Ce document est inspiré de http://etudiant.univ-mlv.fr/~ajouinea/master/BDD/memoPhp.pdf (qui n'est d'ailleurs plus disponible).

Memo pour Php

Patrick TRAU – Unistra – faculté de physique et ingénierie

1) Les bases du Php :

Le Php est un langage de programmation ressemblant au C (du moins, pour les fonctionnalités de base du C). Il s'exécute sur un serveur web (on dit "côté serveur"). Il faut donc que vous disposiez d'un serveur web (ou en installiez un sur votre ordinateur, par exemple LAMP) contrairement au Javascript (qui lui tourne côté client et ne nécéssite qu'un navigateur). On le code à l'intérieur d'une page html, ce qui permet d'utiliser toutes les possibilités graphiques de l'HTML (c'est quand même mieux que "printf"). Il est puissant : on y trouve beaucoup de ce qu'on peut trouver dans les autres langages, en particulier en C (même sans inclure les divers .h). Gratuit (OpenSource), puissant, bien documenté, souple (quelquefois, je trouve que c'est un inconvénient, s'il était plus strict il détecterait plus facilement certaines erreurs), bref beaucoup de qualité qui l'ont rendu célèbre et populaire.

Il possède aussi une énorme bibliothèque de fonctions qui allège énormément la tâche des développeurs, et rend les scripts opérationnels de suite : quasiment tout ce qu'on trouve dans les bibliothèques standard du C, des fonctions d'accès à toutes les bases de données, les accès réseau de haut niveau (IMAP, POP, IRC, LDAP...) et bas niveau (sockets), et même générer du graphique (GTK, flash...).

1.1 L'incorporation de code dans une page

Un fichier source Php peut comporter des parties HTML. Chaque bout de code Php doit être entouré par les tags "<?php" et "?>" (ou "<?" et "?>" ou encore <script language="php"> et </script>)

```
codes HTML
<?php
        Bout de code Php;
?>
codes HTML
```

Le serveur Web enverra les codes HTML tels quels, dans l'ordre où il les trouve. Par contre, dès qu'il est dans du code php, il n'envoie plus rien mais exécute les instructions qu'on lui donne (elles aussi, dans l'ordre). Si l'on veut que quelque chose soit également envoyé au client dans une partie php, il faut utiliser l'instruction "echo" (ou print ou printf, plus puissants mais plus complexes). Tout ce qui est envoyé au client doit respecter le html.

Php est un langage qui semble facile car il ne possède pas toutes les contraintes de langages plus évolués comme le Java ou le C++, même s'il s'en inspire beaucoup. Il est très permissif, mais ceci nécessite de la rigueur et de l'expérience de la part du programmeur.

Un script Php a pour extention .php (ou .inc, fortement déconseillé car le code qui y est stocké n'est pas interprété par un serveur web). Son nom doit correspondre aux règles habituelles (si possible sans espaces, accents, plutôt en minuscules).

Tous les scripts PHP sont une suite d'instructions. Une instruction peut être une assignation (sous la forme où=combien;), un appel de fonction, une structure de contrôle (boucle ou instruction conditionnelle) ou à la rigueur une instruction vide (;). Une instruction se termine par un point virgule ";". De plus, plusieurs instructions peuvent être regroupées en bloc, délimité par des accolades ("{}"). Un bloc est considéré comme une instruction. Si le bloc contient des instructions, elles seront terminées par un ";", y compris

celle avant "}". Par contre il n'y en aura pas derrière l'accolade. Les différents types d'instruction seront décrits plus loin.

1.2 Les variables

Une variable est une mémoire à laquelle on donne un nom. On ne déclare pas les variables en php, on les distingue par le fait que le nom commence toujours par "$" (et contient chiffres, lettres et "_") :

```
$ma_variable
```

On peut aussi définir une constante par la fonction define(nom_constante,valeur_constante); Le nom de la constante ne commence **pas** par "$".

Une variable en Php n'est typée que par sa valeur. Bien que ce ne soit pas obligatoire, je vous conseille d'assigner une valeur (d'un certain type) lors de sa première apparition, et de ne pas changer son type tout au long du programme. En php, il n'y a que 4 types 'scalaires' : booléens, entiers, flottants et chaîne de caractères. Il y a également deux types complexes : tableaux et objets

1.3 Les tableaux

Un tableau se définit comme une variable. On peut utiliser la fonction array() pour l'initialiser :

```
$tab=array(cle1=>val1,...,clen=>valn);
```

Une clé est un entier positif ou une chaîne de caractères, val peut être n'importe quoi. Si on ne précise pas de clef, php utilise un entier, en commençant par 0.

Pour accéder à une valeur on utilise la syntaxe suivante :

```
echo $tab[cle1]; // renvoie val1
```

La taille d'un tableau s'obtient par la fonction sizeof(array), exemples :

```
$mon_tableau=array(1=>1,2=>2,3=>3);
echo sizeof($mon_tableau); //renvoie 3
$mon_tableau[4]=4;
echo sizeof($mon_tableau); //renvoie 4
$mon_tableau[]=5; //ici on ajoute en bout de tableau la valeur 5
echo sizeof($mon_tableau); //renvoie 5
```

1.4 Les variables prédéfinies

On appelle variable d'environnement les variables qui dépendent du serveur. Elles sont utiles pour récupérer des informations telles que les variables postées dans un formulaire, les variables de session... Ainsi on récupère les valeurs d'un formulaire par la variable d'environnement $_POST ou $_GET (suivant la méthode). C'est en fait un tableau associatif (défini avec des clés) qui contient toutes les valeurs postées. Voici ci après un exemple. Soit le formulaire HTML suivant (à droite : le résultat)

```
<html>
   <form method=post action=mon_script.php>
        <input type=text name=nom value=etudiant><br>
        <input type=password name=pass value=password><br>
        <input type=submit name=submit value=Ok>
   </form>
</html>
```

Ce formulaire, une fois validé, est traité par mon_script.php :

```
<?php
echo $_POST['nom']. " <br> ".$_POST['pass'];
?>
```

mon_script.php retourne le résultat suivant :

```
etudiant
password
```

La methode POST est plus sure. La methode GET quand à elle permet de mémoriser les arguments dans une URL. On n'est pas obligé de re-remplir le formulaire, par exemple par l'URL :

```
http://mon_script.php?nom="etudiant"&pass="password"
```

Variables prédéfinies dans php :

$GLOBALS	Variables globales (utile dans les fonctions et entre fichiers, mais aussi plus accessibles aux pirates)
$_SERVER	Variables relatives au serveur
$_GET / $_POST	Variables retournées par les formulaires
$_COOKIE	Variables contenues dans les cookies
$_FILE	Variables relatives aux données d'un fichier suite à son téléchargement
$_ENV	Variables d'environnement
$_SESSION	Variables de sessions

1.5 echo

Pour envoyer du texte au client html, on utilise l'instruction echo ;

```
echo "texte et codes html";
```

Si le texte contient un nom de variable (commençant par $), celle-ci est remplacée par sa valeur. Elle doit alors être encadrée par des espaces ou des accollades (par ex "il est {$h}h"). Si l'on veut envoyer des objets différents on peut les concaténer par "." :

```
echo "<p>bonjour".$nom.", il est ".$h."h</p>";
```

Si on veut envoyer un caractère spécial, ont l'échape en le précédant par \ : \", \$, \\,

printf s'utilise comme en C, et permet des écritures évoluées (voir un cours C pour plus de détails).

Je vous rapelle qu'en fin de compte, ce qui est envoyé du serveur au client doit respecter le language de description (HTML). Envoyer un \n ne change que le code source, aller à la ligne doit se faire par <p> ou
.

2) Les structures de contrôle :

2.1 Instruction IF ... ELSEIF ... ELSE

L'instruction if permet l'exécution conditionnelle d'une partie de code et suit le schéma suivant :

```
<?php
if (expression)
  {
      commmandes executées uniquement si l'expression est vraie
  }
?>
```

On peut décomposer en plusieurs cas grâce à elseif : un (et un seul) if, puis autant de elseif que nécessaire, puis un (ou aucun) else (il y a toujours une condition entre partenthèses derrière if et elseif, mais jamais derrière le dernier else) Si aucun test précédent n'est vrai alors c'est le code derrière le else (en dernier, toujours sans condition) qui est exécuté, comme dans l'exemple :

```
if ($a > $b) {
     echo "a est plus grand que b";
}
elseif ($a == $b) {      //attention, deux signes "=" dans un test
     echo "a est égal à b";
}
else {
     echo "a est plus petit que b";
}
```

remarque sur les conditions : les opérateurs de test sont <, >, <=, >=, != (différent), = = (égal). Attention, un seul = est une affectation ! if($a=0) met 0 dans $a (et répond "faux" car 0=faux). On peut combiner des conditions par && (et), || (ou), ! (complément).

2.2 Instruction WHILE

L'instruction while permet la répétition d'instructions :

```
<?php
while (expression)  //surtout pas de ";", sinon il répète l'instruction vide ;
 {
     commandes executées tant que l'expression est vraie,
     peut-être même jamais !

 }
?>
```

Exemple :

```
$i = 1;
while ($i <= 10000) {
    $i=$i*2;
    echo " $i"; /* affiche toutes les puissances de 2 inférieures à 10000 */
}
```

Si l'expression est fausse dès le début, on n'exécute pas du tout les instructions dans le bloc. Pour certains cas, il est nécessaire d'imposer de toujours faire au moins une fois la boucle. On utilise alors l'écriture suivante :

```
<?php
do
 {
     commandes executées tant que l'expression est vraie,
     mais quoi qu'il arrive au moins une fois !
 }
while (expression); //notez le ";"
?>
```

2.3 Instruction FOR

L'instruction for permet également de répéter des instructions (principalement pour un nombre de fois connu) et se construit comme suit :

```
for (expr1; expr2; expr3)
  {
     instructions executées plusieurs fois;
  }
```

La première expression (expr1) est évaluée (exécutée), quoi qu'il arrive au début de la boucle. Au début de chaque itération, l'expression expr2 est évaluée. Si l'évaluation est vraie, la boucle continue et l'instruction est exécutée. Si l'évaluation est fausse, l'exécution de la boucle s'arrête. A la fin de chaque itération, l'expression expr3 est évaluée (exécutée).

exemple : dire trois fois bravo :

```
for ($i=0;$i<3;$i++)
  {
      echo "bravo ! ";
  }
```

2.4 Instruction FOREACH

L'instruction foreach est un moyen simple de passer en revue un tableau :

```
foreach(array_expression as $value)
  {
      instructions
  }
```

Cette instruction passe en revue le tableau array_expression. A chaque itération, la valeur de l'élément courant est assignée à $value et le pointeur interne de tableau est avancé d'un élément (ce qui fait qu'à la prochaine itération, on accédera à l'élément suivant).

exemple, si vous avez appelé votre page par *ma-page.php?a=18&b=25* :

```
foreach($_GET as $valeur) echo $valeur.'<br/>';
```

affichera les valeurs transmises par GET. Et même mieux :

```
foreach($_GET as $index=>$valeur) echo $index.' : '.$valeur.'<br/>';
```

3) Les fonctions :

Une fonction php se définit par la syntaxe suivante :

```
function ma_fonction($paramètre1,$paramètre2...,$paramètren)
  {
      instructions de la fonction;
  }
```

Les fonctions permettent de regrouper sous un nom une suite d'instructions. Ceci permet de structurer son programme et donc décomposer une description de tâche en plusieurs sous-tâches. Elle peuvent recevoir des arguments (aucun, un ou plusieurs), et retourner au maximum une valeur (mais celle-ci peut être un tableau, donc regrouper plusieurs variables simples). Les variables définies dans une fonction sont locales, c'est à dire utilisables uniquement dans cette fonction. La seule manière d'envoyer une valeur dans une fonction est de la donner en argument (sauf variables globales, définies par le mot clef **global**). Quelques exemples pour comprendre le fonctionnement des fonctions :

```
//Exemple d'une fonction qui ne retourne rien, ne prend pas de
// paramètres (mais fait quand même quelque chose : affiche un message).
function afficheString()
  {
              echo " bonjour tout le monde ";
  }
afficheString(); //affichage du message, pas d'argument ni retour

//Exemple d'une fonction qui calcule la somme de 2 paramètres et
// affiche le résultat (mais ne retourne rien)
function add($x,$y)
  {
              echo $x+$y;
  }
add(1,2); //affiche 3
```

```
//Cet exemple remplit la même fonctionnalité que la fonction add() mais
// au lieu d'afficher le résultat, la fonction retourne la somme.
function add2($x,$y)
  {
              return $x+$y;
  }
echo add2(1,2); //affiche 3
```

ATTENTION : si la fonction veut modifier un de ses arguments, la modification n'est faite que localement dans la fonction, mais pas dans le paramètre donné à l'appel. C'est un "passage par valeur". Mais le passage par référence est également autorisé :

```
function incremente(&$x) { $x++; }

$a=3;
incremente($a);
echo $a; //affichera 4;
```

seule différence, le "&" ajouté dans l'entête de la fonction (aucun changement dans la fonction ni à l'appel (contrairement au passage par adresse du C).

3.2 organisation

Je conseille de mettre toutes les fonctions dans le <head> du fichier. En effet, déclarer les fonctions n'a pas d'effet immédiat sur la page, ce n'est que lorqu'on les apelle (en transmettant des arguments si nécessaire) que le serveur les exécute. On mettra donc les appels plutôt dans le <body>. Si des fonctions doivent être définies dans plusieurs pages, le plus simple est de les déclarer dans un fichier séparé (par ex fonctions.php) et de les inclure dans chaque page par un simple ligne :

```
include("fonctions.php");
```

Une autre utilisation permet de définir, une fois pour toutes, une partie de page dans un fichier séparé. Si plusieurs pages l'incluent, elles auront toutes la même partie. Une seule modification du fichier inclus modifiera toutes les pages qui s'en servent.

4) Références :

Ce document est fortement inspiré de http://etudiant.univ-mlv.fr/~ajouinea/master/BDD/memoPhp.pdf. C'est parce qu'il n'était plus disponible que je l'ai recopié. Si son propriétaire initial pouvait me contacter, j'en serai ravi.

Liens vers des cours PHP et SQL :

- http://www.phpsources.org/tutoriels-PHP.htm
- http://www.phpfrance.com
- http://www.php.net

P. TRAU 2017
