres envoyés avec un ordre GET /url?paramètres
REQUEST_URI	l'Url demandée par le client. Si le navigateur demande l'Url http://machine[:port]/uri, on aura REQUEST_URI=uri
SCRIPT_NAME	$_SERVER['SCRIPT_FILENAME']=$_SERVER['DOCUMENT_ROOT']. $_SERVER['SCRIPT_NAME']

10.3.2 Le client (client1_web_04)

Le client se contente d'afficher tout ce que lui envoie le serveur.

```php
<?php

// données
$HOTE = "localhost";
$PORT = 80;
$urlServeur = "/exemples-web/web_04.php";
// ouverture d'une connexion sur le port 80 de $HOTE
$connexion = fsockopen($HOTE, $PORT);
// erreur ?
if (!$connexion) {
  print "Erreur : $erreur\n";
  exit;
}

// on se connecte sur une URL au serveur Web
// les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
// GET
fputs($connexion, "GET $urlServeur HTTP/1.1\n");
// Host
fputs($connexion, "Host: localhost\n");
// Connection
fputs($connexion,"Connection: close\n");
// ligne vide
fputs($connexion,"\n");
// le serveur va maintenant répondre sur le canal $connexion. Il va envoyer toutes
// ses données puis fermer le canal. Le client lit donc tout ce qui arrive de $connexion
// jusqu'à la fermeture du canal
while ($ligne = fgets($connexion, 1000)) {
  print "$ligne";
}//while
// le client ferme la connexion à son tour
fclose($connexion);
// fin
exit;
```

Résultats

```
HTTP/1.1 200 OK
Date: Sat, 20 Aug 2011 14:11:58 GMT
Server: Apache/2.2.17 (Win32) PHP/5.3.5
X-Powered-By: PHP/5.3.5
Content-Length: 1353
Connection: close
Content-Type: text/plain; charset=utf-8

[HTTP_HOST,localhost]
[HTTP_CONNECTION,close]
[PATH,C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Program Files\Microsoft SQL Server\90\Tools\binn\;...;]
[SystemRoot,C:\Windows]
[COMSPEC,C:\Windows\system32\cmd.exe]
[PATHEXT,.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC]
[WINDIR,C:\Windows]
[SERVER_SIGNATURE,]
[SERVER_SOFTWARE,Apache/2.2.17 (Win32) PHP/5.3.5]
[SERVER_NAME,localhost]
[SERVER_ADDR,127.0.0.1]
[SERVER_PORT,80]
[REMOTE_ADDR,127.0.0.1]
[DOCUMENT_ROOT,C:/serveursSGBD/wamp21/www/]
[SERVER_ADMIN,admin@localhost]
[SCRIPT_FILENAME,C:/serveursSGBD/wamp21/www/exemples-web/web_04.php]
[REMOTE_PORT,54552]
[GATEWAY_INTERFACE,CGI/1.1]
[SERVER_PROTOCOL,HTTP/1.1]
[REQUEST_METHOD,GET]
[QUERY_STRING,]
[REQUEST_URI,/exemples-web/web_04.php]
[SCRIPT_NAME,/exemples-web/web_04.php]
[PHP_SELF,/exemples-web/web_04.php]
[REQUEST_TIME,1313849518]
```

10.4 Gestion des sessions WEB

Dans les exemples client / serveur précédents on avait le fonctionnement suivant :

- le client ouvre une connexion vers le port 80 de la machine du service web
- il envoie la séquence de texte : en-têtes Http, ligne vide, [document]
- en réponse, le serveur envoie une séquence du même type
- le serveur clôt la connexion vers le client
- le client clôt la connexion vers le serveur

Si le même client fait peu après une nouvelle demande au serveur web, une nouvelle connexion est créée entre le client et le serveur. Celui-ci ne peut pas savoir si le client qui se connecte est déjà venu ou si c'est une première demande. Entre deux connexions, le serveur "oublie" son client. Pour cette raison, on dit que le protocole Http est un protocole sans état. Il est pourtant utile que le serveur se souvienne de ses clients. Ainsi si une application est sécurisée, le client va envoyer au serveur un login et un mot de passe pour s'identifier. Si le serveur "oublie" son client entre deux connexions, celui-ci devra s'identifier à chaque nouvelle connexion, ce qui n'est pas envisageable.

Pour faire le suivi d'un client, le serveur procéde de la façon suivante : lors d'une première demande d'un client, il inclut dans sa réponse un identifiant que le client doit ensuite lui renvoyer à chaque nouvelle demande. Grâce à cet identifiant, différent pour chaque client, le serveur peut reconnaître un client. Il peut alors gérer une mémoire pour ce client sous la forme d'un fichier associé de façon unique à l'identifiant du client.

Techniquement cela se passe ainsi :

- dans la réponse à un nouveau client, le serveur inclut l'en-tête Http *Set-Cookie : MotClé=Identifiant*. Il ne fait cela qu'à la première demande.
- dans ses demandes suivantes, le client va renvoyer son identifiant via l'en-tête Http *Cookie : MotClé=Identifiant* afin que le serveur le reconnaisse.

On peut se demander comment le serveur fait pour savoir qu'il a affaire à un nouveau client plutôt qu'à un client déjà venu. C'est la présence de l'en-tête Http *Cookie* dans les en-têtes Http du client qui le lui indique. Pour un nouveau client, cet en-tête est absent.

L'ensemble des connexions d'un client donné est appelé une session.

10.4.1 Le fichier de configuration

Pour que la gestion des sessions fonctionne correctement avec PHP, il faut vérifier que celui-ci est correctement configuré. Sous windows, son fichier de configuration est *php.ini*. Selon le contexte d'exécution (console, web), le fichier de configuration [php.ini] doit être recherché dans des dossiers différents. Pour découvrir ceux-ci, on utilisera le script suivant :

```
1. <?php
2. 
3. // infos Php
4. phpinfo();
```

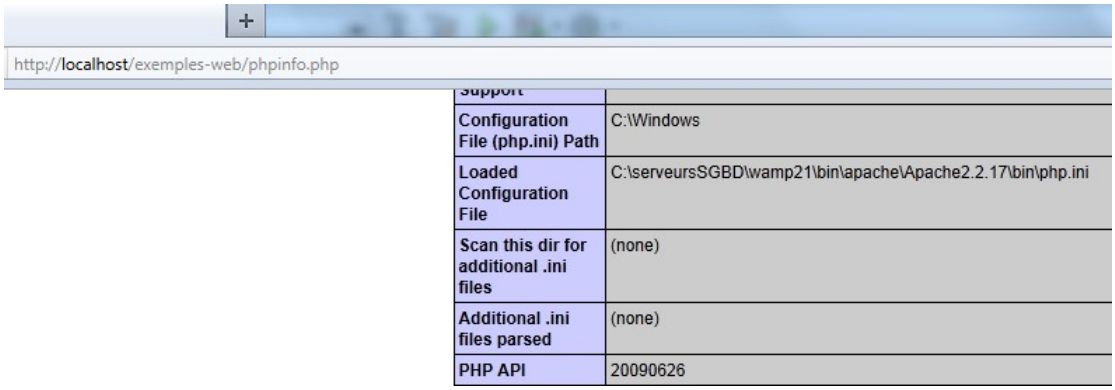
Ligne 4, la fonction *phpinfo* donne des informations sur l'interpréteur Php qui exécute le script. Elle donne notamment le chemin du fichier de configuration [php.ini] utilisé.

Dans un environnement console, on obtient un résultat analogue au suivant :

```
1. …
2. Configuration File (php.ini) Path => C:\Windows
3. Loaded Configuration File => C:\serveursSGBD\wamp21\bin\php\php5.3.5\php.ini
4. Scan this dir for additional .ini files => (none)
5. Additional .ini files parsed => (none)
6. …
```

ligne 2 : le fichier de configuration principal est *c:\windows\php.ini*
ligne 3 : un fichier de configuration secondaire est *C:\serveursSGBD\wamp21\bin\php\php5.3.5\php.ini*. Il permet de modifier certaines options de configuration du fichier de configuration principal.

Dans un environnement web, on obtient le résultat suivant :



Le fichier secondaire de configuration n'est ici pas le même que dans l'environnement console. C'est ce dernier que nous allons consulter. On trouve dans ce fichier une section **session** :

```
1.  [Session]
2.  session.save_handler = files
3.  session.save_path = "C:/serveursSGBD/wamp21/tmp"
4.  session.use_cookies = 1
5.  session.use_only_cookies = 1
6.  session.name = PHPSESSID
7.  session.auto_start = 0
8.  session.cookie_lifetime = 0
9.  session.cookie_path = /
10. session.cookie_domain =
11. session.serialize_handler = php
12. session.cache_expire = 180
13. session.hash_function = 0
```

- ligne 1 : les données d'une session client sont sauvegardées dans un fichier
- ligne 3 : le dossier de sauvegarde des données de session. Si ce dossier n'existe pas, aucune erreur n'est signalée et la gestion des sessions ne fonctionne pas.
- lignes 4-5 : indiquent que l'identifiant de session est géré par les en-têtes Http *Set-Cookie* et *Cookie*
- ligne 6 : l'entête *Set-Cookie* sera de la forme *Set-Cookie : PHPSESSID=identifiant_de_session*
- ligne 7 : une session client n'est pas démarrée automatiquement. Le script serveur doit la demander explicitement par une instruction *session_start()*.

10.4.2 Le serveur 1 (web_05)

La gestion de l'identifiant de session est transparent pour un service web. Cet identifiant est géré par le serveur web. Un service web a accès à la session du client via l'instruction *session_start()*. A partir de ce moment, le service web peut lire / écrire des données dans la session du client via le dictionnaire $_SESSION. Le code suivant montre la gestion en session de trois compteurs.

```
1.  <?php
2.
3.  // gestion des erreurs
4.  ini_set("display_errors", "off");
5.  // entête UTF-8
6.  header("Content-Type: text/plain; charset=utf-8");
7.
8.  // on ouvre une session
9.  session_start();
10. // on met 3 variables en session
11. if (!isset($_SESSION['N1'])) {
12.   $_SESSION['N1'] = 0;
13. }
14. if (!isset($_SESSION['N2'])) {
15.   $_SESSION['N2'] = 10;
16. }
17. if (!isset($_SESSION['N3'])) {
18.   $_SESSION['N3'] = 100;
19. }
20. // incrémentation des 3 variables
21. $_SESSION['N1']++;
```

```
22. $_SESSION['N2']++;
23. $_SESSION['N3']++;
24. // envoi d'informations au client
25. print "N1=".$_SESSION['N1']."\n";
26. print "N2=".$_SESSION['N2']."\n";
27. print "N3=".$_SESSION['N3']."\n";
28. // fin de session
29. session_close();
```

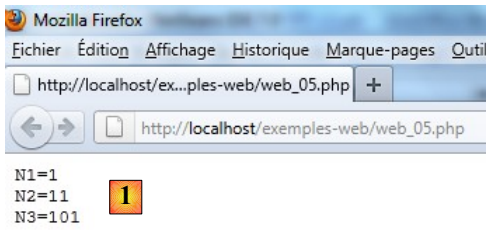
- ligne 9 : démarrage d'une session client
- lignes 11-13 : le tableau $_SESSION est un dictionnaire de couple (clé, valeur). Les données mises dans ce dictionnaire persistent au fil des requêtes d'un même client. C'est la mémoire du client sur le serveur.
- lignes 11-19 : si les trois compteurs N1, N2, N3 ne sont pas en session, on les y met.
- lignes 21-23 : on les incrémente d'une unité
- lignes 25-27 : on envoie leur valeur au client

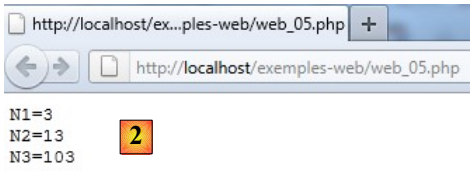
Dans la relation client / serveur, la gestion de la session client sur le serveur dépend des deux acteurs, le client et le serveur :

- le serveur a la charge d'envoyer un identifiant à son client lors de sa première demande
- le client a la charge de renvoyer cet identifiant à chaque nouvelle demande. S'il ne le fait pas, le serveur croira que c'est un nouveau client et génèrera un nouvel identifiant pour une nouvelle session.

Résultats

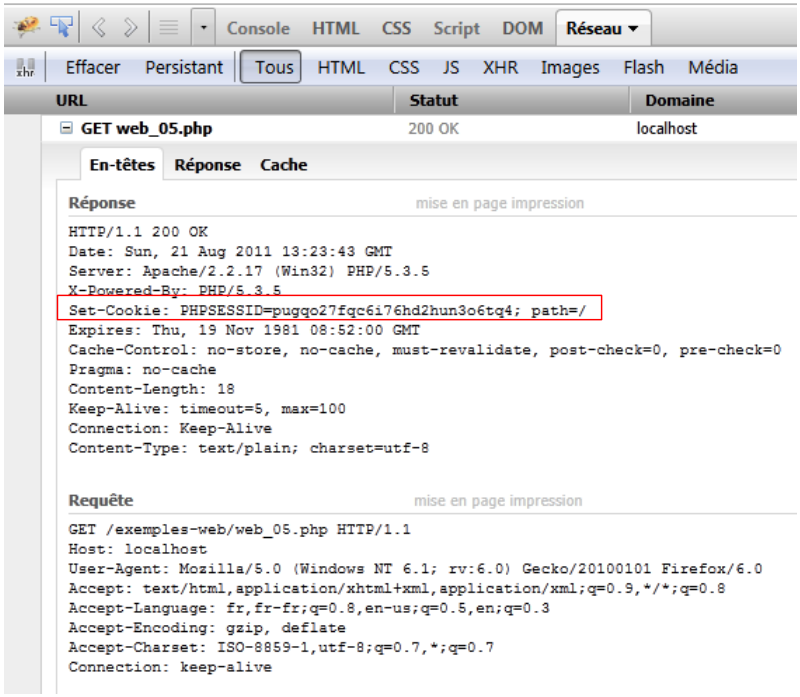
Nous utilisons comme client, un navigateur web. Par défaut (par configuration en fait), celui-ci renvoie bien au serveur les identifiants de session que celui-ci lui envoie. Au fil des requêtes, le navigateur va recevoir les trois compteurs envoyés par le serveur et va voir leurs valeurs s'incrémenter.





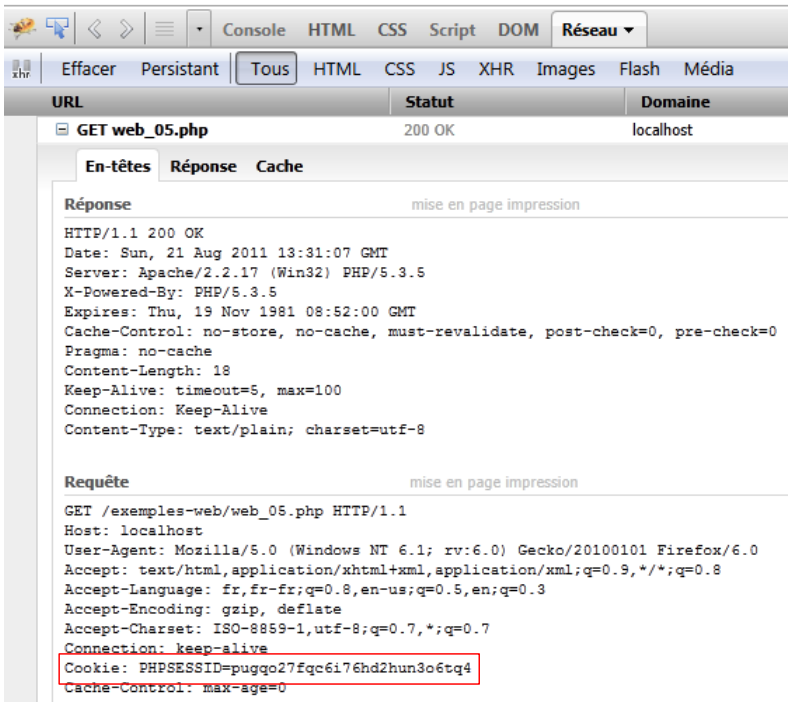
- en [1], la 1ère demande au service web [web_05]
- en [2], la 3ième demande montre que les compteurs sont bien incrémentés. Il y a bien mémorisation des valeurs des compteurs au fil des demandes.

Utilisons Firebug pour voir les en-têtes Http échangés entre le serveur et le client. Nous fermons Firefox pour terminer la session courante avec le serveur, le rouvrons et activons Firebug. Nous demandons le service |web_05] :



Ci-dessus, on voit l'identifiant de session envoyé par le serveur dans sa réponse à la 1ère demande du client. Il utilise l'en-tête Http *Set-Cookie*.

Faisons une nouvelle demande en rafraîchissant (F5) la page dans le navigateur web :



Ci-dessus on remarquera deux choses :

- le navigateur web renvoie l'identifiant de session avec l'en-tête Http *Cookie*.
- dans sa réponse, le service web n'inclut plus cet identifiant. C'est désormais le client qui a la charge de l'envoyer à chacune de ses demandes.

10.4.3 Le client 1 (client1_web_05)

Nous écrivons maintenant un script client du script serveur précédent. Dans sa gestion de la session, il doit se comporter comme le navigateur web :

- dans la réponse du serveur à sa première demande, il doit trouver l'identifiant de session que le serveur lui envoie. Il sait qu'il le trouvera dans l'en-tête Http *Set-Cookie*.
- il doit, à chacune de ses demandes ultérieures, renvoyer au serveur l'identifiant qu'il a reçu. Il le fera avec l'en-tête Http *Cookie*.

Le code du client est le suivant :

```
1.  <?php
2.
3.  // données
4.  $HOTE = "localhost";
5.  $PORT = 80;
6.  $urlServeur = "/exemples-web/web_05.php";
7.  // tests
8.  $cookie = "";
9.  for ($i = 0; $i < 5; $i++) {
10.   list($erreur, $cookie, $N1, $N2, $N3) = connecte($HOTE, $PORT, $urlServeur, $cookie);
11.   print "----------------------------\n";
12.   print "client(erreur,cookie,N1,N2,N3)=[$erreur,$cookie,$N1,$N2,$N3]\n";
13.   print "----------------------------\n";
14. }
15. // fin
16. exit;
17.
18. function connecte($HOTE, $PORT, $urlServeur, $cookie) {
19.   // connecte le client à ($HOTE,$PORT,$urlServeur)
20.   // envoie le cookie $cookie si celui-ci est non vide
21.   // affiche ttes les lignes reçues en réponse
22.   // ouverture d'une connexion sur le port 80 de $HOTE
23.   $connexion = fsockopen($HOTE, $PORT);
24.   // erreur ?
25.   if (!$connexion)
26.     return array("erreur lors de la connexion au serveur ($HOTE, $PORT)");
27.   // on se connecte sur $urlserveur
28.   // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
29.   // GET
30.   fputs($connexion, "GET $urlServeur HTTP/1.1\n");
31.   // Host
32.   fputs($connexion, "Host: localhost\n");
33.   // Connection
34.   fputs($connexion, "Connection: close\n");
35.   // on envoie le cookie s'il est non vide
36.   if ($cookie) {
37.     fputs($connexion, "Cookie: $cookie\n");
38.   }//if
39.   // envoi ligne vide
40.   fputs($connexion, "\n");
41.   // on affiche la réponse du serveur web
42.   // et on prend soin de récupérer l'éventuel cookie et les valeurs des Ni
43.   $N = "";
44.   while ($ligne = fgets($connexion, 1000)) {
45.     print "$ligne";
46.     // cookie - seulement lors de la 1ère réponse
47.     if (!$cookie) {
48.       if (preg_match("/^Set-Cookie: (.*?)\s*$/", $ligne, $champs)) {
49.         $cookie = $champs[1];
50.       }
51.     }
52.     // valeur de N1
53.     if (preg_match("/^N1=(.*?)\s*$/", $ligne, $champs))
54.       $N1 = $champs[1];
55.     // valeur de N2
56.     if (preg_match("/^N2=(.*?)\s*$/", $ligne, $champs))
```

```
57.       $N2 = $champs[1];
58.     // valeur de N3
59.     if (preg_match("/^N3=(.*?)\s*$/", $ligne, $champs))
60.       $N3 = $champs[1];
61.   }//while
62.   // on ferme la connexion
63.   fclose($connexion);
64.   // retour
65.   return array("", $cookie, $N1, $N2, $N3);
66. }
```

Commentaires

- lignes 3-16 : le programme principal
- lignes 18-67 : la fonction *connecte*
- lignes 9-14 : le client appelle 5 fois le serveur et affiche les valeurs successives des compteurs N1, N2 et N3. Si la session est correctement gérée, ces compteurs devraient être incrémentés de 1 à chaque nouvelle demande.
- ligne 10 : la fonction *connecte* utilise les paramètres $*HOTE*, $*PORT*, $*urlServeur* pour connecter le client au service web. Le paramètre $*cookie* représente l'identifiant de session. Au 1er appel, c'est la chaîne vide. Aux suivants, c'est l'identifiant de session envoyé par le serveur en réponse au 1er appel du client. La fonction *connecte* renvoie en résultats, les valeurs des trois compteurs $N1, $N2, $N3, l'identifiant de session $*cookie* et une éventuelle erreur $*erreur*.
- ligne 18 : la fonction *connecte* a les caractéristiques d'un client Http classique. Nous ne commentons que les nouveautés.
- lignes 30-40 : envoi des en-têtes Http.
- lignes 36-38 : si l'identifiant de session est connu, on l'envoie au serveur
- lignes 44-66 : exploitation de toutes les lignes de texte envoyées par le serveur
- lignes 47-51 : si l'identifiant de session n'a pas encore été récupéré, on le récupère dans l'en-tête Http *Set-Cookie* par une expression régulière.
- lignes 53-54 : le compteur N1 est obtenu lui aussi par une expression régulière
- lignes 56-57, 59-60 : idem pour les compteurs N2 et N3
- ligne 63 : fermeture de la connexion avec le serveur.
- ligne 65 : renvoi des résultats sous la forme d'un tableau.

Résultats

L'exécution du script client provoque l'affichage suivant dans la console Netbeans :

```
1.  HTTP/1.1 200 OK
2.  Date: Sun, 21 Aug 2011 13:59:17 GMT
3.  Server: Apache/2.2.17 (Win32) PHP/5.3.5
4.  X-Powered-By: PHP/5.3.5
5.  Set-Cookie: PHPSESSID=ohiqtkv7hu2b26kdshjtqms9p7; path=/
6.  Expires: Thu, 19 Nov 1981 08:52:00 GMT
7.  Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
8.  Pragma: no-cache
9.  Content-Length: 18
10. Connection: close
11. Content-Type: text/plain; charset=utf-8
12.
13. N1=1
14. N2=11
15. N3=101
16. ----------------------------
17. client(erreur,cookie,N1,N2,N3)=[,PHPSESSID=ohiqtkv7hu2b26kdshjtqms9p7; path=/,1,11,101]
18. ----------------------------
19. HTTP/1.1 200 OK
20. Date: Sun, 21 Aug 2011 13:59:18 GMT
21. Server: Apache/2.2.17 (Win32) PHP/5.3.5
22. X-Powered-By: PHP/5.3.5
23. Expires: Thu, 19 Nov 1981 08:52:00 GMT
24. Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
25. Pragma: no-cache
26. Content-Length: 18
27. Connection: close
28. Content-Type: text/plain; charset=utf-8
29.
30. N1=2
31. N2=12
32. N3=102
33. ----------------------------
34. client(erreur,cookie,N1,N2,N3)=[,PHPSESSID=ohiqtkv7hu2b26kdshjtqms9p7; path=/,2,12,102]
35. ----------------------------
36. HTTP/1.1 200 OK
37. Date: Sun, 21 Aug 2011 13:59:18 GMT
```

```
38. Server: Apache/2.2.17 (Win32) PHP/5.3.5
39. X-Powered-By: PHP/5.3.5
40. Expires: Thu, 19 Nov 1981 08:52:00 GMT
41. Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
42. Pragma: no-cache
43. Content-Length: 18
44. Connection: close
45. Content-Type: text/plain; charset=utf-8
46.
47. N1=3
48. N2=13
49. N3=103
50. ----------------------------
51. client(erreur,cookie,N1,N2,N3)=[,PHPSESSID=ohiqtkv7hu2b26kdshjtqms9p7; path=/,3,13,103]
52. ----------------------------
53. HTTP/1.1 200 OK
54. Date: Sun, 21 Aug 2011 13:59:18 GMT
55. Server: Apache/2.2.17 (Win32) PHP/5.3.5
56. X-Powered-By: PHP/5.3.5
57. Expires: Thu, 19 Nov 1981 08:52:00 GMT
58. Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
59. Pragma: no-cache
60. Content-Length: 18
61. Connection: close
62. Content-Type: text/plain; charset=utf-8
63.
64. N1=4
65. N2=14
66. N3=104
67. ----------------------------
68. client(erreur,cookie,N1,N2,N3)=[,PHPSESSID=ohiqtkv7hu2b26kdshjtqms9p7; path=/,4,14,104]
69. ----------------------------
70. HTTP/1.1 200 OK
71. Date: Sun, 21 Aug 2011 13:59:18 GMT
72. Server: Apache/2.2.17 (Win32) PHP/5.3.5
73. X-Powered-By: PHP/5.3.5
74. Expires: Thu, 19 Nov 1981 08:52:00 GMT
75. Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
76. Pragma: no-cache
77. Content-Length: 18
78. Connection: close
79. Content-Type: text/plain; charset=utf-8
80.
81. N1=5
82. N2=15
83. N3=105
84. ----------------------------
85. client(erreur,cookie,N1,N2,N3)=[,PHPSESSID=ohiqtkv7hu2b26kdshjtqms9p7; path=/,5,15,105]
```

- ligne 5 : dans sa première réponse, le serveur envoie l'identifiant de session. Dans les réponses suivantes, il ne l'envoie plus.
- on voit clairement que le serveur Web conserve les valeurs de (N1,N2,N3) au fil des demandes du client. C'est ce qu'on appelle le suivi de session.

Les deux exemples qui suivent montrent qu'on peut aussi sauvegarder les valeurs d'un tableau ou d'un objet.

10.4.4 Le serveur 2 (web_06)

Le script serveur suivant montre qu'on peut mettre en session un tableau ou un dictionnaire.

```
1.  <?php
2.
3.  // gestion des erreurs
4.  ini_set("display_errors", "off");
5.  // entête UTF-8
6.  header("Content-Type: text/plain; charset=utf-8");
7.
8.  // on ouvre une session
9.  session_start();
10. // on enregistre un tableau et un dictionnaire
11. // on initialise ou modifie le tableau
12. if (isset($_SESSION['tableau'])) {
13.   for ($i = 0; $i < count($_SESSION['tableau']); $i++) {
14.     $_SESSION['tableau'][$i]++;
```

```
15.   }
16. } else {
17.   for ($i = 0; $i < 10; $i++) {
18.     $_SESSION['tableau'][$i] = $i * 10;
19.   }
20. }
21. // on initialise ou modifie le dictionnaire
22. if (isset($_SESSION['dico'])) {
23.   foreach (array_keys($_SESSION['dico']) as $clé) {
24.     $_SESSION['dico'][$clé]++;
25.   }
26. } else {
27.   $_SESSION['dico'] = array("zéro" => 0, "dix" => 10, "vingt" => 20);
28. }
29. // envoi d'informations au client
30. print "tableau=" . join(",", $_SESSION['tableau']) . "\n";
31. print "dico=";
32. foreach ($_SESSION['dico'] as $clé => $valeur) {
33.   print "($clé,$valeur) ";
34. }
35. print "\n";
```

Commentaires

- lignes 17-19 : un tableau est initialement mis en session s'il n'y est pas déjà
- lignes 12-15 : s'il y est déjà, ses éléments sont incrémentés de 1
- ligne 27 : un dictionnaire à valeurs numériques est mis en session s'il n'y est pas déjà
- lignes 22-25 : s'il y est déjà, ses valeurs numériques sont incrémentées de 1
- lignes 30-35 : le tableau et le dictionnaire sont envoyés au client

10.4.5 Le client 2 (client1_web_06)

```
1.  <?php
2.
3.  // données
4.  $HOTE = "localhost";
5.  $PORT = 80;
6.  $urlServeur = "/exemples-web/web_06.php";
7.  // tests
8.  $cookie = "";
9.  for ($i = 0; $i < 5; $i++) {
10.   connecte($HOTE, $PORT, $urlServeur, $cookie);
11. }
12. // fin
13. exit;
14.
15. function connecte($HOTE, $PORT, $urlServeur, &$cookie) {
16.   // connecte le client à ($HOTE,$PORT,$urlServeur)
17.   // envoie le cookie $cookie si celui-ci est non vide
18.   // affiche ttes les lignes reçues en réponse
19.   // le cookie est passé par référence pour être partagé entre
20.   // le programme appelé et le programme appelant
21.   // ouverture d'une connexion sur le port $PORT de $HOTE
22.   $connexion = fsockopen($HOTE, $PORT);
23.   // erreur ?
24.   if (!$connexion)
25.     return array("erreur lors de la connexion au serveur ($HOTE, $PORT)");
26.   // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
27.   // GET
28.   fputs($connexion, "GET $urlServeur HTTP/1.1\n");
29.   // Host
30.   fputs($connexion, "Host: localhost\n");
31.   // Connection
32.   fputs($connexion, "Connection: close\n");
33.   // on envoie le cookie s'il est non vide
```

```
34.   if ($cookie) {
35.     fputs($connexion, "Cookie: $cookie\n");
36.   }
37.   // envoi ligne vide
38.   fputs($connexion, "\n");
39.   // on affiche la réponse du serveur web
40.   // et on prend soin de récupérer l'éventuel cookie
41.   while ($ligne = fgets($connexion, 1000)) {
42.     print "$ligne";
43.     // cookie - seulement lors de la 1ère réponse
44.     if (!$cookie) {
45.       if (preg_match("/^Set-Cookie: (.*?)\s*$/", $ligne, $champs)) {
46.         $cookie = $champs[1];
47.       }
48.     }
49.   }
50.   // on ferme la connexion
51.   fclose($connexion);
52.   // retour
53.   return "";
54. }
```

Le code du client est analogue au code du client déjà commenté.

Résultats

```
1.  HTTP/1.1 200 OK
2.  ...
3.  Set-Cookie: PHPSESSID=6lvttr0uhpj5q3sl91h4h7p322; path=/
4.  ...
5.
6.  tableau=0,10,20,30,40,50,60,70,80,90
7.  dico=(zéro,0) (dix,10) (vingt,20)
8.
9.  HTTP/1.1 200 OK
10. ...
11.
12. tableau=1,11,21,31,41,51,61,71,81,91
13. dico=(zéro,1) (dix,11) (vingt,21)
14.
15. HTTP/1.1 200 OK
16. ...
17.
18. tableau=2,12,22,32,42,52,62,72,82,92
19. dico=(zéro,2) (dix,12) (vingt,22)
20.
21. HTTP/1.1 200 OK
22. ...
23.
24. tableau=3,13,23,33,43,53,63,73,83,93
25. dico=(zéro,3) (dix,13) (vingt,23)
26.
27. HTTP/1.1 200 OK
28. ...
29.
30. tableau=4,14,24,34,44,54,64,74,84,94
31. dico=(zéro,4) (dix,14) (vingt,24)
```

10.4.6 Le serveur 3 (web_07)

Le script serveur suivant montre qu'on peut mettre en session un objet.

```
1.  <?php
2.
3.  // gestion des erreurs
4.  ini_set("display_errors", "off");
5.  // entête UTF-8
6.  header("Content-Type: text/plain; charset=utf-8");
7.
8.  // on ouvre une session
```

```php
session_start();
// on initialise ou modifie un objet Personne
if (isset($_SESSION['personne'])) {
  $personne = $_SESSION['personne'];
  // on incrémente l'âge
  $personne->setAge($personne->getAge() + 1);
} else {
  // on définit la personne
  $_SESSION['personne'] = new Personne("paul", "langévin", 10);
}
// affichage au client
print "personne=".$_SESSION['personne']."\n";
// fin
exit;

// ------------------------------------------------------------
class Personne {

// attributs de la classe
  private $prénom;
  private $nom;
  private $âge;

// getters and setters
  public function getPrénom() {
    return $this->prénom;
  }

  public function getNom() {
    return $this->nom;
  }

  public function getAge() {
    return $this->âge;
  }

  public function setPrénom($prénom) {
    $this->prénom = $prénom;
  }

  public function setNom($nom) {
    $this->nom = $nom;
  }

  public function setAge($age) {
    $this->âge = $age;
  }

// constructeur
  function __construct($prénom, $nom, $âge) {
    // on passe par les set
    $this->setPrénom($prénom);
    $this->setNom($nom);
    $this->setAge($âge);
  }

// méthode toString
  function __toString() {
    return "[$this->prénom,$this->nom,$this->âge]";
  }

}
```

Commentaires

- ligne 17 : on met en session un objet de type *Personne* s'il n'y est pas déjà.

- lignes 11-15 : s'il y est déjà, on incrémente son âge de 1
- ligne 20 : on envoie l'objet de type *Personne* au client.

10.4.7 Le client 3 (client1_web_07)

```
1.  <?php
2.
3.  // données
4.  $HOTE = "localhost";
5.  $PORT = 80;
6.  $urlServeur = "/exemples-web/web_07.php";
7.  // tests
8.  $cookie = "";
9.  for ($i = 0; $i < 5; $i++) {
10.   connecte($HOTE, $PORT, $urlServeur, $cookie);
11. }//if
12. // fin
13. exit;
14.
15. function connecte($HOTE, $PORT, $urlServeur, &$cookie) {
16. ...
17. }
```

Commentaires

- ligne 15 : la fonction *connecte* est identique à celle du script client précédent

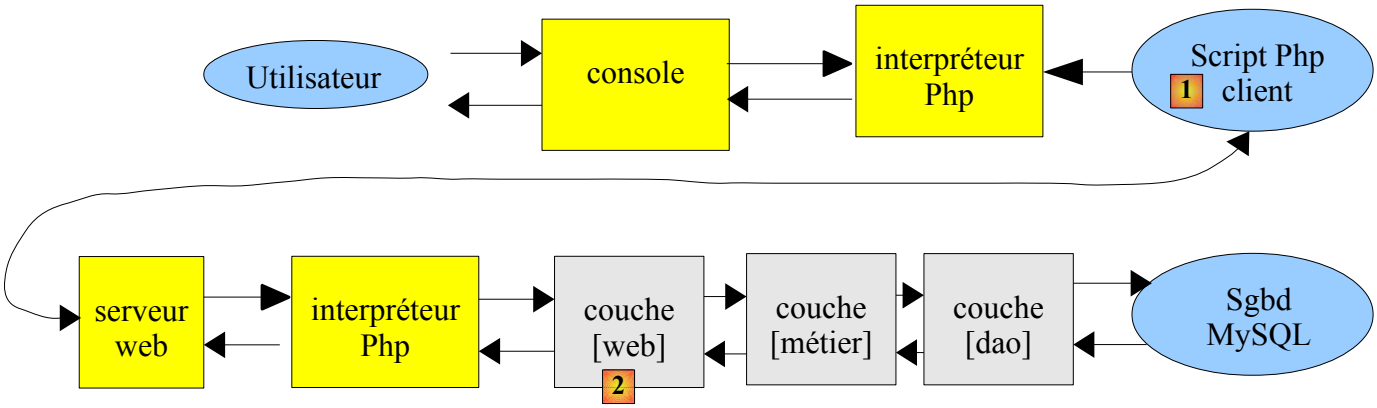
Résultats

```
1.  HTTP/1.1 200 OK
2.  ...
3.
4.  personne=[paul,langévin,10]
5.
6.  HTTP/1.1 200 OK
7.  ...
8.
9.  personne=[paul,langévin,11]
10.
11. HTTP/1.1 200 OK
12. ...
13.
14. personne=[paul,langévin,12]
15.
16. HTTP/1.1 200 OK
17. ...
18.
19. personne=[paul,langévin,13]
20.
21. HTTP/1.1 200 OK
22. ...
23.
24. personne=[paul,langévin,14]
```

11 Exercice IMPOTS avec un service WEB et une architecture à trois couches

Nous allons reprendre l'exercice IMPOTS (cf pages 24, 26, 40) et allons en faire une application client / serveur. Le script serveur sera décomposé en trois éléments :

- une couche appelée [dao] (Data Access Objects) qui s'occupera des échanges avec la base de données MySQL
- une couche appelée [métier] qui fera le calcul de l'impôt
- une couche [web] qui s'occupera des échanges avec les clients web.



Le script client [1] :

- transmet au script serveur les trois informations ($marié, $enfants, $salaire) nécessaires au calcul de l'impôt
- affiche la réponse du serveur sur la console

Le script serveur [2] est constitué par la couche [web] du serveur.

- lors du début d'une nouvelle session client, il placera dans des tableaux les données de la base de données MySQL [dbimpots]. Pour cela, il fera appel à la couche [dao]. Les tableaux ainsi construits seront placés dans la session du client afin d'être utilisables dans les requêtes ultérieures du client.
- lors d'une requête client, il passera les trois informations ($marié, $enfants, $salaire) à la couche [métier] qui calculera l'impôt $impot.
- le script serveur renverra l'impôt $impôt calculé.

11.1 Le script client (clients_impots_05_web)

Le script client sera un client du service web de calcul de l'impôt. Il postera (POST) au serveur des paramètres sous la forme : **params=$marié,$enfants,$salaire** où

- $*marié* sera la chaîne *oui* ou *non*,
- $*enfants* sera le nombre d'enfants,
- $*salaire* le salaire du contibuable

Il trouve les trois paramètres précédents dans un fichier texte [data.txt] sous la forme (marié, enfants, salaire) :

```
oui,2,200000
non,2,200000
oui,3,200000
non,3,200000
oui,5,50000
non,0,3000000
```

Le script client

- lira le fichier texte [data.txt] ligne par ligne
- postera la chaîne **params=$marié,$enfants,$salaire** au service web de calcul de l'impôt
- récupèrera la réponse du service. Celle-ci pourra avoir deux formes :

```
<erreur>message d'erreur</erreur>
<impot>montant de l'impôt</impot>
```

- enregistrera la réponse du serveur dans un fichier texte [resultats.txt] sous l'une des deux formessuivantes :

```
marié:enfants:salaire:erreur
marié:enfants:salaire:impôt
```

Le code du script client est le suivant :

```php
<?php

// client impôts
// gestion des erreurs
ini_set("display_errors", "off");

// ------------------------------------------------------------------------
// une classe de fonctions utilitaires
class Utilitaires {

  function cutNewLinechar($ligne) {
    ...
  }

}

// main ----------------------------------------------------
// définition des constantes
$DATA = "data.txt";
$RESULTATS = "resultats.txt";
// données serveur
$HOTE = "localhost";
$PORT = 80;
$urlServeur = "/exemples-web/impots_05_web.php";

// les paramètres des personnes imposables (statut marital, nombre d'enfants, salaire annuel)
// ont été placés dans le fichier texte $DATA à raison d'une ligne par contribuable
// les résultats (statut marital, nombre d'enfants, salaire annuel, impôt à payer)
// ou (statut marital, nombre d'enfants, salaire annuel, msg d'erreur) sont placés dans
// le fichier texte $RESULTATS à raison d'un résultat par ligne

// classe Utilitaires
$u = new Utilitaires();

// ouverture fichier des données des contribuables
$data = fopen($DATA, "r");
if (!$data) {
  print "Impossible d'ouvrir en lecture le fichier des données [$DATA]\n";
  exit;
}

// ouverture fichier des résultats
$résultats = fopen($RESULTATS, "w");
if (!$résultats) {
  print "Impossible de créer le fichier des résultats [$RESULTATS]\n";
  exit;
}

// on exploite la ligne courante du fichier des données contribuables
while ($ligne = fgets($data, 100)) {
  // on enlève l'éventuelle marque de fin de ligne
  $ligne = $u->cutNewLineChar($ligne);
  // on récupère les 3 champs marié:enfants:salaire qui forment $ligne
  list($marié, $enfants, $salaire) = explode(":", $ligne);
  // on calcule l'impôt
  list($erreur, $impôt) = calculerImpot($HOTE, $PORT, $urlServeur, $cookie, array($marié, $enfants, $salaire));
  // on inscrit le résultat
  $résultat = $erreur ? "$marié:$enfants:$salaire:$erreur" : "$marié:$enfants:$salaire:$impôt";
  fputs($résultats, "$résultat\n");
  // donnée suivante
```

```
61.  }
62.  // on ferme les fichiers
63.  fclose($data);
64.  fclose($résultats);
65.
66.  // fin
67.  print "Terminé...\n";
68.  exit;
69.
70.  function calculerImpot($HOTE, $PORT, $urlServeur, &$cookie, $params) {
71.    // connecte le client à ($HOTE,$PORT,$urlServeur)
72.    // envoie le cookie $cookie si celui-ci est non vide. $cookie est passé par référence
73.    // envoie $params au serveur
74.    // exploite l'unique ligne renvoyée par le serveur
75.
76.    // ouverture d'une connexion sur le port 80 de $HOTE
77.    $connexion = fsockopen($HOTE, $PORT);
78.    // erreur ?
79.    if (!$connexion)
80.      return array("erreur lors de la connexion au serveur ($HOTE, $PORT)");
81.    // les entêtes (headers) du protocole HTTP doivent se terminer par une ligne vide
82.    // POST
83.    fputs($connexion, "POST $urlServeur HTTP/1.1\n");
84.    // Host
85.    fputs($connexion, "Host: localhost\n");
86.    // Connection
87.    fputs($connexion, "Connection: close\n");
88.  // on envoie le cookie s'il est non vide
89.    if ($cookie) {
90.      fputs($connexion, "Cookie: $cookie\n");
91.    }//if
92.    // maintenant on envoie l'instruction client après l'avoir encodée
93.    $infos = "params=" . urlencode(implode(",", $params));
94.    // on indique quel type d'informations on va envoyer
95.    fputs($connexion, "Content-type: application/x-www-form-urlencoded\n");
96.    // on envoie la taille (nombre de caractères) des infos qui vont être envoyées
97.    fputs($connexion, "Content-length: " . strlen($infos) . "\n");
98.    // on envoie une ligne vide
99.    fputs($connexion, "\n");
100.   // on envoie les infos
101.   fputs($connexion, $infos);
102.   // on affiche la réponse du serveur web
103.   // et on prend soin de récupérer l'éventuel cookie
104.   while ($ligne = fgets($connexion, 1000)) {
105.     // cookie - seulement lors de la 1ère réponse
106.     if (!$cookie) {
107.       if (preg_match("/^Set-Cookie: (.*?)\s*$/", $ligne, $champs)) {
108.         $cookie = $champs[1];
109.       }//if
110.     }
111.     // dès qu'on a une ligne vide la réponse Http est terminée
112.     if (trim($ligne) == "") {
113.       break;
114.     }
115.   }//while
116.   // lecture ligne du résultat
117.   $ligne = fgets($connexion, 1000);
118.   // on ferme la connexion
119.   fclose($connexion);
120.   // calcul résultat
121.   $erreur="";
122.   $impôt="";
123.   if (preg_match("/^<erreur>(.*?)<\/erreur>\s*$/", $ligne, $champs)) {
124.     $erreur = $champs[1];
125.   } else {
126.     if (preg_match("/^<impot>(.*?)<\/impot>\s*$/", $ligne, $champs)) {
```

```
127.     $impôt = $champs[1];
128.   }else{
129.     $erreur="résultat du serveur non exploitable";
130.   }
131. }
132. // retour
133. return array($erreur, $impôt);
134. }
```

Commentaires

Le code du script client reprend des choses déjà vues :

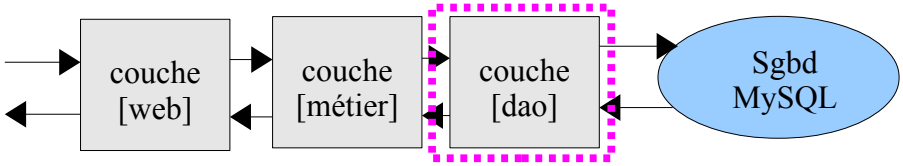
- lignes 9-15 : la classe [Utilitaires] a été présentée dans la version 3 page 40
- lignes 17-68 : le programme principal est analogue à celui de la version 1 page 24. Il n'en diffère que par le calcul de l'impôt, ligne 56.
- ligne 56 : la fonction de calcul de l'impôt admet les paramètres suivants :
  - $*HOTE*, $*PORT*, $*urlServeur* : permettent de se connecter au service web
  - $*cookie* : est le cookie de session. Ce paramètre est passé par référence. Sa valeur est fixée par la fonction de calcul de l'impôt. Lors du 1er appel, il n'a pas de valeur. Ensuite il en a une.
  - ***array**($marié, $enfants, $salaire)* : représente une ligne du fichier [data.txt]

  La fonction de calcul de l'impôt rend un tableau de deux résultats ($*erreur*, $*impôt*) où $*erreur* est un éventuel message d'erreur et $*impôt* le montant de l'impôt.

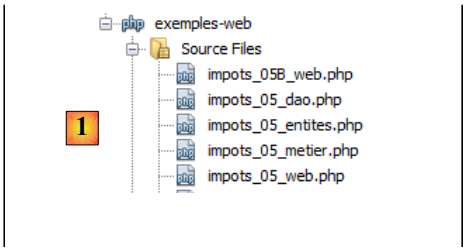
- lignes 70 – 134 : on a là un classique client Http comme nous en avons beaucoup rencontré. On notera les points suivants :
- ligne 83 : les paramètres ($marié, $enfants, $salaire) sont transmis au serveur par un POST
- lignes 89-91 : si le client a un identifiant de session, il l'envoie au serveur
- ligne 93 : création du paramètre *params*
- ligne 101 : envoi du paramètre *params*
- lignes 104-115 : le client lit tous les en-têtes Http envoyés par le serveur jusqu'à rencontrer la ligne vide de fin des en-têtes. Il en profite pour récupérer l'identifiant de session lors de la réponse à sa première demande.
- lignes 123-125 : on exploite une éventuelle ligne de la forme *<erreur>message</erreur>*
- lignes 126-128 : on fait de même avec une éventuelle ligne de la forme *<impot>montant</impot>*
- ligne 133 : on rend le résultat

11.2 Le service web de calcul de l'impôt

Nous nous intéressons ici aux trois scripts qui composent le serveur :



Le projet Netbeans coorespondant est le suivant :



En [1], le serveur est formé des scripts Php suivants :

- [impots_05_entites] contient les classes utilisées par le serveur
- [impots_05_dao] contient les classes et interfaces de la couche [dao]

- [impots_05_metier] contient les classes et interfaces de la couche [metier]
- [impots_05_web] contient les classes et interfaces de la couche [dao]

Nous commençons par présenter deux classes utilisées par les différentes couches du service web.

11.2.1 Les entités du service web (impots_05_entites)

La base MySQL [dbimpots] a une table [impots] qui contient les données nécessaires au calcul de l'impôt [1] :

limites	coeffR	coeffN
12620.00	0.00	0.00
13190.00	0.05	631.00
15640.00	0.10	1290.50
24740.00	0.15	2072.50
31810.00	0.20	3309.50
39970.00	0.25	4900.00
48360.00	0.30	6898.50
55790.00	0.35	9316.50
92970.00	0.40	12106.00
127860.00	0.45	16754.50
151250.00	0.50	23147.50
172040.00	0.55	30710.00
195000.00	0.60	39312.00
0.00	0.65	49062.00

1

Nous stockerons les données de la table MySQL [impots] dans un tableau d'objets *Tranche* où *Tranche* est la classe suivante :

```
1.  <?php
2.
3.  // une tranche d'impôt
4.  class Tranche {
5.
6.    // champs privés
7.    private $limite;
8.    private $coeffR;
9.    private $coeffN;
10.
11.   // getters et setters
12.   public function getLimite() {
13.     return $this->limite;
14.   }
15.
16.   public function setLimite($limite) {
17.     $this->limite = $limite;
18.   }
19.
20.   public function getCoeffR() {
21.     return $this->coeffR;
22.   }
23.
24.   public function setCoeffR($coeffR) {
25.     $this->coeffR = $coeffR;
26.   }
27.
28.   public function getCoeffN() {
29.     return $this->coeffN;
30.   }
31.
32.   public function setCoeffN($coeffN) {
33.     $this->coeffN = $coeffN;
```

```
34.   }
35.
36.   // constructeur
37.   public function __construct($limite, $coeffR, $coeffN) {
38.     $this->setLimite($limite);
39.     $this->setCoeffR($coeffR);
40.     $this->setCoeffN($coeffN);
41.   }
42.
43.   // toString
44.   public function __toString(){
45.     return "[$this->limite,$this->coeffR,$this->coeffN]";
46.   }
47. }
```

Les champs privés [$limite, $coeffR, $coeffN] serviront à stocker les colonnes [limites, coeffR, coeffN] d'une ligne de la table MySQL [impots].

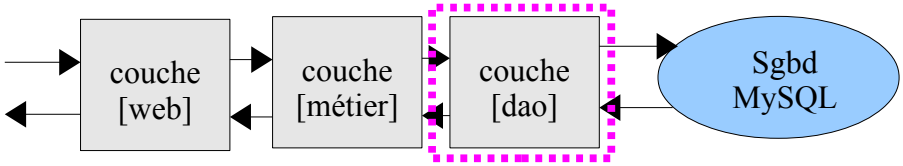
Par ailleurs, le code serveur utilisera une exception qui lui sera propre, la classe *ImpotsException* :

```
1.  class ImpotsException extends Exception {
2.
3.    public function __construct($message, $code=0) {
4.      parent::__construct($message, $code);
5.    }
6.
7.  }
```

- ligne 1 : la classe [ImpotsException] dérive de la classe [Exception] prédéfinie dans Php 5
- ligne 3 : le constructeur de la classe [ImpotsException] admet deux paramètres :
  - $*message* : un message d'erreur
  - $*code* : un code d'erreur

11.2.2 La couche [dao] (impots_05_dao)

La couche [dao] assure l'accès aux données de la base de données :



La couche [dao] présente l'interface suivante :

```
1.  // interface Dao
2.  interface IImpotsDao {
3.
4.  // rend un tableau d'entités Tranche
5.    function getData();
6.  }
```

L'interface *IImpotsDao* n'expose que la fonction *getData*. Cette fonction place dans un tableau d'objets *Tranche*, les différentes lignes de la table MySQL [dbimpots.impots].

La classe d'implémentation est la suivante :

```
1.  <?php
2.
3.  // couche Dao
4.  // dépendances
5.  require_once "impots_05_entites.php";
6.  // constantes
```

```php
define("TABLE", "impots");

// ----------------------------------------------------------------
// implémentation abstraite
abstract class ImpotsDaoWithPdo implements IImpotsDao {

// champs privés
  private $dsn;
  private $user;
  private $passwd;
  private $tranches;

// getters et setters
  public function getDsn() {
    return $this->dsn;
  }

  public function setDsn($dsn) {
    $this->dsn = $dsn;
  }

  public function getUser() {
    return $this->user;
  }

  public function setUser($user) {
    $this->user = $user;
  }

  public function getPasswd() {
    return $this->passwd;
  }

  public function setPasswd($passwd) {
    $this->passwd = $passwd;
  }

  // constructeur
  public function __construct($dsn, $user, $passwd) {
    // on enregistre les paramètres
    $this->setDsn($dsn);
    $this->setUser($user);
    $this->setPasswd($passwd);
    // on récupère les données du Sgbd
    // connecte ($user,$pwd) à la base $dsn
    try {
      // connexion
      $connexion = new PDO($dsn, $user, $passwd, array(PDO::ATTR_PERSISTENT => true));
      // lecture de la table $TABLE
      $requête = "select limites,coeffR,coeffN from " . TABLE;
      // exécute la requête $requête sur la connexion $connexion
      $statement = $connexion->prepare($requête);
      $statement->execute();
      // exploitation du résultat de la requête
      while ($colonnes = $statement->fetch()) {
        $this->tranches[] = new Tranche($colonnes[0], $colonnes[1], $colonnes[2]);
      }
      // déconnexion
      $connexion=NULL;
    } catch (PDOException $e) {
      // retour avec erreur
      throw new ImpotsException($e->getMessage(), 1);
    }
  }

  public function getData(){
```

```
73.     return $this->tranches;
74.   }
75. }
```

- ligne 5 : l'implémentation de l'interface [IImpotsDao] a besoin des classes définies dans le script [impots_05_entites].
- ligne 11 : définition d'une classe abstraite. Une classe abstraite est une classe qu'on ne peut instancier. Une classe abstraite doit être obligatoirement dérivée pour être instanciée. Une classe peut être déclarée abstraite parce qu'on ne **peut** pas l'instancier (certaines de ses méthodes ne sont pas définies) ou qu'on ne **veut** pas instancier. Ici, on ne veut pas instancier la classe [ImpotsDaoWithPdo]. On instanciera des classes dérivées.
- ligne 11 : la classe [ImpotsDaoWithPdo] implémente l'interface [IImpotsDao]. Elle doit donc définir la méthode *getData*. On trouve cette méthode lignes 72-74.
- ligne 14 : $*dsn* (Data Source Name) est une chaîne de caractères qui identifie de façon unique le Sgbd et la base de données utilisée.
- ligne 15 : $*user* identifie l'utilisateur qui se connecte à la base de données
- ligne 16 : $*passwd* est le mot de passe de l'utilisateur précédent
- ligne 17 : $*tranches* est le tableau d'objets *Tranche* dans lequel on va mémoriser la table MySQL [dbimpots.impots].
- lignes 45-70 : le constructeur de la classe. Ce code a déjà été rencontré dans la version 4, page 61. On notera que la construction de l'objet [ImpotsDaoWithPdo] peut échouer. Une exception de type [ImpotsException] est alors lancée.
- lignes 72-74 : la méthode [getData] de l'interface [IImpotsDao].

La classe [ImpotsDaoWithPdo] convient à tout Sgbd. Le constructeur de la classe, ligne 45, impose de connaître le *Data Source Name* de la base de données. Cette chaîne de caractères dépend du Sgbd utilisé. On choisit de ne pas imposer à l'utilisateur de la classe de connaître ce *Data Source Name*. Pour chaque Sgbd, il y aura une classe particulière dérivée de [ImpotsDaoWithPdo]. Pour le Sgbd MySQL, ce sera la classe suivante :

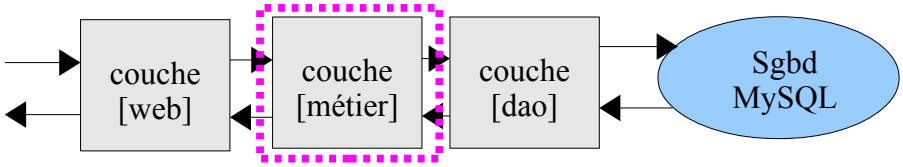
```
1.  class ImpotsDaoWithMySQL extends ImpotsDaoWithPdo {
2.
3.    public function __construct($host, $port, $base, $user, $passwd) {
4.      parent::__construct("mysql:host=$host;dbname=$base;port=$port", $user, $passwd);
5.    }
6.
7.  }
```

- ligne 3, le constructeur ne demande pas le *Data Source Name*, mais simplement le nom de la machine hôte du Sgbd ($host), son port d'écoute ($port) et le nom de la base de données ($base).
- ligne 4, le *Data Source Name* de la base de données MySQL est construit et utilisé pour appeler le constructeur de la classe parente.

On notera que pour s'adapter à un autre Sgbd, il suffit d'écrire la classe dérivée de [ImpotsDaoWithPdo] qui convient. Il s'agit à chaque fois de construire le *Data Source Name* propre au Sgbd utilisé.

11.2.3 La couche [métier] (impots_05_metier)

La couche [metier] contient la logique du calcul de l'impôt :



La couche [métier] présente l'interface suivante :

```
1.  <?php
2.
3.  // interface metier
4.  interface IImpotsMetier {
5.    public function calculerImpot($marié, $enfants, $salaire);
6.  }
```

L'interface [IImpotsMetier] n'expose qu'une méthode, la méthode [calculerImpot] qui permet de calculer l'impôt d'un contribuable à partir des paramètres suivants :

- $*marié* : chaîne oui / non selon que le contribuable est marié ou non

- $*enfants* : le nombre d'enfants du contribuable
- $*salaire* : son salaire

C'est la couche [web] qui lui fournira ces paramètres.

L'implémentation de l'interface [IImpotsMetier] est la suivante :

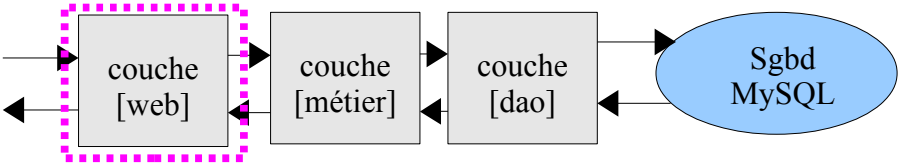
```
1.  // dépendances
2.  require_once "impots_05_dao.php";
3.
4.  // ----------------------------------------------------------------
5.  // classe d'implémentation
6.  class ImpotsMetier implements IImpotsMetier {
7.
8.    // couche dao
9.    private $dao;
10.   // tableau d'objets [Tranche]
11.   private $data;
12.
13.   // getter et setter
14.   public function getDao() {
15.     return $this->dao;
16.   }
17.
18.   public function setDao($dao) {
19.     $this->dao = $dao;
20.   }
21.
22.   public function setData($data){
23.     $this->data=$data;
24.   }
25.
26.   public function __construct($dao) {
27.     // on récupère les données nécessaires au calcul de l'impôt
28.     $this->setDao($dao);
29.     $this->setData($this->dao->getData());
30.   }
31.
32.   public function calculerImpot($marié, $enfants, $salaire) {
33.     // $marié : oui, non
34.     // $enfants : nombre d'enfants
35.     // $salaire : salaire annuel
36.     // nombre de parts
37.     $marié = strtolower($marié);
38.     if ($marié == "oui")
39.       $nbParts = $enfants / 2 + 2;
40.     else
41.       $nbParts=$enfants / 2 + 1;
42.     // une 1/2 part de plus si au moins 3 enfants
43.     if ($enfants >= 3)
44.       $nbParts+=0.5;
45.     // revenu imposable
46.     $revenuImposable = 0.72 * $salaire;
47.     // quotient familial
48.     $quotient = $revenuImposable / $nbParts;
49.     // est mis à la fin du tableau limites pour arrêter la boucle qui suit
50.     $N = count($this->data);
51.     $this->data[$N - 1]->setLimite($quotient);
52.     // calcul de l'impôt
53.     $i = 0;
54.     while ($i < $N and $quotient > $this->data[$i]->getLimite()) {
55.       $i++;
56.     }
57.     // du fait qu'on a placé $quotient à la fin du tableau $limites, la boucle précédente
58.     // ne peut déborder du tableau $limites
59.     // maintenant on peut calculer l'impôt
60.     return floor($revenuImposable * $this->data[$i]->getCoeffR() - $nbParts * $this->data[$i]->getCoeffN());
```

```
61.   }
62.
63. }
```

- ligne 2 : la couche [métier] a besoin des classes de la couche [dao] et des entités (*Tranche*, *ImpotsException*).
- ligne 6 : la classe [ImpotsMetier] implémente l'interface [IimpotsMetier].
- lignes 9-11 : les champs privés de la classe :
  - $*dao* : référence sur la couche [dao]
  - $*data* : tableau d'objets de type [Tranche] fourni par la couche [dao]
- lignes 26-30 : le constructeur de la classe initialise les deux champs précédents. Il reçoit comme paramètre une référence sur la couche [dao].
- lignes 32-61 : implémentation de la méthode [calculerImpot] de l'interface [IimpotsMetier]. Cette méthode a été rencontrée dès la version 1 (page 24).

11.2.4 La couche [web] (impots_05_web)

La couche [metier] contient la logique du calcul de l'impôt :



La couche [web] est constitué du service web qui répond aux clients web. On rappelle que ceux-ci font une demande au service web en postant le paramètre suivant : *params=marié,enfants,salaire*. On a affaire à un service web tel que nous avons pu les construire dans les paragraphes précédents. Son code est le suivant :

```
1.  <?php
2.
3.  // couche métier
4.  require_once "impots_05_metier.php";
5.
6.  // gestion des erreurs
7.  ini_set("display_errors", "off");
8.
9.  // entête UTF-8
10. header("Content-Type: text/plain; charset=utf-8");
11.
12. // ------------------------------------------------------------------------
13. // le service web des impôts
14. // définition des constantes
15. $HOTE = "localhost";
16. $PORT = 3306;
17. $BASE = "dbimpots";
18. $USER = "root";
19. $PWD = "";
20. // les données nécessaires au calcul de l'impôt ont été placées dans la table mysql IMPOTS
21. // appartenant à la base $BASE. La table a la structure suivante
22. // limites decimal(10,2), coeffR decimal(6,2), coeffN decimal(10,2)
23. // les paramètres des personnes imposables (statut marital, nombre d'enfants, salaire annuel)
24. // sont envoyés par le client sous la forme params=statut marital, nombre d'enfants, salaire annuel
25. // les résultats (statut marital, nombre d'enfants, salaire annuel, impôt à payer) sont renvoyés au client
26. // sous la forme <impot>impot</impot>
27. // ou sous la forme <erreur>erreur</erreur>, si les paramètres sont invalides
28.
29. // on récupère la couche [métier] dans la session
30. session_start();
31. if (!isset($_SESSION['metier'])) {
32. // instanciation de la couche [dao] et de la couche [métier]
33.   try {
34.     $_SESSION['metier'] = new ImpotsMetier(new ImpotsDaoWithMySQL($HOTE, $PORT, $BASE, $USER, $PWD));
35.   } catch (ImpotsException $ie) {
```

```
36.     print "<erreur>Erreur : " . utf8_encode($ie->getMessage() . "</erreur>");
37.     exit;
38.   }
39. }
40. $metier = $_SESSION['metier'];
41.
42. // on récupère la ligne envoyée par le client
43. $params = utf8_encode(htmlspecialchars(strtolower(trim($_POST['params']))));
44. $items = explode(",", $params);
45. // il ne doit y avoir que 3 paramètres
46. if (count($items) != 3) {
47.   print "<erreur>[$params] : nombre de paramètres invalides</erreur>\n";
48.   exit;
49. }//if
50. // le premier paramètre (statut marital) doit être oui/non
51. $marié = trim($items[0]);
52. if ($marié != "oui" and $marié != "non") {
53.   print "<erreur>[$params] : 1er paramètre invalide</erreur>\n";
54.   exit;
55. }//if
56. // le second paramètre (nbre d'enfants) doit être un nombre entier
57. if (!preg_match("/^\s*(\d+)\s*$/", $items[1], $champs)) {
58.   print "<erreur>[$params] : 2ième paramètre invalide</erreur>\n";
59.   exit;
60. }//if
61. $enfants = $champs[1];
62. // le troisième paramètre (salaire) doit être un nombre entier
63. if (!preg_match("/^\s*(\d+)\s*$/", $items[2], $champs)) {
64.   print "<erreur>[$params] : 3ième paramètre invalide</erreur>\n";
65.   exit;
66. }//if
67. $salaire = $champs[1];
68. // on calcule l'impôt
69. $impôt = $metier->calculerImpot($marié, $enfants, $salaire);
70. // on renvoie le résultat
71. print "<impot>$impôt</impot>\n";
72. // fin
73. exit;
```

- ligne 4 : la couche [web] a besoin des classes de la couche [métier]
- lignes 30-40 : la référence sur la couche [métier] est mise en session. Si on se rappelle que cette couche [métier] a une référence sur la couche [dao] et que cette dernière mémorise les données du Sgbd, on comprend alors :
  - que la 1ère requête du client va provoquer un accès au Sgbd
  - que les requêtes suivantes du même client vont utiliser les données mémorisées par la couche [dao]. Il n'y a donc pas d'accès au Sgbd.
- ligne 34 : construction d'une couche [métier] travaillant avec une couche [dao] implémentée pour le Sgbd MySQL
- lignes 35-37 : gestion d'une éventuelle erreur dans l'opération précédente. Dans ce cas, une ligne *<erreur>message</erreur>* est envoyé au client.
- ligne 43 : on récupère le paramètre 'params' qui a été posté par le client.
- lignes 46-49 : on vérifie le nombre d'informations trouvées dans 'params'
- lignes 51-55 : on vérifie la validité de la 1ère information
- lignes 56-60 : idem pour la 2ième
- lignes 62-66 : idem pour la 3ième
- ligne 69 : c'est la couche [métier] qui calcule l'impôt.
- ligne 71 : envoi du résultat au client

Résultats

On se rappelle que le client [client_impots_web_05] exploite le fichier [data.txt] suivant :

```
oui,2,200000
non,2,200000
oui,3,200000
non,3,200000
oui,5,50000
non,0,3000000
```

A partir de ces lignes (marié, enfants, salaire), le client interroge le serveur de calcul d'impôt et inscrit les résultats dans le fichier texte [resultats.txt]. Après exécution du client, le contenu de ce fichier est le suivant :

```
oui:2:200000:22504
non:2:200000:33388
oui:3:200000:16400
non:3:200000:22504
oui:5:50000:0
non:0:3000000:1354938
```

où chaque ligne est de la forme (marié, enfants, salaire, impôt calculé).

12 Traitement de documents XML (simplexml_01)

Nous considérons le fichier [data.xml] suivant :

```
<?xml version="1.0" encoding="UTF-8"?>

<tribu>
 <enseignant>
  <personne sexe="M">
   <nom>dupont</nom>
   <prenom>jean</prenom>
   <age>28</age>
ceci est un commentaire
  </personne>
  <section>27</section>
 </enseignant>
 <etudiant>
  <personne sexe="F">
   <nom>martin</nom>
   <prenom>charline</prenom>
   <age>22</age>
  </personne>
  <formation>dess IAIE</formation>
 </etudiant>
</tribu>
```

Nous analysons ce document avec le script suivant :

```
1.  <?php
2.
3.  // fichier XML à exploiter
4.  $FILE_NAME = "data.xml";
5.  // exploitation
6.  $xml = simplexml_load_file($FILE_NAME);
7.  print_r($xml);
8.  print_r($xml->enseignant->personne['sexe']);
9.  $nom=$xml->enseignant->personne->nom;
10. print "nom=$nom\n";
11. $sexe=$xml->enseignant->personne['sexe'];
12. print "sexe=$sexe\n";
13. $formation=$xml->etudiant->formation;
14. print "formation=$formation\n";
15. print "isset=".isset($xml->enseignant->personne->nom)."\n";
16. print "isset=".isset($xml->enseignant->personne->xx)."\n";
```

Nous utilisons ici un module Php appelé [simpleXML] qui permet d'exploiter des documents XML.

- ligne 6 : chargement du fichier XML
- ligne 7 : affichage du document XML
- ligne 8 : affichage de la valeur de l'attribut '*sexe*' d'une personne enseignante : <*enseignant*><*personne sexe='...'*>
- ligne 9 : affichage de la valeur de la 1ère balise <*enseignant*><*personne*><*nom*>

Résultats

```
1.  SimpleXMLElement Object
2.  (
3.      [enseignant] => SimpleXMLElement Object
4.          (
5.              [personne] => SimpleXMLElement Object
6.                  (
7.                      [@attributes] => Array
8.                          (
9.                              [sexe] => M
10.                         )
11.
12.                     [nom] => dupont
13.                     [prenom] => jean
14.                     [age] => 28
15.                 )
16.
17.             [section] => 27
18.         )
19.
20.     [etudiant] => SimpleXMLElement Object
21.         (
22.             [personne] => SimpleXMLElement Object
23.                 (
24.                     [@attributes] => Array
25.                         (
26.                             [sexe] => F
27.                         )
28.
29.                     [nom] => martin
30.                     [prenom] => charline
31.                     [age] => 22
32.                 )
33.
34.             [formation] => dess IAIE
35.         )
36.
37. )
38. SimpleXMLElement Object
39. (
40.     [0] => M
41. )
42. nom=dupont
43. sexe=M
44. formation=dess IAIE
45. isset=1
46. isset=
```

- lignes 1-37 : le document XML sous la forme d'un objet de type [simpleXML].

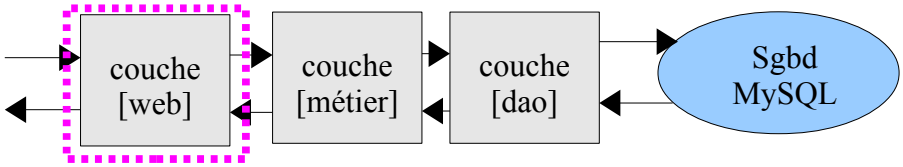
Le script précédent ne nous montre toutes les possibilités du module [simpleXML] mais il nous suffit pour écrire une dernière version de l'exercice IMPOTS.

13 Exercice IMPOTS avec XML

Dans cet exercice déjà étudié de nombreuses fois, le serveur renvoie les résultats au client sous la forme d'un flux XML.

13.1 Le serveur (impots_05B_web)

Le service web de calcul de l'impôt écrit précédemment a une architecture en couches :



Dans l'exemple précédent, le service web envoyait à ses clients l'une des deux lignes suivantes :

```
<erreur>message d'erreur</erreur>
<impot>montant de l'impôt</impot>
```

Ce n'est pas un document XML bien formé car il n'a pas de balise racine. Le nouveau service web enverra sa réponse sous la forme suivante :

```
<reponse><erreur>message d'erreur</erreur></reponse>
```
ou
```
<reponse><impot>montant de l'impôt</impot></reponse>
```

Dans les deux cas, la réponse est un document XML bien formé qui peut être exploité par le module [simpleXML].

Dans le schéma à trois couches, seule la couche [web] doit être modifiée. Son code [impots_05B_web] devient le suivant :

```
1.  <?php
2.
3.  // couche métier
4.  require_once "impots_05_metier.php";
5.
6.  // gestion des erreurs
7.  ini_set("display_errors", "off");
8.
9.  // entête UTF-8
10. header("Content-Type: text/plain; charset=utf-8");
11.
12. // ----------------------------------------------------------------------------
13. // le service web des impôts
14. // définition des constantes
15. $HOTE = "localhost";
16. $PORT = 3306;
17. $BASE = "dbimpots";
18. $USER = "root";
19. $PWD = "";
20. // les données nécessaires au calcul de l'impôt ont été placées dans la table mysql $TABLE
21. // appartenant à la base $BASE. La table a la structure suivante
22. // limites decimal(10,2), coeffR decimal(6,2), coeffN decimal(10,2)
23. // les paramètres des personnes imposables (statut marital, nombre d'enfants, salaire annuel)
24. // sont envoyés par le client sous la forme params=statut marital, nombre d'enfants, salaire annuel
25. // les résultats (statut marital, nombre d'enfants, salaire annuel, impôt à payer) sont renvoyés au client
26. // sous la forme <impot>impot</impot>
27. // ou sous la forme <erreur>erreur</erreur>, si les paramètres sont invalides
28. // on récupère la couche [métier] dans la session
29. session_start();
30. if (!isset($_SESSION['metier'])) {
31. // instanciation de la couche [dao] et de la couche [métier]
32.   try {
```

```
33.     $_SESSION['metier'] = new ImpotsMetier(new ImpotsDaoWithMySQL($HOTE, $PORT, $BASE, $USER, $PWD));
34.   } catch (ImpotsException $ie) {
35.     print "<reponse><erreur>Erreur : " . utf8_encode($ie->getMessage() . "</erreur></reponse>");
36.     exit;
37.   }
38. }
39. $metier = $_SESSION['metier'];
40.
41. // on récupère la ligne envoyée par le client
42. $params = utf8_encode(htmlspecialchars(strtolower(trim($_POST['params']))));
43. //print "paramètres reçus --> $params\n";
44. $items = explode(",", $params);
45. // il ne doit y avoir que 3 paramètres
46. if (count($items) != 3) {
47.   print "<reponse><erreur>[$params] : nombre de paramètres invalides</erreur></reponse>\n";
48.   exit;
49. }//if
50. // le premier paramètre (statut marital) doit être oui/non
51. $marié = trim($items[0]);
52. if ($marié != "oui" and $marié != "non") {
53.   print "<reponse><erreur>[$params] : 1er paramètre invalide</erreur></reponse>\n";
54.   exit;
55. }//if
56. // le second paramètre (nbre d'enfants) doit être un nombre entier
57. if (!preg_match("/^\s*(\d+)\s*$/", $items[1], $champs)) {
58.   print "<reponse><erreur>[$params] : 2ième paramètre invalide</erreur></reponse>\n";
59.   exit;
60. }//if
61. $enfants = $champs[1];
62. // le troisième paramètre (salaire) doit être un nombre entier
63. if (!preg_match("/^\s*(\d+)\s*$/", $items[2], $champs)) {
64.   print "<reponse><erreur>[$params] : 3ième paramètre invalide</erreur></reponse>\n";
65.   exit;
66. }//if
67. $salaire = $champs[1];
68. // on calcule l'impôt
69. $impôt = $metier->calculerImpot($marié, $enfants, $salaire);
70. // on renvoie le résultat
71. print "<reponse><impot>$impôt</impot></reponse>\n";
72. // fin
73. exit;
```

Le format de la réponse du service web est modifiée aux lignes 35, 47, 53, 58, 64 et 71.

13.2 Le client (client_impots_05b_web)

Le client est modifié pour tenir compte du nouveau format de la réponse. On utilise [simpleXML] pour traiter celle-ci :

```
1.  <?php
2.
3.  // client impôts
4.  // gestion des erreurs
5.  ini_set("display_errors", "off");
6.
7.  // ------------------------------------------------------------------------
8.  // une classe de fonctions utilitaires
9.  class Utilitaires {
10. ...
11. }
12.
13. // main ------------------------------------------------
14. // définition des constantes
15. $DATA = "data.txt";
16. $RESULTATS = "resultats.txt";
```

```
17.  // données serveur
18.  $HOTE = "localhost";
19.  $PORT = 80;
20.  $urlServeur = "/exemples-web/impots_05B_web.php";
21.
22.  ...
23.  exit;
24.
25.  function calculerImpot($HOTE, $PORT, $urlServeur, &$cookie, $params) {
26.    // connecte le client à ($HOTE,$PORT,$urlServeur)
27.  // envoie le cookie $cookie si celui-ci est non vide. $cookie est passé par référence
28.  // envoie $params au serveur
29.  // exploite la ligne du résultat
30.  ...
31.    // lecture ligne du résultat
32.    $ligne = fgets($connexion, 1000);
33.    // on ferme la connexion
34.    fclose($connexion);
35.    // calcul résultat
36.    $xml = new SimpleXMLElement($ligne);
37.    $erreur = isset($xml->erreur) ? $xml->erreur : "";
38.    $impôt = isset($xml->impot) ? $xml->impot : "";
39.    // retour
40.    return array($erreur, $impôt);
41.  }
```

- ligne 32 : la réponse du serveur est lue. C'est un document XML.
- ligne 36 : un objet SimpleXML est construit à partir du document XML reçu.
- ligne 37 : l'éventuel message d'erreur
- ligne 38 : l'éventuel montant de l'impôt