

Scripting with Php

Ivan Kurzweg

11 février 2008

Scripting with Php
by Ivan Kurzweg

Copyright © 2008 Ivan KURZWEG, *ik-r@wanadoo.fr*

Permission to use, copy, modify, and distribute this documentation for any purpose with or without fee is here by granted, provided that the above copyright notice and this permission notice appear in all copies.

Table des matières

1	Notions de bases	3
1.1	Présentation de Php	3
1.2	L'interpréteur PHP	3
1.3	Instructions, expressions, commentaires,	4
1.4	Le fameux hello world	4
1.5	Types de données	5
1.6	Variables	6
1.7	Opérateurs	7
1.7.1	Opérateurs arithmétiques	7
1.7.2	Opérateurs de comparaison	7
1.7.3	Opérateurs logiques	7
1.7.4	Opérateurs de transtypage	7
1.8	Tableaux	8
1.8.1	Tableau à une dimension	8
1.8.2	Tableau à plusieurs dimension	8
1.8.3	Fonctions sur les tableaux	8
1.8.4	Parcours d'un tableau	9
1.9	Garbage collector	10
1.10	Scripts interactifs	10
1.11	Exercices d'applications	11
1.11.1	Analyse de code	11
1.11.2	Instructions simples	11
2	Structures de contrôles	12
2.1	Conditionnelles	12
2.1.1	if	12
2.1.2	Le Switch	13
2.2	Répétitives	13
2.2.1	La boucle for	13
2.2.2	La boucle while	14
2.2.3	La boucle do ... while	14
2.2.4	La boucle foreach	15
2.2.5	Sortie d'une boucle	16
2.3	Exercices d'applications	17
2.3.1	Conditionnelles	17
2.3.2	Boucles	17
2.3.3	Tableaux	18
2.3.4	Carré magique	18
3	Fonctions	18
3.1	Appel d'une fonction	18
3.2	Définition d'une fonction	19
3.3	Portée des variables	20
3.3.1	Variables globales	20
3.3.2	Variables statiques	21
3.4	Paramètres	21
3.4.1	Passage par valeur	21
3.4.2	Passage par adresse	21
3.5	Récurtivité	22
3.6	Exercices d'applications	23
3.6.1	Fonctions	23
3.6.2	Récurtivité	23
4	Fichiers	23
5	Objets	23

6	Bases de données	23
7	Références	23
7.1	Références	23

Liste des tableaux

1	Opérateurs arithmétiques	7
2	Opérateurs de comparaison	7
3	Opérateurs logiques	7
4	Opérateurs logiques	8
5	Fonctions concernant les tableaux	9
6	Carré magique	18

Résumé

Ce document présente une introduction à la programmation PHP, mais en se focalisant sur un usage interactif plus que sur une vision orientée "Web" du développement en PHP. Les exemples de ce cours sont donnés sous Linux, mais devraient pouvoir, sans trop de soucis, être adaptés à un environnement Windows.

1 Notions de bases

1.1 Présentation de Php

Php pour *PHP:Hypertext Preprocessor* est un langage de script, écrit en PERL, spécialement conçu pour permettre la création de pages Web dynamiques. Néanmoins, Php permet de s'initier à la programmation fonctionnelle, et nous nous intéresserons dans ce cours plus particulièrement à son utilisation en ligne de commande.

Php fait donc partie de la famille des langages de programmation *interprété*, contrairement à d'autres langages dits *compilés* (ADA, C, C++,...). La différence de ces deux grandes familles est la manière dont le compilateur traduit une fois pour toute le code source en un *fichier indépendant exécutable* (donc utilisant du code machine ou du code d'assemblage), alors que l'interpréteur est nécessaire à *chaque lancement* du programme interprété, pour traduire au fur et à mesure le code source en code machine. Cette traduction à la volée a pour conséquence la relative lenteur des programmes interprétés par rapport aux programmes compilés, malgré les progrès des processeurs et des interpréteurs. Néanmoins, la facilité d'évolution d'un programme interprété et la rapidité de mise en place de petits programmes (généralement destinés au Web) pèsent dans la balance en faveur des langages interprétés.

Il est à noter qu'une troisième famille de langage peut-être placée entre les deux précédentes, il s'agit des langages permettant une première compilation en langage intermédiaire, l'exécution se faisant au sein d'une machine virtuelle sur le système cible. L'exemple le plus connu est ainsi JAVA™.

1.2 L'interpréteur PHP

Sous Linux, l'interpréteur PHP est installé via le paquet PHP-CLI. Une fois ce paquet installé, la commande **php** est disponible. Sa localisation dans le filesystem peut s'obtenir grâce à la commande :

```
ikare@ix:~$ which php
/usr/bin/php
```

L'appel de l'interpréteur Php va permettre bien sûr d'exécuter un script Php, mais la commande dispose de nombreuses options :

```
ikare@ix:~$ /usr/bin/php -h
Usage: php [options] [-f] <file> [--] [args...]
       php [options] -r <code> [--] [args...]
       php [options] [-B <begin_code>] -R <code> [-E <end_code>] [--] [args...]
       php [options] [-B <begin_code>] -F <file> [-E <end_code>] [--] [args...]
       php [options] -- [args...]
       php [options] -a
```

```

-a          Run interactively
-c <path>|<file> Look for php.ini file in this directory
-n          No php.ini file will be used
-d foo[=bar] Define INI entry foo with value 'bar'
-e          Generate extended information for debugger/profiler
-f <file>   Parse and execute <file>.
-h          This help
-i          PHP information
-l          Syntax check only (lint)
-m          Show compiled in modules
-r <code>   Run PHP <code> without using script tags <?...?>
-B <begin_code> Run PHP <begin_code> before processing input lines
-R <code>   Run PHP <code> for every input line
-F <file>   Parse and execute <file> for every input line
-E <end_code> Run PHP <end_code> after processing all input lines
-H          Hide any passed arguments from external tools.
-s          Display colour syntax highlighted source.
-v          Version number
-w          Display source with stripped comments and whitespace.
-z <file>   Load Zend extension <file>.

args...    Arguments passed to script. Use -- args when first argument
           starts with - or script is read from stdin

--ini      Show configuration file names

--rf <name> Show information about function <name>.
--rc <name> Show information about class <name>.
--re <name> Show information about extension <name>.
--ri <name> Show configuration for extension <name>.

```

1.3 Instructions, expressions, commentaires, ...

Un programme / script PHP est une suite d'instructions exécutées séquentiellement par l'interpréteur. Chaque instruction est séparé de la suivante par un *point virgule*. Une instruction peut-être considérée comme du code PHP dont l'exécution va modifier l'état de la machine.

Certaines instructions peuvent contenir des *expressions*. Les instructions sont exécutés, les expressions sont évaluées.

Les instructions peuvent être regroupées dans des *blocs*, délimités par { }. Un bloc garanti que toutes les instructions sont exécutées de manière séquentielles de la première à la dernière.

Le langage PHP est dit sensible à la casse (il fait la différence entre les majuscules et les minuscules) pour les variables, objets ou fonctions définies par le programmeur. A l'inverse, il n'est pas sensible à la casse pour l'ensemble des mots-clefs et des primitives du langage.

les commentaires ne sont pas pris en compte par l'interpréteur, mais il convient de prendre la bonne habitude de commenter les codes que vous produisez. Les caractères encadrant des commentaires sur plusieurs lignes sont /* */ ou // permettant de commenter le reste d'une seule ligne.

Enfin, la présence ou non de retours chariots, tabulations ou espaces n'a pas d'influence sur le comportement du programme.

1.4 Le fameux hello world

Commençons donc par l'indémorable "bonjour au monde", programme classique. Voici le code source du programme :

Exemple 1.1 Hello world

```
#!/usr/bin/php
<?
    echo "Hello World \n";
?>
```

L'exécution du programme se fait soit par l'appel de la commande `php` en passant le nom du fichier en paramètre, soit sous Linux en modifiant les droits du fichier :

```
ikare@ix $ ls
hello.php
ikare@ix $ php hello.php
Hello World
ikare@ix $ chmod u+x hello.php
ikare@ix $ ./hello.php
Hello World
```

Le changement de permission sur le fichier (**chmod**) permet de rendre le script exécutable via le lancement de l'interpréteur indiqué dans la première ligne du fichier source (*shebang*).

1.5 Types de données

Php utilise les types de données scalaires classiques :

- *Entiers* : la plage des nombres entiers disponibles dépend de la plate forme d'exécution, mais est généralement comprise entre -2 147 483 648 et +2 147 483 647. Les entiers peuvent être représentés sous leur forme décimale, octale (commençant par un 0) ou hexadécimale (commençant par 0x), précédée de leur signe :

```
//des entiers décimaux
1998
-641
+33

//des entiers octaux
0755 // valeur decimale 493
+010 // valeur decimale 8

//des entiers hexadécimaux
0xFF // valeur decimale 255
0x10 // valeur decimale 16
-0xDAD1 // valeur decimale -56017
```

- *Nombres à virgule flottante* : Comme les entiers, les nombres à virgule prennent leur valeur sur une plage fonction de la plate forme cible. Usuellement, de 1.7E-3308 à 1.7E+308, avec une précision de 15 chiffres après la virgule. Il est à noter que Php comprends les notations scientifiques :

```
3.14
0.017
-7.1
0.314E1 // 0.314*101, soit 3.14
17.0E-3 // 17.0*10-3, soit 0.017
```

Attention, la représentation des nombres à virgule flottante est souvent une approximation sur de nombreux systèmes. Par exemple, 3.1 est parfois représenté par la valeur 3.0999999999999999. Dans ce

cas, la comparaison de deux nombres réels peut poser certains soucis, et il est souvent nécessaire de passer par une conversion en entier pour vérifier une égalité à X décimales près.

- *Chaînes de caractères* : elles sont délimitées soit par des apostrophes, soit par des guillemets. Dans le premier cas, les variables incluses dans la chaîne de caractères ne sont pas résolues, dans le second, elles sont remplacées par leur valeur (cf paragraphe sur les variables). Il est à noter un certain nombre de caractères d'échappement :

```
\ " //le CARACTERE guillemets
\n //saut de ligne
\t //tabulation
\\ //CARACTERE Antislash
```

- *Booléens* : les booléens prennent les valeurs vrai ou faux. Par défaut, toutes les valeurs sont VRAI, sauf : le mot clef **FALSE**, l'entier 0, le réel 0.0, la chaîne vide "", un tableau avec aucun élément, ou la valeur **NULL**.
- *Arrays* : Les tableaux multidimensionnels sont gérés sous Php. Un paragraphe leur est consacré. ils permettent de stocker des valeurs multiples dans une même variable.

1.6 Variables

Une variable est un espace mémoire utilisé pour stocker une information. Elle est définie par un nom. On peut affecter une valeur à une variable, et utiliser cette valeur plus tard dans le programme. Un exemple d'utilisation des variables peut être de stocker les données entrées par un utilisateur, ou encore des résultats intermédiaires lors de traitements.

Si les noms des variables sont au choix du programmeur, il est important de noter quelques règles de nommages :

- les noms de variables commencent par un \$
- les noms de variables sont de longueur quelconque
- les noms de variables peuvent contenir des chiffres, des lettres et le signe underscore
- les noms des variables sont sensibles à la casse.

Contrairement à beaucoup d'autres langages, la déclaration des variables ne se fait pas en PHP. Cette déclaration est implicite au moment de la première affectation. *L'affectation* est l'instruction qui permet de stocker une valeur dans une variable, elle est définie par le signe =.

C'est donc la première affectation qui détermine le type de la variable. Le type d'une variable représente le type de données qu'elle va pouvoir contenir : numérique, caractère, tableau, ...

Exemple 1.2 Affectations et variables

```
<p>Hello World!
<? $age = 33;
    $nom = "Ivan"; ?>
<br />Hello <? echo $nom; ?>
<br />Tu as <? echo $age; ?> ans
</p>
```

NOTE



Il est possible de supprimer une variable (et pas seulement sa valeur) en utilisant la fonction **unset(\$var)**.

Le type d'une variable peut automatiquement évoluer dans le temps, en fonction des diverses affectations. Une variable de type entier peut donc devenir une variable de type texte. On appelle ce changement de type en cours d'exécution le *transtypage*. Dans certains calculs, Php tentera également de changer le type de certaines variables de manière à pouvoir évaluer certaines expressions : par exemple, la concaténation d'une chaîne et d'un entier provoquera le transtypage de l'entier vers une chaîne, puis la concaténation des deux chaînes. Par contre, l'addition d'une chaîne de caractère et d'un entier ne fonctionne pas (aucune erreur n'est renvoyé, mais seul l'entier est pris en compte dans le calcul).

1.7 Opérateurs

Nous avons vu comment déclarer des variables et leur affecter des valeurs. Voyons maintenant comment manipuler ces valeurs :

1.7.1 Opérateurs arithmétiques

TAB. 1 Opérateurs arithmétiques

Opérateur	Description
+	Addition
-	Soustraction
*	Multiplication
/	Division
%	Modulo (reste de la division entière)

1.7.2 Opérateurs de comparaison

Les opérateurs de comparaison permettent d'obtenir une valeur booléenne VRAI ou FAUX selon le résultat de l'expression :

TAB. 2 Opérateurs de comparaison

Opérateur	Description
==	Egalité
>	Strictement supérieur
>=	Supérieur ou égal
<	Strictement inférieur
<=	Inférieur ou égale
!=	Non égalité
<>	Non égalité

1.7.3 Opérateurs logiques

Les opérateurs logiques permettent par exemple de combiner les résultats de comparaisons. Ils renvoient la valeur booléenne VRAI ou FAUX.

TAB. 3 Opérateurs logiques

Opérateur	Vrai quand ...
OR	une des deux opérandes est VRAI
AND	les deux opérandes sont VRAI
XOR	Uniquement lorsqu'un opérande est VRAI, pas les deux

1.7.4 Opérateurs de transtypage

Bien que Php puisse être considéré comme un langage peu typé, il est possible de forcer le type des variables en utilisant les opérateurs de transtypage :

TAB. 4 Opérateurs logiques

Opérateur	Change le type vers :
(int)	entier
(float)	Réel
(string)	Chaîne de caractère
(bool)	booléen
(array)	Tableau

Exemple :

```
$a = "5" // $a est de type chaîne
$a = (int) $a; // $a devient de type entier
```

1.8 Tableaux

Les tableaux sont des structures de données permettant de stocker plusieurs valeurs dans une même variable. Très utilisés en PHP, on dispose de tableaux à une ou plusieurs dimensions.

1.8.1 Tableau à une dimension

Un tableau à une dimension permet donc de stocker plusieurs valeurs *de même type* dans une même variable. L'accès à chaque cellule du tableau se fait en utilisant un indice : une entier commençant à 0 ou une chaîne de caractère identifiant la cellule.

Exemple :

```
$prof[0] = "Kurzweg";
$prof[1] = "Picard";
$prof[2] = "Gouraud";
$prenoms_enseignant['Kurzweg'] = "Ivan";
$prenoms_enseignant['Picard'] = "Pascal";
$prenoms_enseignant['Gouraud'] = "Gilles";
```

Il est possible de déclarer et d'affecter des valeurs à un tableau en utilisant le constructeur `array()`.

Exemple :

```
$prof = array("Kurzweg", "Picard", "Gouraud");
$prenoms_enseignant = array('kurzweg' => "Ivan",
    'Picard' => "Pascal",
    'Gouraud' => "Gilles");
```

1.8.2 Tableau à plusieurs dimension

Les tableaux à plusieurs dimensions permettent de stocker plusieurs valeurs dans chaque cellule du tableau. Il faut néanmoins respecter l'ordre et le type de chaque valeur.

Exemple :

```
$prof[0][0] = "Kurzweg";
$prof[1][0] = "Ivan";
$prof[2][0] = "Picard";
$prof[0][1] = "Pascal";
$prof[1][1] = "Gouraud";
$prof[2][1] = "Gilles";
```

1.8.3 Fonctions sur les tableaux

Il existe un certain nombre de fonctions prédéfinies en Php, dont voici une liste non exhaustive :

TAB. 5 Fonctions concernant les tableaux

Fonctions	Commentaires	Exemple
count (\$var)	compte le nombre d'éléments affectés	<pre>\$dept = array ("974","972"); \$nb = count(\$dept);</pre>
sort(\$var)	trie les éléments selon un ordre numérique ou alphabétique et réaffecte les indices du tableau	<pre>\$dept=array ("des","dpa","dbe"); sort(\$dept); while (list(\$indice, \$val) =each(\$dept)) { echo ("\$indice - \$val"); // "0 - dbe" puis "1 - des" // puis "2 - dpa" }</pre>
asort(\$var)	trie les éléments mais ne réaffecte pas les indices	<pre>\$dept=array ("des","dpa","dbe"); asort(\$dept); // "2 - dbe" puis "0 - des" // puis "1 - dpa"</pre>

1.8.4 Parcours d'un tableau

Le parcours d'un tableau consiste à récupérer les valeurs stockées afin de les traiter (les afficher par exemple. Il existe de nombreuses manières de parcourir un tableau, mais une des plus courante est l'utilisation de la *boucle* **foreach** :

```
Exemple :
#!/usr/bin/php
<?
$prof = array("Kurzweg", "Picard", "Gouraud");
foreach ($prof as $nom)
{
    echo $nom, "\n";
}
?>
```

```
Exemple :
#!/usr/bin/php
<?
$enseignant = array('kurzweg' => "Ivan",
    'Picard' => "Pascal",
    'Gouraud' => "Gilles");

foreach ($enseignant as $nom => $prenom)
{
    echo $nom, "\t", $prenom, "\n";
}
?>
```

1.9 Garbage collector

Php utilise deux mécanismes de gestion de la mémoire : *copy-on-write* et *reference counting* :

- *copy-on-write* : lors d'affectation de variable à variable, Php retarde la copie de la valeur échangée jusqu'au moment où une modification intervient sur une des variables. Ainsi, seule la référence de la variable est modifiée. On économise ainsi mémoire et temps dans de nombreuses situations.

```
Exemple :
#!/usr/bin/php
<?
$enseignant = array("kurzweg", "Ivan", "33");

$prof = $enseignant; //on ne copie pas les valeurs du tableau, mais Php met
                    //juste à jour la table des symboles, indiquant que ←
                    pour
                    //le moment, $prof et $enseignant pointent sur les mêmes zones ←
                    mémoires

$prof[2] = "34";     //les valeurs sont modifiées, la variable $prof est ←
                    écrite en mémoire
?>
```

- *reference counting* : Php maintient à jour le nombre de références pointant sur les zones mémoire, c'est-à-dire le nombre de moyen d'aboutir à chaque valeur utilisée dans le programme. Dès que ce nombre atteint 0, la zone mémoire est libérée. Dans l'exemple précédent, l'entrée de la table des symboles est référencée 1 fois à la création du tableau `$enseignant`, puis référencé 2 fois à l'affectation `$prof = $enseignant`, puis référencé de nouveau 1 seule fois lors de l'écriture des valeurs de `$prof`.

1.10 Scripts interactifs

Dans ce cours, nous allons utiliser Php comme langage de scripts systèmes. Pour les besoins des exercices et des exemples, nous aurons besoin de saisir des données entrées par l'utilisateur. Pour effectuer cette saisie, nous utiliserons des fonctions de Php, permettant de lire dans un fichier (et sous Unix, tout est fichier ... le clavier peut donc être lu comme un fichier !). L'exemple suivant permet de lire un entier, puis une chaîne de caractère. Nous reviendrons dans le chapitre consacré aux fichiers sur ces fonctions ...

```
Exemple :#!/usr/bin/php
<?
echo "\n Entrez votre nom : \t";

//lecture de la chaîne de caractères
$nom = trim((string)fgets(STDIN)); //on supprime les espaces et le caractère ←
newline

echo "\n Entrez votre âge : \t";

//lecture de l'entier :
$age = (int)fgets(STDIN);

//message d'accueil :
echo "\nBonjour $nom, vous avez $age ans\n";

?>
ikare@ix $ ./saisie.php

Entrez votre nom :      Ivan

Entrez votre âge :      34

Bonjour Ivan, vous avez 34 ans
```

1.11 Exercices d'applications

1.11.1 Analyse de code

Interprétez et corrigez si nécessaires les extraits de code suivants :

- ```
$string = 'c'est surement un probleme de guillemets';
echo $string
```
- ```
$age = 12;  
$result1 = "$age";  
$result2 = "$age";  
echo $result1;  
echo "<br />";  
echo $result2;
```
- ```
$nb = 10;
$ch1 = "Il y a '$nb' personnes connectées";
$ch2 = "Il y a \"$nb\" personnes connectées. ";
echo $ch1, "
\n";
echo $ch2;
```
- ```
$string1 = 'Hello';  
$string2 = 'World!';  
$stringall = $string1.$string2;  
echo $stringall;
```
- ```
$nix = "BSD";
echo "Free$nix, Open$nix, PC$nix sont tous des Unix-like libres ...";
```
- ```
$number = 2;  
$string = "Hello";  
$combined = $number + $string;  
$combined2 = $number.$string;  
echo $combined;  
echo <br />;  
echo $combined2;
```
- ```
$cpt=4;
$cpt+=2;
$cpt-=3;
$cpt*=2;
$cpt/=3;
echo $cpt;
```

### 1.11.2 Instructions simples

1. Ecrivez le code permettant d'inverser le contenu de deux variables \$x et \$y. (au départ, \$x contient 2 et \$y contient 4, et c'est l'inverse en fin de programme). Le programme devra afficher le contenu des variables avant et après le traitement.
2. La surface et la circonférence d'un cercle de 2cm de rayon (PI et le rayon devront être stockés dans des variables)
3. Le résultat de l'évaluation de l'expression ((VRAI et FAUX) OU VRAI) ET FAUX
4. Le résultat de l'évaluation de l'expression ((VRAI et FAUX) OU (VRAI ET FAUX))
5. La phrase la date actuelle est :date et il est : heure

## 2 Structures de contrôles

### 2.1 Conditionnelles

#### 2.1.1 if

Les structures conditionnelles permettent de n'exécuter un bloc d'instructions que si une condition est réalisée. Le synopsis de la conditionnelle est le suivant :

```
if (condition1 ...❶)
{
 Bloc d'instructions 1..❷
}
elseif (condition2 ...) ❸
{
 Bloc d'instructions 2..❹
}
else❺
{
 Bloc d'instructions 3..
}
❻
```

- ❶ Une *expression* dont l'évaluation donne VRAI ou FAUX
- ❷ Si la condition 1 est vrai alors tout le bloc 2 est exécuté et le programme va en 6
- ❸ Si la condition 1 est FAUX alors la valeur de la condition 3 est testée
- ❹ Si la condition 2 est VRAI alors le bloc 4 est exécuté et le programme va en 6
- ❺ Si la condition 2 est FAUSSE alors le bloc 5 est exécuté et le programme va en 6
- ❻ Fin de la conditionnelle

Voici un exemple d'enchaînement de conditionnelles :

---

#### Exemple 2.1 Emboitements de condition

---

```
if ($country == "Germany")
{
 $version = "German";
 $message = " Sie sehen unseren Katalog auf Deutsch";
}
elseif ($country == "France")
{
 $version = "French";
 $message = " Vous verrez notre catalogue en francais";
}
elseif ($country == "Italy")
{
 $version = "Italian";
 $message = " Vedrete il nostro catalogo in Italiano";
}
else
{
 $version = "English";
 $message = "You will see our catalog in English";
}
echo "$message
";
```

### 2.1.2 Le Switch

Nous avons vu dans l'exemple précédent la nécessité de renoter plusieurs fois le même test d'égalité. Dans le cas de multiples tests de la valeur d'une variable, l'écriture peut devenir vite fastidieuse. C'est là qu'intervient le **switch** :

```
switch ($variable)
{
 case value :
 bloc d'instructions
 break;
 case value :
 bloc d'instructions
 break;
 ...
 default:
 bloc d'instructions
 break;
}
```

Il faut bien noter ici que l'on ne peut tester que la valeur d'une seule variable !

Un exemple d'utilisation du switch :

---

#### Exemple 2.2 Selon ...

---

```
switch ($dept)
{
 case 974 :
 $dept_nom = "Réunion";
 break;
 case 972 :
 $dept_nom = "Guyane";
 default:
 $dept_nom = "France métropolitaine";
 break;
}
```

---

## 2.2 Répétitives

Les structure répétitives (itératives) permettent de répéter un certain nombre de fois un bloc d'instructions. Nous en étudierons deux, la boucle **for** et la boucle **while**

### 2.2.1 La boucle for

L'instruction **for** permet de répéter un bloc d'instructions pour un nombre de fois défini. Elle doit définir la variable qui sera modifiée à chaque itération, sa valeur de départ, la valeur de fin de boucle, et l'opération à effectuer à chaque itération :

---

#### Exemple 2.3 Affichage des nombres inférieurs à 100

---

```
<?
for ($cpt=1;$cpt<100;$cpt++)
{
 echo "$cpt
";
}
?>
```

---

**Exemple 2.4** Calcul de la somme des nombres pairs inférieurs à 100

```
<?
$som=0;
$nb=100;
for ($cpt=1;$cpt<$nb+1;$cpt+=2)
{
 $som=$som+$nb;
}
echo "La somme des $nb premiers entiers est de $som";
?>
```

**2.2.2 La boucle while**

L'instruction **while** permet de répéter un bloc d'instructions tant qu'une condition n'est pas remplie. Elle suit le synopsis suivant :

```
while (condition❶)
{
 bloc d'instructions ... ;
}
```

❶ tant que cette condition est VRAI, on exécute le bloc et on réévalue la condition.

Bien sûr, il est à la charge du programmeur de bien s'assurer que la condition d'exécution de la boucle est modifiée et passe à un moment à FAUX .. sinon, on est dans le cas d'une boucle infinie.

**Exemple 2.5** Affichage des nombres inférieurs à 100

```
<?
$cpt=1;
//on affiche tous les nombres jusque 99
while ($cpt<100)
{
 echo "$cpt
";
 $cpt++;
}
?>
```

**Exemple 2.6** Une boucle infinie !!

```
<?
$cpt=1;
while ($cpt<100)
{
 echo "$cpt
";
 // $cpt++; on commente la ligne, la valeur de $cpt ne changera plus,
 // la boucle ne s'arrêtera pas
}
?>
```

**2.2.3 La boucle do ... while**

La boucle **do { }while** est similaire à la première, à la différence que la condition d'arrêt est examinée à la fin de la boucle et non plus au début :

```
do
{
 bloc d'instructions ... ;
}
while (condition❶)
```

- ❶ si cette condition est VRAI, on ré-exécute le bloc et on réévalue la condition.

Bien sûr, il est à la charge du programmeur de bien s'assurer que la condition d'exécution de la boucle est modifiée et passe à un moment à FAUX .. sinon, on est dans le cas d'une boucle infinie.

---

### Exemple 2.7 Affichage des nombres inférieurs OU ÉGAUX à 100

---

```
<?
$cpt=1;
//on affichera les nombres, 100 y compris
do
{
 echo "$cpt
";
 $cpt++;
}
while ($cpt<100)
?>
```

---

#### 2.2.4 La boucle foreach

La boucle **foreach**, abordée dans le premier chapitre, permet de parcourir un tableau

```
foreach ($tableau as $cellule❶)
{
 bloc d'instructions ... ;
}
```

- ❶ \$cellule prendra successivement les valeurs de chaque cellule du tableau.

Pour parcourir le tableau en accédant à la fois à la clef et à la valeur, on utilise la syntaxe suivante :

```
foreach ($tableau as $clef => $cellule❶)
{
 bloc d'instructions ... ;
}
```

- ❶ \$clef et \$cellule contiennent respectivement les valeurs de la clef et de la cellule.

Bien sûr, il est à la charge du programmeur de bien s'assurer que la condition d'exécution de la boucle est modifiée et passe à un moment à FAUX .. sinon, on est dans le cas d'une boucle infinie.

**Exemple 2.8** Affichage des nombres inférieurs OU ÉGAUX à 100

```

Exemple :
#!/usr/bin/php
<?
$prof = array("Kurzweg", "Picard", "Gouraud");
foreach ($prof as $nom)
{
 echo $nom, "\n";
}

$enseignant = array('kurzweg' => "Ivan",
 'Picard' => "Pascal",
 'Gouraud' => "Gilles");

foreach ($enseignant as $nom => $prenom)
{
 echo $nom, "\t", $prenom, "\n";
}
?>

```

Une utilisation des tableaux à plusieurs dimensions est également fréquente. Dans ce cas, il peut être intéressant d'imbriquer les parcours de chaque dimensions :

```

//tableau à deux dimension : catégorie et produits
$prod['vetements']['T-shirt'] = 20.00;
$prod['vetement']['pantalon'] = 22.50;
$prod['literie']['taie'] = 25.00;
$prod['literie']['drap'] = 50.00;
$prod['fourniture']['lampe'] = 44.00;
$prod['fourniture']['table'] = 75.00;

//récupération du prix d'un T-Shirt
$prix = $prod['vetements']['T-shirt'];

//affichage du prix d'un pantalon
echo $prod['vetement']['pantalon'];

//affichage d'un texte
echo "le prix d'une lampe est \${$prod['fourniture']['lampe']}";

echo "<table border=1>";
foreach($prod as $categorie)
{
 foreach($categorie as $produit => $prix)
 {
 $ch_prix = sprintf("%01.2f", $prix);
 echo "<tr><td>$produit:</td><td>\$$ch_prix</td></tr>";
 }
}
echo "</table>";

```

**2.2.5 Sortie d'une boucle**

Même si les boucles devraient être écrites pour s'arrêter d'elles mêmes, il peut être intéressant de forcer la sortie, avec l'instruction **break**. **break** permet de sortir complètement de la boucle et de reprendre le cours du programme après le bloc d'instructions.

```

$cpt = 0;
while ($cpt < 5)

```

```

{
 $cpt++;
 If ($cpt == 3)
 {
 echo "break
";
 break;
 }
 echo "Fin de boucle: cpt=$cpt
";
}
echo "Sortie de boucle<p>";

```

## 2.3 Exercices d'applications

### 2.3.1 Conditionnelles

1. Ecrire un script qui lit l'âge de l'utilisateur et affiche : 1. Vous êtes un enfant (moins de 18 ans) 2. Vous êtes un adultes 3. ou Vous êtes du 3ème age (plus de 60 ans)
2. Ecrire un script qui lit deux entiers et dit si les deux entiers sont multiples ou non. Le programme doit être capable de détecter si le premier est multiple du second, ou le second du premier.
3. Ecrire un script qui simule le fonctionnement d'un calculatrice : l'utilisateur entre un premier nombre, puis un signe ( + - / \* ). Le programme donne le résultat.
4. Ecrire un script qui calcule un montant de facture : à partir d'un prix unitaire, du nombre de produits, du taux de TVA, le programme renvoie le montant total (avec une réduction de 10% si le montant dépasse 1000 euros.)
5. Ecrire un programme qui indique si une année est bissextile ou non. Une année est dite bissextile si elle divisible par 4, sauf si elle est divisible par 100, par contre celles divisibles par 400 sont bissextiles.

### 2.3.2 Boucles

1. Affectez une valeur à la variable `nbre` et afficher la somme des entiers de 1 à `nbre`. Donnez les versions du programme en utilisant l'instruction **FOR** puis avec l'instruction **WHILE**.
2. Modifier le script précédent pour calculer la somme des entiers pairs uniquement.
3. Ecrire un script qui lit 10 réels, puis donne la somme des entiers, la moyenne, le nombre maximum entré, ainsi que le minimum.
4. Modifier le programme précédent pour qu'on saisisse un nombre indéfini de réels. La saisie se termine quand l'utilisateur entre -1.
5. Afficher la table de multiplication de 10.
6. Ecrire un script qui affiche un carré d'étoiles, dont la taille est entrée au clavier. Par exemple, pour une taille de 4, le programme affichera :

```



```

7. Même exercice, mais cette fois ci avec un triangle, dont la nombre de lignes est entrée au clavier. Par exemple, pour une taille de 4, le programme affichera :

```

 *


```

8. Un nombre entier est dit premier s'il n'est divisible que par 1 et par lui-même (1,2,3,5,7,11,13,17,19,...). Ecrire un script qui lit au clavier un nombre et indique si il est premier ou non. Vérifiez que votre programme marche correctement dans le cas d'une saisie d'un grand nombre pair (100 000 000 000) : la réponse doit être immédiate.
9. Modifiez le script précédent pour qu'il affiche les nombres premiers inférieurs à 100 000.
10. Un nombre entier est dit parfait si il est égal à la somme de ses diviseurs. Ecrire un script qui lit au clavier un nombre et indique si il est parfait ou non.
11. Modifiez le script précédent pour qu'il affiche les nombres parfaits inférieurs à 100 000.

### 2.3.3 Tableaux

1. Ecrire un script initialise une liste de 10 élèves (noms et prénoms), les stocke dans un tableau, puis réaffiche la liste.
2. Modifier le programme précédent pour qu'il lise un réel associé à chaque élève. En fin de saisie, le programme devra afficher la moyenne de la classe, puis le nom du meilleur élève.
3. Modifier le programme de manière à fournir le nombre d'élèves au dessus de 10, et le nombre d'élèves en dessous de la moyenne générale.
4. Sur la même base de tableau d'élève, écrivez le programme qui permet de vérifier si un élève est dans la classe. On saisira son nom au clavier, et en cas d'homonymie, on donnera la liste des élèves portant le même nom (par exemple, Olivier Grondin et Pierre Grondin).
5. Stocker dans un tableau les 10 premiers multiples d'un entier saisi au clavier.

### 2.3.4 Carré magique

Un tableau de nombres entiers est dit *magique* quand la somme des cases de chaque colonne est égale à la somme des cases de chaque ligne, et à la somme des cases de chaque diagonale.

Vous devez écrire une page PHP qui affiche le tableau suivant, et qui prouve que le carré est magique. Vérifiez en changeant une des valeurs. Toutes les valeurs du carré doivent être stockées dans une seule variable.

**TAB. 6** Carré magique

|    |    |    |    |    |
|----|----|----|----|----|
| 17 | 24 | 1  | 8  | 15 |
| 23 | 5  | 7  | 14 | 16 |
| 4  | 6  | 13 | 20 | 22 |
| 10 | 12 | 19 | 21 | 3  |
| 11 | 18 | 25 | 2  | 9  |

## 3 Fonctions

### Résumé

Une fonction est un bloc d'instructions nommé, pouvant accéder à des valeurs qui lui sont passées en arguments, les *paramètres* de la fonction. Permettant de réutiliser du code plusieurs fois dans un même programme, voire dans des programmes différents, elles sont essentielles dans la programmation

### 3.1 Appel d'une fonction

Qu'elles soient des fonctions prédéfinies (primitives du langage) ou définies par l'utilisateur, les fonctions sont appelées dans le code selon la même syntaxe :

```
$var = nom_fonction(parametre1, parametre2, ...)
```

Le nombre de paramètres dépend de la définition de la fonction. Dans le cas de fonctions prédéfinies où incluses dans des modules externes, la documentation précise le nombre et le type des paramètres (voir [www.php.net](http://www.php.net)).

Les exemples suivants illustrent des appels à des fonctions de Php :

```
// strlen() est une fonction qui retourne la taille d'une chaîne de caractères
$lg = strlen("PHP"); // $lg contient 3
// unlink permet d'effacer un fichier
$result = unlink("functions.txt") or die("Impossible !");
```

Dans le second exemple la fonction `unlink()` qui demande l'effacement d'un fichier retourne comme valeur `FALSE` si le traitement échoue. De nombreuses fonctions sont implémentées de cette manière en PHP.

### 3.2 Définition d'une fonction

Une fonction est définie dans un script par la syntaxe suivante :

```
function nom_fonction ([parametre [, ...]])
{
 Instructions ...
}
```

Le nom de la fonction est soumis aux mêmes règles que le nom des variables, mais il est insensible à la casse.

Toute fonction doit retourner une valeur. Ce renvoi est effectif à l'exécution de l'instruction **return**. Cette instruction a pour effet de terminer l'appel de la fonction, et de revenir au code d'où a été effectué cet appel.

---

#### Exemple 3.1 Concaténation de chaînes

---

```
function concat($gauche, $droite) {
 $chaîne = $gauche . $droite;
 return $chaîne;
}
```

Dans l'exemple précédent, la fonction `concat` accepte deux paramètres, effectue la concaténation des chaînes de caractères, stocke cette concaténation dans une variable, et renvoi la valeur de cette variable. Le passage par une variable intermédiaire est inutile :

---

#### Exemple 3.2 Concaténation de chaînes améliorée

---

```
function concat($gauche, $droite) {
 return $gauche . $droite;
}
```

Un exemple de script complet :

---

**Exemple 3.3** Concaténation de chaînes

---

```
<?php
function concat($gauche, $droite) {
 return $gauche . $droite;
}
$ch1 = "c'est une ";
$ch2 = "phrase complète";
echo concat($ch1, $ch2);
?>
```

### 3.3 Portée des variables

Jusque maintenant, sans les fonctions, nous avons toujours considéré qu'une variable pouvait être utilisée dans n'importe quelle instruction d'un script. Mais avec les fonctions, qui peuvent contenir leur propres variables nécessaires à leurs traitements, ce n'est plus le cas. Les variables internes aux fonctions ne sont effectivement plus visible à l'extérieur de la fonction :

---

**Exemple 3.4** Portée d'une variable

---

```
#!/usr/bin/php
<?
$a = 3;
function foo() {
 $a += 2;
}
foo();
echo "\n", $a, "\n";
?>

ikare@ix $./portee.php

3
```

Dans cet exemple, la valeur de `$a` n'a pas changé dans le programme principal : le `$a` de la fonction est une variable distincte, dont la durée de vie est celle de la fonction.

#### 3.3.1 Variables globales

Dans certains cas, il peut être utile de pouvoir accéder à une variable depuis une fonction. Il faut à ce moment là déclarer la variable comme *globale* :

---

**Exemple 3.5** variable globale

---

```
#!/usr/bin/php
<?
$a = 3;
function foo() {
 global $a;
 $a += 2;
}
foo();
echo "\n", $a, "\n";
?>

ikare@ix $./globale.php

5
```

---

### 3.3.2 Variables statiques

Une variable est dite statique quand sa valeur est partagée par tous les appels à la fonction, c'est-à-dire que sa valeur n'est pas réinitialisée à chaque appel :

---

**Exemple 3.6** Variable statique

---

```
#!/usr/bin/php
<?
function compteur() {
 static $cpt = 0;
 return $cpt++;
}
for ($i = 1; $i <= 5; $i++) {
 echo compteur();
}

?>
ikare@ix $./static.php
01234
```

---

## 3.4 Paramètres

### 3.4.1 Passage par valeur

Dans la plupart des cas, le passage des paramètres depuis le programme appelant vers la fonction se fait *par valeur* : la valeur du paramètres est assignée à la variable correspondante dans la fonction.

### 3.4.2 Passage par adresse

Le passage des paramètres par adresse (encore appelé passage par référence), permet de contourner la portée d'une variable en la rendant directement accessible depuis le corps de la fonction. Il s'agit en effet de passer non plus la valeur contenue dans la variable, mais l'adresse mémoire de la variable, en utilisant le symbole &.

```
function doubler(&$value) {
 $value = $value*2;
}
$a = 3;
```

```
doubler($a);
echo $a;
```

Dans l'exemple précédent, la fonction `doubler` n'a pas d'instructions **return**. La variable passée en paramètre est accessible par la fonction via son adresse mémoire, sa référence. A noter l'appel de la fonction, qui s'effectue cette fois ci sans affectation !

### 3.5 Récursivité

Une fonction récursive est une fonction qui est utilisée dans sa propre définition. Par exemple, on peut définir la fonction factorielle comme suit :

```
0! = 1
n! = n*(n-1)! pour n > 0
```

Dans une fonction récursive, il y a toujours deux choses :

- un cas de base non récursif, la condition d'arrêt
- un cas de récursion qui rapproche du cas de base, si bien qu'à chaque appel, on se rapproche de la fin du calcul.

Il est prouvé théoriquement que chaque fonction récursive peut s'écrire avec des boucles, et inversement, mais dans certains cas, l'une ou l'autre des manières permet une écriture simplifiée des algorithmes.

---

#### Exemple 3.7 Fonctions factorielle itérative et récursive

---

```
#!/usr/bin/php
<?
function fact_iterative($n) {
 $res=1;
 for ($cpt=1;$cpt<=$n;$cpt++)
 $res=$res*$cpt;
 return $res;
}

function fact_recursive($n) {
 if ($n <= 0)
 return 1;
 else
 return $n*fact_recursive($n-1);
}

echo "\nFactorielle it~rative de 5 = \t",fact_iterative(5);
echo "\nFactorielle r~cursive de 5 = \t",fact_recursive(5);

?>
ikare@ix $./fact.php

Factorielle it~rative de 5 = 120
Factorielle r~cursive de 5 = 120
```

Lors des appels récursifs à la fonction, les paramètres de la fonction sont empilés, en attendant le retour de la condition d'arrêt. A ce moment là, les paramètres sont dépilés, en calculant à chaque fois leur valeur définitive.

## 3.6 Exercices d'applications

### 3.6.1 Fonctions

1. Ecrire une fonction qui renvoie le maximum de deux entiers. En utilisant cette fonction, écrire un programme qui saisit 3 entiers au clavier, et qui les réaffiche dans l'ordre croissant.
2. Ecrire une série de fonctions sur les tableaux d'entiers, avec les scripts qui permettent de les tester :
  - une fonction qui détermine si un nombre est présent dans un tableau
  - une fonction qui donne l'indice de la première occurrence d'un nombre dans un tableau. Si ce nombre n'apparaît pas, la fonction retourne -10
  - une fonction qui retourne le nombre d'occurrence d'un nombre dans un tableau.
  - une fonction qui retourne un tableau contenant les 10 premiers multiples d'un nombre n donné en paramètre.
3. Une permutation est un arrangement d'éléments pris dans un ensemble fini. La fonction de permutation  $P(n,k)$  donne le nombre de permutations différentes de k éléments pris dans un ensemble de n. On peut calculer cette fonction en utilisant la formule :  $P(n,k) = n! / (n-k)!$ . Coder la fonction en Php.

### 3.6.2 Récursivité

1. Ecrire une fonction récursive qui calcule la somme des n premiers carrés (1+4+9+...)
2. Ecrire une fonction récursive puissance qui prend en paramètre deux entiers : le nombre à élever à la puissance, et la puissance.
3. Ecrire une fonction exponentielle Exp, qui calcule  $xy$  sans utiliser l'opérateur \*\* mais en exploitant la définition récursive :

```
xy = 1, si y=0,
xy = (x*x)y/2, si y est pair,
xy = (x*x)y/2*x, si y est impair.
```

4. Ecrire une fonction recursive Palindrome qui teste si un mot est un palindrome.
5. Ecrire une fonction Paire\_E qui teste si un mot contient un nombre pair de 'E'.

## 4 Fichiers

## 5 Objets

## 6 Bases de données

## 7 Références

### 7.1 Références

- [1] Welling Luke et Thomson Laura, *PHP and MySQL - Web Development*, SAMS Edition .
- [2] Ratschiller Tobias et Gerken Till, *Web Application Development with Php 4.0*, New Riders .
- [3] Valade Janet, *PHP and MySQL for Dummies*, Wiley Publishing, Inc .
- [4] Lane David, *Web database applications*, O'Reilly .
- [5] Rasmus Lerdorf, *Programming PHP*, O'Reilly .