LE LANGAGE PHP

P. VÉRON

Table des matières

1. Introduction

Php est un langage permettant de construire dynamiquement des pages Html contenant les résultats de calculs ou de requêtes SQL adressées à un système de gestion de bases de données (SGBD). On doit la première version de Php à Rasmus Lerdorf qui l'a mise au point pour ses propres besoins en 1994. Cette version était destinée à son usage personnel, d'où le nom (Personal Home Pages). Initialement développé en Perl, il a été complètement réécrit en C. Comme pour JavaScript, les instructions de ce langage de commandes peuvent être intégrées dans le code d'un document Html. Elles sont alors exécutées par le serveur Web qui héberge la page (comme pour un script CGI) via l'interprète Php, ou bien directement par le démon http puisque Php peut être recompilé pour devenir un module du serveur Apache. Attention, il ne faut pas confondre Php et JavaScript. Il s'agit de deux choses diffrentes : Php est un langage de script destiné à être exécuté par le serveur, alors que le JavaScript est chargé et exécuté dans le navigateur. Php est donc comparable à l'ASP de Microsoft dans le sens où il s'exécutent tous les deux "côté serveur" (la comparaison s'arrête là : Php comporte beaucoup plus de fonctions qu'ASP, supporte pratiquement tous les standards du Web, et est extensible). Le langage Php suit plus ou moins la syntaxe du

Date: Mars 2013.

langage C. De plus il intègre différentes primitives permettant de se connecter à une base de données, ce qui permet d'inclure dans du code Html des requêtes SQL. Il est ainsi possible d'interroger facilement via le Web une base de données. Actuellement Php interagit avec la quasi totalité des SGBD du marché, qu'ils soient commerciaux ou qu'ils viennent du monde du Free Software : Oracle, Sybase, mSQL, MySQL, Solid, Generic ODBC, PostgreSQL, Adabas D, Filepro, Velocis, dBase, Unix dbm, ...
Php est distribué librement et gratuitement sous la licence Gnu Gpl. Il peut être téléchargé depuis le site mirroir de l'Université de Michel de Montaigne Bordeaux III à `http://fr.php.net`.

2. La balise <SCRIPT> ou < ?php ?>

L'insertion de code Php dans une page Html se fait à l'aide de la balise <SCRIPT>, on place généralement une instruction par ligne. Une instruction est terminée par un point-virgule en fin de ligne.

```
<SCRIPT language="php">
...
code PHP
...
</SCRIPT>
```

Exemple 1.

```
<HTML>
<HEAD><TITLE> Exemple </TITLE></HEAD>
<BODY>
<SCRIPT language="php">
print '<B> Bonjour le monde </B>';
</SCRIPT>
</BODY>
</HTML>
```

Remarque 1. On peut aussi utiliser la syntaxe suivante :

```
<?php
.
Code PHP
.
?>
```

Deux autres syntaxes sont utilisables mais elles nécessitent la configuration de certaines options dans le fichier d'initialisation de php. Le fait que ces options soient actives ou non dépend du package que vous utilisez, c'est pourquoi nous ne les mentionnons pas.

Supposons que la portion de code Html ci-dessus corresponde au fichier `hello.html` qui réside sur le serveur `xxx`. Lorsque l'on rentre l'adresse `http//xxx/hello.html` sous un navigateur quelconque (firefox par exemple), une requête est envoyée au serveur `xxx`. Celui-ci consulte le fichier `hello.html` et détecte la présence de code Php, il fait alors appel à l'interprète Php qui exécute les instructions et renvoie le résultat sur la sortie

standard (ici la chaine `<B> Bonjour le monde </B>`). Le serveur renvoie alors le tout (i.e. le document et le résultat de l'exécution du script) sur l'entrée standard du navigateur. Notez bien que votre navigateur ne fait que recevoir du code HTML qu'il sera capable d'interpréter correctement. Ainsi dans le cadre de notre exemple, le navigateur recevra :

```
<HTML>
<HEAD><TITLE> Exemple </TITLE></HEAD>
<BODY>
<B> Bonjour le monde </B>
</BODY>
</HTML>
```

et affichera donc le message **Bonjour le monde** en gras.

REMARQUE IMPORTANTE 2. Afin que le serveur lance l'exécution de l'interprète PHP, il faut en réalité que l'extension du fichier source ne soit pas `html` mais `php`. Le fichier de l'exemple 1 doit donc s'appeler `hello.php`.

3. Les variables en PHP

PHP n'est pas un langage fortement typé. Il décide à l'exécution, en fonction du contenu, quel type sera le plus indiqué pour une variable. Il est inutile de déclarer les variables au préalable. Leur nom doit être précédé du symbole $. Une variable n'ayant jamais été déclarée possède pour valeur la chaine vide : `""`.

Exemple 2.

```
<SCRIPT language="php">
$texte = 'Bonjour le monde';
$a = 3;
print "un texte: $texte\n";
print "un chiffre: $a";
</SCRIPT>
```

Cette portion de code renvoie sur la **sortie standard** :

```
un texte: Bonjour le monde
un chiffre : 3
```

Remarque 3. Généralement une chaîne de caractères est placée entre apostrophes. Dans l'exemple ci-dessus la chaîne suivant l'instruction `print` est encadrée par des guillemets. Consultez le §5 pour la différence d'interprétation de ces deux symboles.

Attention : Il faut insister sur le fait que le résultat de l'exécution du code précédent est envoyé sur la sortie standard. Le symbole "\n" permet d'avoir le résultat sur 2 lignes, ce qui ne signifie pas, qu'une fois interprété par le navigateur, le résultat sera encore sur 2 lignes, en effet le HTML ne reconnait pas le retour chariot. Si on veut vraiment obtenir le résultat sur 2 lignes dans le navigateur, il faut modifier le source PHP comme suit :

```
print "un texte: $texte<BR>";
```

Le script PHP ne fait que construire le code HTML envoyé à votre navigateur. On aboutit ainsi aux situations suivantes :

• Cas 1 :

```
<SCRIPT language="php">
$texte = 'Bonjour le monde';
$a = 3;
print "un texte: $texte\n";
print "un chiffre: $a";
</SCRIPT>
```

Code construit :

```
    un texte: Bonjour le monde
    un chiffre: 3
```

Résultat du code HTML *interprété :*

```
    un texte: Bonjour le monde un chiffre: 3
```

• Cas 2 :

```
<SCRIPT language="php">
$texte = 'Bonjour le monde';
$a = 3;
print "un texte: $texte<BR>";
print "un chiffre: $a";
</SCRIPT>
```

Code construit :

```
    un texte: Bonjour le monde<BR>un chiffre: 3
```

Résultat du code HTML *interprété :*

```
    un texte: Bonjour le monde
    un chiffre: 3
```

• Cas 3 :

```
<SCRIPT language="php">
$texte = 'Bonjour le monde';
$a = 3;
print "un texte: $texte<BR>\n";
print "un chiffre: $a";
</SCRIPT>
```

Code construit :

```
    un texte: Bonjour le monde<BR>
    un chiffre : 3
```

Résultat du code HTML *interprété :*

```
    un texte: Bonjour le monde
    un chiffre: 3
```

4. Déclaration de fonctions

L'instruction `function` permet de déclarer une fonction en la faisant suivre de son nom. On a ensuite, mise entre parenthèses, une liste d'arguments séparés par des virgules. Mais rien ne vous interdit de n'avoir aucun argument. Syntaxe :
function *nom_fonction(liste de paramètres)*

```
{
   instructions ;
}
```

Exemple 3.

```
function somme($a, $b)
{
  print $a+$b;
}
```

En Php, le principe des variables locales à une fonction est différent de celui des autres langages. Toute variable intervenant dans une fonction, et qui n'est pas un paramètre de cette dernière, est considérée comme locale même si elle a été déjà affectée à une valeur dans le programme principal.

Exemple 4.

```
<SCRIPT language="php">
function test()
{
  print $a;
}
$a = 3;
test();
</SCRIPT>
```

Le code ci-dessus ne produit aucun résultat car la variable `$a` de la fonction `test` n'a aucun lien avec la variable globale déclarée dans le programme principal. Elle est locale à la fonction et comme aucune valeur ne lui a été attribuée (dans la fonction), `$a` contient la chaine vide.

Pour pouvoir utiliser une variable globale dans une fonction il faut la déclarer en utilisant le mot clé `global`.

Exemple 5.

```
<SCRIPT language="php">
function test()
{
  global $a;

  print $a;
}
$a = 3;
test();
</SCRIPT>
```

5. Syntaxe du langage

Nous donnons ici un rapide résumé du langage. Consultez le manuel en ligne de Php pour plus d'informations.
. Les chaines de caractères doivent être placées entre apostrophes : `'Ceci est une chaîne'`. A l'intérieur de la chaîne le caractère $ perd sa signification, i.e. `$a` ne représente pas le contenu de la variable a. Pour que ce

caractère soit interprété, il suffit de placer la chaîne entre guillemets.
. Le caractère \ permet de despécialiser le caractère qui le suit.
. L'instruction `print` *expression* permet d'afficher sur la sortie standard l'expression qui peut être une variable, une expression arithmétique, une chaîne de caractères ou une expression composée de ces différents objets.
. La concaténation de chaînes de caractères est réalisée à l'aide de l'opérateur "." :

Exemple 6.

```
$heure = 10;
$minutes = 30;
$message = 'Il est exactement';
print $message;
print ' ';
print $heure;
print 'h';
print $minutes;
print 'min';
```

affiche sur la sortie standard

Il est exactement 10h30min

Le même résultat peut être de différentes façons :

```
print "$message $heure h $minutes min";
```

ou

```
print $message.' '.$heure.' h '.$minutes.' min';
```

. L'instruction `if` suit la même syntaxe que le C, la seule différence portant sur l'instruction `else if` qui doit être écrite en un seul mot : `elseif`.
. Les boucles `while, do .. while, for` suivent la même syntaxe que le C.
. L'instruction de sélection `switch` suit de même la syntaxe du C dont nous rappelons ici la structure :

```
switch($variable)
{
  case cas1:
     suite d'instructions;
     break;
   .
   .
   .
   case casN:
      suite d'instructions;
      break;
}
```

. PHP reconnaît tous les opérateurs arithmétiques, logiques et opérateurs de comparaison standards :
. +, −, /, ∗, %,
. &&, ||, , `xor`
. ==, <=, >=, <, >.

6. Tableaux et Tableaux associatifs

Il est inutile de déclarer un tableau en Php, toute variable de la forme `$nom[index]` est considérée comme un tableau et sa taille est gérée dynamiquement.

Exemple 7. Les instructions suivantes :

```
$t[0] = 'hello';
$t[1] = 3;
```

créé un tableau à 2 éléments dont le premier est une chaine de caractères et le second un entier.

On peut aussi directement remplir un tableau avec la fonction `array`. Par exemple pour le tableau ci-dessus on aurait pu écrire :

```
$t=array('hello', 3);
```

Par défaut l'index des éléments commence à 0 pour la fonction `array`. Vous pouvez modifier cela en utilisant l'opérateur => suivi de la valeur du premier index du tableau.

Exemple 8. Les instructions suivantes :

```
$t=array(1=>'hello', 3);
```

créé un tableau $\$t$ tel que $\$t[1]$ ='hello' et $\$t[2] = 3$. La fonction prédéfinie `count` de Php permet de connaître le nombre d'éléments dans un tableau.

Un tableau associatif est un tableau dont l'index n'est pas nécessairement de type numérique, ce peut être aussi une chaine de caractères.

Exemple 9.

```
$t['bleu'] = 'blue';
$t['rouge'] = 'red';
```

créé un tableau qui associe au nom d'une couleur sa traduction en anglais.

Remarque 10. On aurait pu aussi écrire

```
$t=array("bleu"=>"blue", "rouge"=>"red");
```

S'il est facile de parcourir les éléments d'un tableau indexé par des entiers en utilisant un simple compteur dans une boucle `for`, le problème est différent pour les tableaux associatifs. Php propose plusieurs méthodes pour réaliser ce travail, nous présentons ici celle qui nous semble la plus simple. Tout d'abord, il est utile de savoir que Php gère pour chaque tableau un pointeur interne permettant de se positionner sur chaque élément de ce dernier. Il existe en Php trois fonctions prédéfinies :

- `reset`, permet de placer le pointeur interne sur le premier élément du tableau,
- `key`, renvoie l'index de l'élément courant,
- `next`, déplace le pointeur sur le prochain élément.

Le parcours d'un tableau associatif peut alors s'effectuer de la façon suivante :

```
reset($t);
for ($i = 0; $i < count($t); $i++)
{
 $index = key($t);
 print $t[$index];
 next($t);
}
```

7. Interaction avec des formulaires Html

Nous ne traiterons ici que le cas des formulaires utilisants la méthode POST. La syntaxe générale d'un formulaire Html est la suivante :

```
<FORM METHOD=POST ACTION=script>
...
zones de saisies
...
<INPUT TYPE=SUBMIT VALUE=Valider>
</FORM>
```

Au moment de la validation du formulaire le script dont le nom est spécifié dans l'attribut `ACTION` est exécuté par le serveur. Le problème consiste à pouvoir récupérer dans le script toutes les données saisies par l'utilisateur dans le formulaire. Le langage Php gère avec une simplicité exemplaire cette interaction.
Lors de l'exécution du script, le tableau associatif `$_POST` est automatiquement créé. Pour chaque variable définie par l'attribut `NAME` dans les balises `INPUT` ou par l'attribut `OPTION` dans les balises `SELECT`, un index de même nom est créé dans ce tableau et la valeur associée est celle saisie dans le formulaire.

Exemple 11. On suppose que le fichier `exemple.html` contient le code suivant :

```
<FORM METHOD=POST ACTION=exemple.php>
Nom: <INPUT NAME=nom>
Pr&eacute;nom: <INPUT NAME=prenom>
<INPUT TYPE=SUBMIT VALUE=Validez>
</FORM>
```

Le script `exemple.php` contient les instructions suivantes :

```
<HTML>
<HEAD><TITLE> Affiche </TITLE></HEAD>
<BODY>
<SCRIPT language="php">
  print "Votre nom: <B>".$_POST['nom']."</B><BR>";
  print "Votre pr&eacute;nom: <I>".$_POST['prenom']."</I>";
</SCRIPT>
</BODY>
</HTML>
```

La validation du formulaire entraine l'affichage du nom (en gras) et du prénom (en italique) saisies dans le formulaire. Aux variables Html `nom` et

prenom ont été associé deux variables PHP $_POST['nom'] et $_POST['prenom'] qui contiennent les valeurs fournies par l'utilisateur.

REMARQUE IMPORTANTE 12. Lors de l'exécution du script, la variable $_POST est considérée comme une variable globale.

8. Le type HIDDEN

On peut très bien reproduire l'exemple précédent en utilisant qu'un seul fichier. Ce dernier sera un script PHP contenant 2 fonctions : l'une chargée d'afficher le formulaire et l'autre chargée d'afficher le contenu des variables $_POST['nom'] et $_POST['prenom']. On utilise alors une variable etat pour savoir si l'on doit :

- afficher le formulaire (première connexion au script, $etat = ""),
- afficher le contenu des variables (formulaire validé, $etat = "validation").

```
<HTML>
<HEAD><TITLE> Affiche </TITLE></HEAD>
<BODY>
<SCRIPT language="php">
function affiche($n, $p)
{
  print "Votre nom: <B>$n</B><BR>\n";
  print "Votre pr&eacute;nom: <I>$p</I>\n";
}
function formulaire()
{
  print "<FORM METHOD=POST ACTION=exemple.php3>\n";
  print "Nom: <INPUT NAME=nom>\n";
  print "Pr&eacute;nom: <INPUT NAME=prenom>\n";
  print "<INPUT TYPE=SUBMIT VALUE=Validez>\n";
  print "</FORM>";
}
switch($etat)
{
 case "":
   $etat = "validation";
   formulaire();
   break;
 case "validation":
   affiche($_POST['nom'], $_POST['prenom']);
   break;
}
</SCRIPT>
</BODY>
</HTML>
```

On supposera que ce code correspond au fichier exemple.php. Lorsque l'on effectue une requête à l'adresse http//xxx/exemple.php, le script exemple.php est exécuté par le serveur. La variable $etat n'ayant jamais été définie, elle a pour valeur la chaine vide, la fonction formulaire() s'exécute. Le serveur

renvoie sur l'entrée standard du navigateur le code Html du formulaire.
L'utilisateur remplit le formulaire et clique sur le bouton de validation ce qui a pour effet de demander (`ACTION=exemple.php`), l'exécution du script `exemple.php` par le serveur. Il faut **insister** sur le fait que `exemple.php` est **de nouveau** exécuté, ce qui signifie que la variable `$etat` n'a toujours pas de valeur même si elle a été positionnée à "validation" lors de la précédente exécution, il s'agit de deux exécutions bien distinctes. Le résultat sera donc une fois de plus l'affichage du formulaire au lieu du contenu des variables `$_POST['nom']` et `$_POST['prenom']`.
Il faut donc trouver un moyen permettant, au moment de la validation du formulaire, de transmettre au script la nouvelle valeur de `$etat`. On sait que lors d'une validation, toute variable Html associée à une zone de saisie devient une variable Php globale pour le script associée à l'attribut `ACTION`. Il suffit donc de créer dans le formulaire initial une variable Html nommée `etat`. C'est dans ce cadre qu'intervient le type `HIDDEN`, ce dernier permet de définir une zone de formulaire invisible à l'écran. Son but est donc essentiellement de permettre de définir dans un formulaire des variables Html supplémentaires non reliées à des données saisies par l'utilisateur. La syntaxe générale est :
`<INPUT TYPE=HIDDEN NAME=nom_variable VALUE=valeur_variable>`
Le code de `exemple.php` est donc le suivant (notez au passage l'utilisation de la fonction `isset` qui renvoie VRAI si une variable existe, FAUX sinon) :

```
<HTML>
<HEAD><TITLE> Affiche </TITLE></HEAD>
<BODY>
<SCRIPT language="php">
function affiche($n, $p)
{
  print "Votre nom: <B>$n</B><BR>";
  print "Votre pr&eacute;nom: <I>$p</I>";
}
function formulaire()
{
  print "<FORM METHOD=POST ACTION=exemple.php>\n";
  print "Nom: <INPUT NAME=nom>\n";
  print "Pr&eacute;nom: <INPUT NAME=prenom>\n";
/*  PASSAGE DE LA VALEUR DE LA VARIABLE ETAT  */
/*                                            */
  print "<INPUT TYPE=HIDDEN NAME=etat VALUE=validation>\n";
/*                                            */
  print "<INPUT TYPE=SUBMIT VALUE=Validez>\n";
  print "</FORM>";
}
if (isset($_POST['etat']))
   affiche($_POST['nom'],$_POST['prenom']);
else
   formulaire();
}
```

```
</SCRIPT>
</BODY>
</HTML>
```

9. Les Cookies

Il existe une autre méthode (un peu plus souple) permettant à un script de générer des valeurs utilisables par d'autres scripts : les cookies. Un cookie est une information stockée par le serveur Web sur le poste client. Il est évident que ceci peut poser des problèmes de sécurité. Ainsi pour que cette méthode fonctionne, il est indispensable de configurer votre navigateur afin que ce dernier accepte le stockage des cookies. Pour "poser" un cookie sur un poste client vous disposez de l'instruction :

`setcookie(`"*nom_cookie*"`,` *valeur_du_cookie*`)`

ATTENTION :

- les cookies doivent être envoyés avant toute instruction d'affichage ou code HTML présent dans le fichier (ceci est une restriction des cookies, pas de PHP). Ceci impose en particulier que l'appel de cette fonction doit être fait avant les balises <`html`> ou <`head`>.
- les cookies ne sont pas accessibles par le script en cours d'exécution, ils le seront par contre pour tout script exécuté après celui qui les a déposés. Pour chaque script qui s'exécutera le serveur ira automatiquement récupérer chez le client les cookies déposés et les placera dans le tableau associatif `$_COOKIE`. La valeur du cookie de nom `nom_cookie` sera donc accessible via `$_COOKIE['nom_cookie']`.

Pour effacer un cookie, il faut lui affecter une date d'expiration antérieure à l'heure actuelle afin que le navigateur déclenche son mécanisme de destruction automatique des cookies. Ceci se réalise en Php via l'instruction `setcookie("nom cookie","",time()-3600)` qui indique que le cookie doit expirer une heure avant l'heure actuelle, autrement dit qu'il n'est plus valide depuis une heure.

En prenant en compte toutes ces contraintes, voilà comment écrire le script précédent en utilisant un cookie pour stocker la valeur de `état`. Vous remarquerez cette fois-ci que les entêtes HTML ont été incluses dans le code des fonctions.

```
<SCRIPT language="php">
function affiche($n, $p)
{
  setcookie("etat", "", time() - 3600);
  print "<HTML><HEAD><TITLE> Affichage </TITLE></HEAD>\n";
  print "<BODY>\n";
  print "Votre nom: <B>$n</B><BR>";
  print "Votre pr&eacute;nom: <I>$p</I>";
  print "</BODY></HTML>\n";
}
function formulaire()
{
  setcookie("etat","validation");
```

```
  print "<HTML><HEAD><TITLE> Affichage </TITLE></HEAD>\n";
  print "<BODY>\n";
  print "<FORM METHOD=POST ACTION=exemple.php>\n";
  print "Nom: <INPUT NAME=nom>\n";
  print "Pr&eacute;nom: <INPUT NAME=prenom>\n";
  print "<INPUT TYPE=SUBMIT VALUE=Validez>\n";
  print "</FORM>";
  print "</BODY></HTML>\n";
}
if (isset($_COOKIE['etat']))
   affiche($_POST['nom'], $_POST['prenom']);
else
   formulaire();
</SCRIPT>
```

Avez-vous compris à quoi sert la première instruction de la fonction `affiche` ?

10. Les variables de session

Le nombre d'informations que l'on peut stocker dans un cookie est limité. Il existe une autre méthode pour conserver des informations lorsqu'un utilisateur visite un ensemble de pages : les variables de session. Contrairement aux cookies, ces dernières sont stockées sur le serveur et elles sont accessibles via le tableau associatif `$_SESSION`. Le principe est le suivant :

- Pour chaque page où l'on souhaite utiliser les variables de session, il faut faire appel à la fonction `session_start()` et ceci AVANT d'écrire le moindre code HTML.
- Pour créer une variable de session, il suffit d'écrire dans votre script `$_SESSION['nom var']=valeur`.
- Cette variable ainsi créée sera disponible pour tout autre script Php (via `$_SESSION['nom var']`) dès lors que ce dernier débute par `session_start()`.
- Pour détruire une session, il faut appeler la fonction `session_destroy()`. Cette fonction est automatiquement appelée, si le visiteur ne charge plus de page au bout d'un certain timeout.

Reprenons l'exemple précédent avec ce principe.

```
<SCRIPT language="php">
function affiche($n, $p)
{
  session_destroy();
  print "<HTML><HEAD><TITLE> Affichage </TITLE></HEAD>\n";
  print "<BODY>\n";
  print "Votre nom: <B>$n</B><BR>";
  print "Votre pr&eacute;nom: <I>$p</I>";
  print "</BODY></HTML>\n";
}
function formulaire()
{
  $_SESSION['etat']='validation';
  print "<HTML><HEAD><TITLE> Affichage </TITLE></HEAD>\n";
  print "<BODY>\n";
```

```
  print "<FORM METHOD=POST ACTION=exemple.php>\n";
  print "Nom: <INPUT NAME=nom>\n";
  print "Pr&eacute;nom: <INPUT NAME=prenom>\n";
  print "<INPUT TYPE=SUBMIT VALUE=Validez>\n";
  print "</FORM>";
  print "</BODY></HTML>\n";
}
session_start();
if (isset($_SESSION['etat']))
   affiche($_POST['nom'], $_POST['prenom']);
else
   formulaire();
</SCRIPT>
```

11. Cas particulier des sélections multiples

En Html les balises suivantes :
- <INPUT TYPE=CHECKBOX ...>
- <SELECT NAME=... MULTIPLE>

permettent d'effectuer plusieurs choix qui seront affectés à une même variable Html.

Exemple 13.

```
<FORM METHOD=POST ACTION=...>
Composez votre pizza:
<INPUT TYPE=CHECKBOX NAME=pizza VALUE="Roquefort"> Roquefort <BR>
<INPUT TYPE=CHECKBOX NAME=pizza VALUE="Champignons"> Champignons <BR>
<INPUT TYPE=CHECKBOX NAME=pizza VALUE="Jambon"> Jambon <BR>
<INPUT TYPE=CHECKBOX NAME=pizza VALUE="Anchois"> Anchois <BR>
<INPUT TYPE=CHECKBOX NAME=pizza VALUE="Olives"> Olives <BR>
<INPUT TYPE=SUBMIT ACTION=Validez>
</FORM>
```

A la validation du formulaire le choix `pizza` peut contenir (par exemple) uniquement "Roquefort" ou bien "Roquefort", "Jambon" et "Champignons". Pour pouvoir accéder à tous les choix, il faut déclarer la variable Html comme suit :
<INPUT TYPE=CHECKBOX NAME=pizza[] VALUE=...>
La variable Php correspondante `$_POST['pizza']` sera alors un tableau dont chaque élément sera un des choix sélectionnés par l'utilisateur.

Exemple 14. Si l'utilisateur choisit "Roquefort" et "Champignons" alors `$_POST['pizza'][0]` contiendra "Roquefort" et `$_POST['pizza'][1]` contiendra "Champignons".

L'instruction `Count($_POST['pizza'])` permet de savoir combien d'éléments contient le tableau `$_POST['pizza']`.

12. Passage de valeurs via un lien hypertexte

Jusqu'à présent, nous avons vu comment passer des valeurs à un script Php via un formulaire. Il est possible de passer à un script Php un ensemble

de variables et leurs valeurs via un lien hypertexte. Pour celà, il suffit d'accoler à l'adresse du script Php, la ligne suivante :
`?nom_var1=valeur1&nom_var2=valeur2&...&nom_varN=valeurN`

Exemple 15. Supposons que dans un document Html on ait le lien hypertexte suivant :
`<A HREF="http://.../essai.php?etat=module&code=3">Cliquez ici</A>`
lorsque l'utilisateur cliquera sur le lien ceci déclenchera l'exécution du script `essai.php` qui créera automatiquement un tableau associatif `$_GET` tel que : `$_GET['etat']=module` et `$_GET['code']=3`.

13. Interface avec les bases de données

13.1. **MySQL.** Vous trouverez en annexe une liste des différentes fonctions permettant d'interroger une base de données MySQL via un script Php. La démarche à suivre est "essentiellement" toujours la même :

(1) Connection au serveur :
```
. mysql_connect("serveur", "username", "password");
```
(2) Sélection d'une base :
```
. mysql_select_db("nom de la base");
```
(3) Requête SQL :
```
. $result = mysql_query("select * from table");
```

`$result` contient alors un ensemble de lignes qui correspondent au résultat de la requête. On accède à chacune de ces lignes par appel successif à la fonction `mysql_fetch_row` :
`$row = mysql_fetch_row($result)`
`$row` reçoit un tableau qui contient chacun des champs (sélectionnés par la commande `select`) de la ligne résultat. La fonction renvoie 0 lorsque toutes les lignes de `$result` ont été parcourues.
Pour obtenir l'intitulé des champs sélectionnés par `select` on dispose de la fonction `mysql_fetch_field` :
`$field = mysql_fetch_field($result)`
Chaque appel à cette fonction permet d'obtenir différents renseignements sur un des champs résultants de la requête (par ordre de sélection) tels que :
- son nom (`$field->name`),
- le nom de la table auquel appartient le champ (`$field->table`),
- le type du champ (`$field->type`),

etc ...
`mysql_fetch_field` renvoie 0 lorsque tous les champs ont été parcourus. `mysql_num_fields($result)` contient le nombre de champs concernés par la requête. Une fois que l'on a terminé les différents traitements nécessaires, il ne faut pas oublier de libérer la mémoire allouée pour stocker le résultat de la requête et de clore l'ouverture de la base :

(4) Libération de la mémoire :
```
. mysql_free_result($result);
```
(5) Déconnexion :
```
. mysql_close();
```

Exemple 16. Supposons que l'on ait une base *contact* contenant une table *adresse* dont la structure est :
- un champ `nom` : chaine de caractères,
- un champ `adresse` : chaine de caractères,
- un champ `code_postal` : variable entière.

et qui contient :

Nom	**Adresse**	**Code_postal**
Mulder	Los Angeles	0001
Scully	Los Angeles	0002
Bond	Londres	0007

• Connexion à la base :
. `mysql_connect("serveur", "username", "password");`
. `mysql_select_db("contact");`

• Requête SQL :
. `$result = mysql_query("select * from adresse");`

• Affichage des intitulés de champs *(nom, adresse, code_postal)*

```
while ($field = mysql_fetch_field($result))
  print "$field->name";
```

• Affichage du résultat de la requête

```
while ($row = mysql_fetch_row($result))
  for ($i = 0; $i < mysql_num_fields($result); $i++)
     print "$row[$i]";
```

A partir de cette structure générale, il est assez facile de développer un script PHP affichant dans un tableau HTML le résultat d'une requête de type `select * ....`

```
<HTML>
<HEAD><TITLE> Resultat requete </TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="php">
    mysql_connect("serveur","username","password");
    mysql_select_db("nom_base");
    $result = mysql_query("select * from nom_table");
    print '<TABLE BORDER=1>\n';
    print '<TR>';
    while ($field = mysql_fetch_field($result))
            print "<TH> $field->name";
    while ($row = mysql_fetch_row($result))
          {
            print '<TR>';
            for ($i = 0; $i < mysql_num_fields($result); $i++)
                 print "<TD> $row[$i]";
```

```
          }
     print '</TABLE>';
     mysql_free_result($result);
     mysql_close();
</SCRIPT>
</BODY>
</HTML>
```

13.2. **PostgreSQL.** Sans entrer dans les détails, voici le même script d'interrogation lorsque la base est gérée par PostgreSQL (on considère ici qu'il n'y a pas de mot de passe sur la base).

```
<HTML>
<HEAD><TITLE> Resultat requete </TITLE></HEAD>
<BODY>
<SCRIPT LANGUAGE="php">
    $db=pg_connect("user=XXXX host=XXXX dbname=XXXX");
    $result=pg_exec($db, "select * from nom_table");
    $nbchamps=pg_numfields($result);
    print '<TABLE BORDER=1>\n';
    print '<TR>';
    for($j=0;$j<$nbchamps;$j++)
       print "<TH>".pg_fieldname($result, $j);
    $nblignes=pg_numrows($result);
    for($i=0; $i<$nblignes; $i++)
          {
            print '<TR>';
            $row=pg_fetch_row($result,$i);
            for ($j = 0; $j < $nbchamps; $j++)
                 print "<TD> $row[$j]";
          }
     print '</TABLE>';
     pg_freeresult($result);
     pg_close($db);
</SCRIPT>
</BODY>
</HTML>
```