LES PROCÉDURES ET LES FONCTIONS

8

PLAN

OBJECTIFS

➤ Maîtriser la programmation en modules de traitements autonomes avec passage de paramètres et retour du traitement.

8.1 INTRODUCTION

Les chapitres précédents ont étudié les données, ainsi que les instructions qui les manipulent. Les traitements présentés jusque-là dans les exemples et les exercices sont séquentiels, et la répétition d'un même traitement dans le programme suppose la répétition des lignes de codes qui l'effectuent. Or une programmation ne doit pas être *linéaire* mais *modulaire*. Si un ensemble d'instructions correspond à un traitement particulier, il faut le définir en module autonome, un *sous-programme*.

Ce chapitre aborde la programmation structurée. Il montre comment créer des sous-programmes et organiser le programme en un programme principal appelant des sous-programmes implémentant des traitements spécifiques.

8.2 Principe

Notion de sous-programme

Une *procédure* et une *fonction* sont des sous-programmes. Un sous-programme se comporte exactement comme un programme, il possède des variables et des calculs qui lui sont propres. Il effectue des traitements autonomes, et il est appelé par le programme principal, ou par un autre sous-programme. Un fichier source PHP peut contenir plusieurs sous-programmes.

Rôle et type de sous-programme

Un sous-programme regroupe un ensemble d'instructions qui sont vues par le programme appelant comme une unique instruction. Cela permet de construire un traitement « autonome » en lui donnant un *nom* à travers lequel il sera appelé, et qui peut accepter des données en entrée, ses *arguments* ou *paramètres*, à partir desquels il effectue le traitement, et éventuellement retourner un *résultat*. Un sous-programme trouve son utilité dans les cas suivants :

- quand un ensemble d'instructions sont répétées plusieurs fois : il faut créer un sous-programme unique, facile à maintenir, qui est appelé plusieurs fois ;
- quand il faut organiser le programme en traitements distincts et autonomes. Cette programmation par « modules de traitement » rend la programmation efficace et évolutive. Ainsi une fonction de recherche pourra être améliorée, en changeant uniquement son algorithme interne, sans modifier le reste du programme qui appelle cette fonction, dès lors que les interfaces (structure des données en entrée et en sortie de la fonction) restent inchangées.

La bibliothèque PHP

Le langage PHP propose un très grand nombre de fonctions « prêtes à emploi ». Les chapitres précédents en ont présenté quelques-unes.

Remarque

Il est toujours préférable de vérifier qu'une fonction existe dans la bibliothèque PHP avant d'en créer une nouvelle.

La documentation en ligne

L'index des fonctions, par ordre alphabétique, est accessible à l'URL :
http://php.net/manual/fr/indexes.functions.php

Une liste par catégorie est disponible à l'URL :
http://fr.php.net/manual/fr/funcref.php

On y trouve des fonctions classiques comme :

- les fonctions de traitement de chaînes de caractères ;
- les fonctions mathématiques ;
- les fonctions sur les fichiers.

Mais également des fonctions avancées comme :

- les fonctions de requêtes sur des bases de données ;
- les fonctions de cryptage ;
- les fonctions d'envoi de courriel ;
- les fonctions de gestion de la date et de l'heure ;
- les fonctions de traitement d'images ;
- les fonctions de tri ou de recherche ;
- etc.

8.3 Écriture d'une fonction

Nous présentons l'écriture d'une nouvelle fonction ou procédure et son utilisation.

Déclaration

La syntaxe de déclaration d'une fonction est :

```
function nomfonction($param1,$param2,…)
{
 ...
 return $valeur_retour ;
}
```

où :

- `nomfonction` est le nom choisi pour la fonction. Le nom d'une fonction suit la même règle que celui d'une variable (voir section 4.1 du chapitre 4) ;
- `$param1,$param2,...` est la liste des paramètres ;
- `$valeur_retour` est la valeur retournée par la fonction ;

Remarque

La présence de l'instruction `return` est propre à la fonction qui retourne une valeur. Si cette instruction est absente, c'est une procédure. Même en sa présence (fonction), l'appel par le programme appelant peut ignorer la valeur retournée, comme dans le cas d'une procédure.

Le nom des fonctions est insensible à la casse, mais il est préférable de conserver la même syntaxe à la déclaration et lors de l'appel !

Le programme `fonction_exemple1_shell.php` présente la fonction `Addition()` ayant deux paramètres. Elle est appelée une première fois comme une *procédure*, puis une seconde fois comme une fonction.

Listing 8.1 – Programme fonction_exemple1_shell.php

```
<?php
function Addition($i,$j) // --- Déclaration de la fonction ---
{echo "    --> Dans la fonction : " ;
 $somme=$i+$j;
 echo "$i+$j=$somme".PHP_EOL;
 return $somme;
}
echo "Entrez 2 entiers :"; // Saisie de deux entiers
fscanf(STDIN,"%d %d",$a,$b);
echo "--- Appel type procédure ---".PHP_EOL ;
Addition($a,$b); //Premier appel
echo "--- Appel type fonction ---".PHP_EOL ;
$resultat=Addition(2*$a,2*$b); //Second appel
echo "Retour de l'appel = $resultat".PHP_EOL ;
?>
```

Lors de son exécution, la fonction affiche le traitement et retourne le résultat. Au premier appel, seul l'affichage interne de la fonction apparaît. Au second appel, l'affichage interne apparaît, et la valeur retournée est récupérée par le programme principal dans la variable `$resultat`.

Listing 8.2 – Exécution de fonction_exemple1_shell.php

```
$ php fonction_exemple1_shell.php
Entrez 2 entiers :3 4
--- Appel type procédure ---
    --> Dans la fonction : 3+4=7
--- Appel type fonction ---
    --> Dans la fonction : 6+8=14
Retour de l'appel = 14
```

Visibilité

Une fonction (ou procédure) peut être déclarée n'importe où dans le fichier. Même déclarée à la fin, elle est utilisable dès le début. Les fonctions en PHP ont une *portée globale*, elles peuvent être définies à l'intérieur d'une autre fonction, et être utilisées à l'extérieur. Néanmoins, si la fonction est définie de manière conditionnelle (à l'intérieur d'un test), alors elle ne peut être utilisée qu'après sa définition.

Exemples

Le programme `fonction_exemple2_shell.php` déclare la fonction en fin du fichier. Elle est utilisable dès le début du programme. Les « ... » remplacent les lignes identiques au programme précédent.

Listing 8.3 – Programme fonction_exemple2_shell.php

```
<?php
echo "Entrez 2 entiers :";
fscanf(STDIN,"%d %d",$a,$b);
echo "--- Appel type fonction ---".PHP_EOL ;
$resultat=Addition(2*$a,2*$b);
echo "Retour de l'appel = $resultat".PHP_EOL ;
// Déclaration de la fonction à la fin du programme
function Addition($i,$j)
{ ... }
?>
```

Le programme `fonction_exemple3_shell.php`, présente la déclaration conditionnelle de la fonction `Division()`. Elle se trouve dans un `if` et l'appel de la fonction est avant sa déclaration.

Listing 8.4 – Programme fonction_exemple3_shell.php

```
<?php
echo "Entrez 2 entiers :";
fscanf(STDIN,"%d %d",$a,$b);
$resultat=Division($a,$b); // Appel de la fonction Division()
echo "$a / $b = $resultat".PHP_EOL ;
if ($b >=0)
{// Déclaration de la fonction Division() dans le if
 function Division($i,$j)
 {$res=$i/$j;
  return $res;
 }
}
?>
```

À l'exécution, l'appel de la fonction `Division()` échoue.

Listing 8.5 – Exécution de fonction_exemple3_shell.php

```
$ php fonction_exemple3_shell.php
Entrez 2 entiers :3 4
Fatal error: Call to undefined function Division() in /.../fonction_
exemple3_shell.php on line 4
```

Le programme s'exécute correctement si l'appel intervient après la déclaration.

8.4 Passage de paramètres

Les variables transmises aux fonctions sont les *paramètres* ou *arguments*. On appelle *paramètres formels* les variables utilisées à la déclaration et *paramètres effectifs*, les variables passées par le programme appelant au moment de l'appel.

Il existe plusieurs méthodes pour passer des données aux fonctions :

- le passage par valeur ;
- le passage par adresse ou référence ;
- les valeurs par défaut ;
- la liste variable d'arguments.

Le passage par valeur

Avec ce type de transmission de données, les variables déclarées dans l'en-tête de la fonction (paramètres formels) reçoivent *une copie de la valeur* des variables passées par le programme appelant (paramètres effectifs). C'est le comportement pas défaut de PHP.

Le programme `fonction_param_valeur_shell.php` en présente un exemple :

Listing 8.6 – Programme fonction_param_valeur_shell.php

```
<?php
function Addition($i,$j) // Déclaration de la fonction
{echo "   --> Dans Addition() : ".PHP_EOL;;
 echo "       valeurs initiales : i=$i, j=$j".PHP_EOL;
 $somme=$i+$j;
 $i++; // modification de $i
 $j++; // modification de $j
 echo "       valeurs finales   : i=$i, j=$j".PHP_EOL;
 return $somme;
}
echo "Entrez 2 entiers : ";
fscanf(STDIN,"%d %d",$a,$b);
echo "--- Appel de Addition($a,$b) ---".PHP_EOL;
$resultat = Addition($a,$b);
echo "--- Retour de l'appel ---".PHP_EOL;
echo "$a + $b = $resultat".PHP_EOL ;
?>
```

Son exécution montre que les variables `$a` et `$b` transmettent leur valeur respectivement à `$i` et `$j`. `$i` et `$j` sont modifiées dans la fonction, mais cela n'impacte aucunement `$a` et `$b` qui conservent leur valeur.

Listing 8.7 – Exécution de fonction_param_valeur_shell.php

```
$ php fonction_param_valeur_shell.php
Entrez 2 entiers : 3 7
--- Appel de Addition(3,7) ---
   --> Dans Addition() :
       valeurs initiales : i=3, j=7
       valeurs finales   : i=4, j=8
--- Retour de l'appel ---
3 + 7 = 10
```

La figure 8.1 représente le passage de paramètres par valeur.

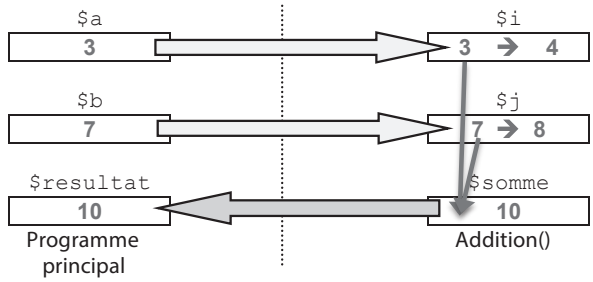

Figure 8.1 – Passage par valeur.

Le passage par adresse ou référence

Avec ce type de transmission des données, les variables déclarées dans l'en-tête de la fonction reçoivent *l'adresse* des variables utilisées au moment de l'appel. Elles pointent sur *la même zone en mémoire* que les variables transmises par le programme appelant. Il faut utiliser le symbole **&** devant la variable devant recevoir l'adresse, dans la déclaration de la fonction. Par exemple :

```
function Addition($i,$j,&$somme)
```

Le programme `fonction_param_ref_shell.php` présente une procédure (aucune instruction `return`) ayant trois paramètres : les deux premiers `$i` et `$j` sont passés par valeur, le troisième paramètre `$somme` est passé par adresse.

Listing 8.8 – Programme fonction_param_ref_shell.php

```
<?php
function Addition($i,$j,&$somme) // Déclaration de la fonction
{echo "    --> Dans Addition() : ".PHP_EOL;;
 echo "        valeurs initiales : i=$i, j=$j".PHP_EOL;
 $somme=$i+$j;
}
$resultat=0;
echo "Entrez 2 entiers : ";
fscanf(STDIN,"%d %d",$a,$b);
// Appel de la fonction
echo "--- Appel de Addition($a,$b,$res) ---".PHP_EOL;
Addition($a,$b,$resultat);
echo "--- Retour de l'appel ---".PHP_EOL;
echo "$a + $b = $resultat".PHP_EOL ;
?>
```

Son exécution montre que les variables `$a` et `$b` transmettent respectivement leur valeur à `$i` et `$j`. `$resultat` transmet son adresse à `$somme`. Les variables `$resultat` et `$somme` pointent sur le même espace en mémoire**.** La modification de `$somme` modifie également `$resultat`.

Listing 8.9 – Exécution de fonction_param_ref_shell.php

```
$ php fonction_param_ref_shell.php
Entrez 2 entiers : 3 7
--- Appel de  Addition(3,7,) ---
    --> Dans Addition() :
        valeurs initiales : i=3, j=7
--- Retour de l'appel ---
3 + 7 = 10
```

La figure 8.2 représente le passage de paramètres par adresse de `$somme`.

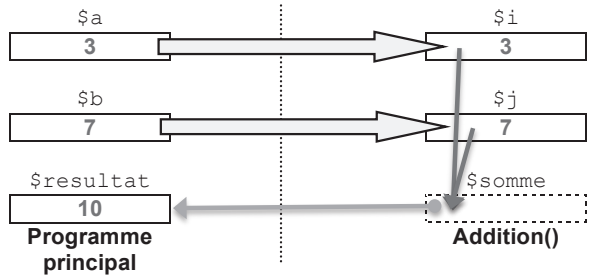

Figure 8.2 – Passage par adresse.

Les valeurs par défaut

Il est possible de fixer une *valeur par défaut* à chaque paramètre. Si aucune donnée n'est fournie lors de l'appel, la valeur par défaut est utilisée. Cela s'applique aux paramètres scalaires : booléens, entiers, réels et chaînes de caractères. Les valeurs par défaut sont des *constantes*, aucune variable ni fonction n'est autorisée.

Remarque

Les paramètres n'ayant aucune valeur par défaut doivent être indiqués en premier et les paramètres avec des valeurs par défaut doivent être en fin de la liste.

Les syntaxes suivantes présentent une fonction ayant ses *deux paramètres* avec des *valeurs par défaut* (`-1`). Le premier appel utilise les variables `$a` et `$b`. Le second appel n'utilise aucune variable, les valeurs par défaut (`-1`) sont utilisées.

```
function Addition($i=-1,$j=-1)
{$somme=$i+$j;
 return $somme;
}
$resultat=Addition($a,$b); // addition de $a et $b
$resultat=Addition();      // addition de -1 et -1
```

Les syntaxes suivantes présentent une fonction ayant son *deuxième paramètre* avec des valeurs par défaut (`-1`). L'appel transmet la première variable, la valeur par défaut (`-1`) est utilisée pour la seconde.

```
function Addition($i,$j=-1)
{$somme=$i+$j;
 return $somme;
}
$resultat=Addition($a); // addition de $a et -1
```

L'en-tête de la fonction ci-dessous ayant ses paramètres mal ordonnés (valeur par défaut en premier et non en fin de liste) provoque une erreur d'exécution :

```
function Addition($j=-1, $i) // erreur de conception
```

Liste variable de paramètres

Le langage PHP supporte une liste variable de paramètres. La syntaxe est implémentée par " ... " en PHP 5.6 et ultérieure, ou les fonctions `func_num_args()`, `func_get_arg()` et `func_get_args()` en PHP 5.5 et antérieure.

La syntaxe « ... »

Avec cette syntaxe, les arguments sont passés dans un paramètre de type tableau.

Le programme `fonction_param_liste_variable1_shell.php` utilise le tableau `$parametres` pour récupérer la liste des valeurs fournies au moment de l'appel.

Une boucle `foreach` récupère chaque valeur et l'ajoute à la variable `$somme`.

La fonction est appelée avec deux valeurs, puis avec quatre valeurs.

Listing 8.10 – Programme fonction_param_liste_variable1_shell.php

```
<?php
// Déclaration de la fonction syntaxe PHP 5.6 et +
function Addition(...$parametres)
{$somme=0;
 foreach($parametres as $val) {$somme += $val;}
 return $somme;
}
echo "Entrez 2 entiers : ";
fscanf(STDIN,"%d %d",$a,$b);
echo "--- Appel de Addition($a,$b) ---".PHP_EOL;
$resultat=Addition($a,$b);
echo "$a+$b = $resultat".PHP_EOL ;
echo "Entrez 4 entiers : ";
fscanf(STDIN,"%d %d %d %d",$c,$d,$e,$f);
echo "--- Appel de Addition($c,$d,$e,$f) ---".PHP_EOL;
$resultat=Addition($c,$d,$e,$f);
echo "$c+$d+$e+$f = $resultat".PHP_EOL ;
?>
```

Voici son exécution :

Listing 8.11 – Exécution de fonction_param_liste_variable1_shell.php

```
$ php fonction_param_liste_variable1_shell.php
Entrez 2 entiers : 3 4
--- Appel de Addition(3,4) ---
3+4 = 7
Entrez 4 entiers : 3 4 5 6
--- Appel de Addition(3,4,5,6) ---
3+4+5+6 = 18
```

La syntaxe " ... " peut également être utilisée pour des fonctions ayant un nombre fixe de paramètres. Il faut l'utiliser au moment de l'appel pour extraire les valeurs d'un tableau. `fonction_param_extract_tab_shell.php` présente la fonction `Addition()` ayant deux paramètres et son appel sur le tableau `$tab`.

Listing 8.12 – Programme fonction_param_extract_tab_shell.php

```
<?php
function Addition($i,$j) // Déclaration de la fonction
{$somme=$i+$j;
 return $somme;
}
$tab=array(3,4); // Création du tableau avec deux valeurs
echo '--- Appel de Addition(...$tab) ---'.PHP_EOL;
$resultat=Addition(...$tab);
echo "resultat=$resultat".PHP_EOL ;
?>
```

Les fonctions func_num_args(), func_get_arg() et func_get_args()

Cette syntaxe s'applique pour les versions de PHP 5.5 et antérieures. Aucun paramètre n'est indiqué dans l'en-tête de la fonction au moment de sa déclaration, les informations sur la liste variable d'argument sont récupérées *via* ces trois fonctions.

- `func_num_args()` : retourne le nombre d'arguments passés à la fonction ;
- `func_get_arg()` : retourne un élément de la liste des arguments ;
- `func_get_args()` : retourne les arguments sous la forme d'un tableau.

Le programme `fonction_param_liste_variable2_shell.php` utilise ces fonctions. Une boucle `for` récupère chaque valeur et l'ajoute à la variable `$somme`.

Listing 8.13 – Programme fonction_param_liste_variable2_shell.php

```
<?php
// Déclaration de la fonction syntaxe PHP 5.5 et antérieure
function Addition()
{$nbarguments=func_num_args();
 $tab_arguments=func_get_args();
```

```
 $prem_argument=func_get_arg(0);
 $dern_argument=func_get_arg($nbarguments-1);
 echo "--- Etats des arguments ---";
 echo "Nombre d'arguments : $nbarguments".PHP_EOL;
 echo "Premier argument : $prem_argument".PHP_EOL;
 echo "Dernier argument : $dern_argument".PHP_EOL;
 $somme=0;
 for ($i=0; $i<$nbarguments ; $i++)
 {$somme += $tab_arguments[$i];}
 return $somme;
}
echo "Entrez 2 entiers : ";
fscanf(STDIN,"%d %d",$a,$b);
echo "--- Appel de Addition($a,$b) ---".PHP_EOL;
$resultat=Addition($a,$b);
echo "$a + $b = $resultat".PHP_EOL ;
echo "Entrez 4 entiers : ";
fscanf(STDIN,"%d %d %d %d",$c,$d,$e,$f);
echo "--- Appel de Addition($c,$d,$e,$f) ---".PHP_EOL;
$resultat=Addition($c,$d,$e,$f);
echo "$c+$d+$e+$f = $resultat".PHP_EOL ;
?>
```

Voici son exécution :

Listing 8.14 – Exécution de fonction_param_liste_variable2_shell.php

```
$ php fonction_param_liste_variable2_shell.php
Entrez 2 entiers : 3 4
--- Appel de Addition(3,4) ---
--- Etats des arguments ---Nombre d'arguments : 2
Premier argument : 3
Dernier argument : 4
3 + 4 = 7
Entrez 4 entiers : 3 4 5 6
--- Appel de Addition(3,4,5,6) ---
--- Etats des arguments ---Nombre d'arguments : 4
Premier argument : 3
Dernier argument : 6
3+4+5+6 = 18
```

8.5 Retour d'une fonction

Le retour du résultat du traitement de la fonction utilise le mot-clef `return`. Si cette instruction est omise, la fonction retourne la valeur NULL. Une *seule valeur* peut être retournée, mais elle peut être de *n'importe quel type*. Pour retourner plusieurs valeurs, il faut les inclure dans un tableau et le retourner.

8.6 Variables locales et globales

Principe

Selon qu'une variable est déclarée dans une fonction ou à l'extérieur, sa visibilité (portée) change. On parle de variables *locales* ou *globales*. Cette notion a été abordée en détail à la section 4.5 du chapitre 4. Nous rappelons brièvement ces notions.

Variables locales

Une variable locale est déclarée dans un contexte, comme le bloc principal, ou un sous-programme, et elle *n'est connue que de ce contexte*. Elle est créée à l'appel du sous-programme et détruite à la fin de celui-ci. À chaque nouvel appel du sous-programme, elle est à nouveau créée. Le programme `variables_locales_shell.php` de la section 4.5 du chapitre 4 montre la visibilité des variables locales.

Variables locales statiques

Une variable locale statique est une variable locale, elle n'est connue que de son contexte de déclaration et est créée lors du premier appel du sous-programme. Mais à la différence des variables locales « simples », elle n'est pas supprimée à la fin du sous-programme et conserve sa valeur lors des appels successifs. Sa déclaration est précédée du mot `static`. En voici un exemple :

```
function somme($a,$b)
{static $compteur=1;
 ...
}
```

Le programme `variables_statiques_shell.php` de la section 4.5 du chapitre 4 présente ces variables.

Variables globales

Une variable globale est connue de tous les sous-programmes. Elle est définie dans le programme et est « redéfinie » comme globale dans les sous-programmes. Cela se fait grâce au mot `global` comme dans la syntaxe suivante :

```
function somme()
{global $a, $b ;
 ...
}
```

Le programme `variables_globales1_shell.php` de la section 4.5 du chapitre 4 présente une mise en œuvre complète.

La déclaration de variables globales peut aussi se faire grâce au tableau associatif prédéfini `$GLOBAL[ ]`. Voici un exemple de syntaxe :

```
function somme()
{
  $result = $GLOBALS["a"] + $GLOBALS["b"];
  ...
 }
```

8.7 Fonction variable

Le langage PHP supporte la notion de fonction variable, c'est-à-dire le fait d'affecter le nom d'une fonction à une variable et d'utiliser cette variable suivie des éventuels arguments pour appeler la fonction.

Le programme `fonction_variable_shell.php` présente cette syntaxe. Deux fonctions sont déclarées `Addition()` et `Soustraction()`. Selon la valeur saisie « + » ou « – », la variable `$fonction` recevra le nom de la fonction `Addition` ou `Soustraction`, puis sera exécutée avec les deux valeurs entières saisies.

Listing 8.15 – Programme fonction_variable_shell.php

```
<?php
function Addition($i,$j)
{$res=$i+$j;
 return $res;
}
function Soustraction($i,$j)
{$res=$i-$j;
 return $res;
}
echo "Entrez 2 entiers : ";
fscanf(STDIN,"%d %d",$a,$b);
echo "Entrez une opération (+ ou -) : ";
fscanf(STDIN,"%s",$ope);
// Sélection de la fonction
if ($ope=="+") $fonction='Addition';
else $fonction='Soustraction';
// Appel de la fonction variable
echo "--- Appel de $fonction($a,$b) ---".PHP_EOL;
$resultat=$fonction($a,$b);
echo "$a $ope $b = $resultat".PHP_EOL ;
?>
```

Voici son exécution :

Listing 8.16 – Exécution de fonction_variable_shell.php

```
$ php fonction_variable_shell.php
Entrez 2 entiers : 3 4
```

```
Entrez une opération (+ ou -) : +
--- Appel de Addition(3,4) ---
3 + 4 = 7
```

8.8 Fonctions sur les fonctions

Le tableau 8.1 présente quelques fonctions de gestion des fonctions. La liste complète est disponible à l'URL : http://php.net/manual/fr/ref.funchand.php.

Tableau 8.1 – Fonctions sur la gestion des fonctions

Fonction	Exemple
`call_user_func`	Appelle une fonction de rappel
`func_get_arg`	Retourne un élément de la liste des arguments
`func_get_args`	Retourne les arguments d'une fonction (tableau)
`func_num_args`	Retourne le nombre d'arguments passés
`function_exists`	Indique si une fonction est définie
`get_defined_functions`	Liste toutes les fonctions définies

8.9 La récursivité

Principe

Les syntaxes présentées jusque-là s'appuient sur la méthode itérative, basée sur les boucles, qui décrit le calcul pas à pas. La récursivité propose une autre approche des traitements, à la fois plus séduisante, plus simple à écrire, mais souvent plus complexe à concevoir.

La description d'un calcul basé sur la méthode itérative définit un traitement élémentaire qui progresse grâce à une boucle, du premier jusqu'au dernier calcul. À la fin de la boucle, le dernier traitement donne le résultat final. Ainsi, le calcul de x^N se résume à calculer x^1, puis x^2, x^3, … N fois. Il progresse à chaque nouvelle itération de la boucle, en réutilisant le résultat de l'itération précédente et en le multipliant par x, ce qui correspond au calcul suivant :

$$x^N = (\ldots(\,(\,(\,(\,(x)*x)*x\,)*x\,)*x\,)*\ldots)*x \qquad N \text{ fois}$$

La récursivité propose une approche opposée. Le calcul final est exprimé en fonction du calcul précédent, ce qui donne : $x^N = x*x^{N-1}$. Avec cette méthode, le calcul se définit par lui-même : c'est la notion de récursivité. Ce formalisme mathématique donne l'impression de ne pas avancer dans la recherche de la solution, car si on connaît la

valeur de x, on ne connaît pas celle de x^{N-1}. En poursuivant le raisonnement, on peut exprimer x^{N-1} comme égal à $x*x^{N-2}$, et ainsi de suite, jusqu'à obtenir $x^1 = x*x^0$ puis $x^0 = 1$. Ainsi, de proche en proche, le calcul devient :

$$x^N = x*x^{N-1} = x*(x*x^{N-2}) = x*(\ x*(x*x^{N-3})) = x*(x*(x*(\ldots*(x*x^0)\ldots)))$$

La figure 8.3 présente la comparaison entre ces deux méthodes sur le calcul de x^4.

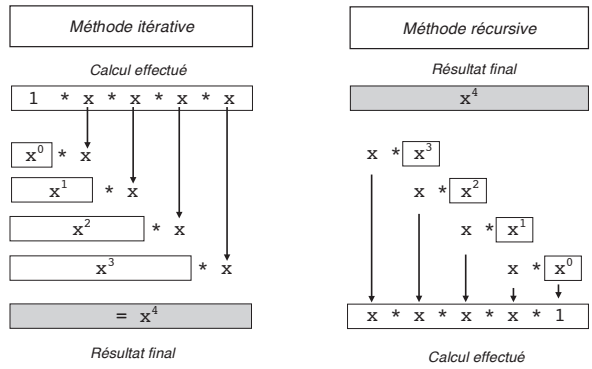

Figure 8.3 – Comparaison entre calcul itératif et récursif.

La méthode itérative utilise clairement une boucle. La valeur initiale (en dehors de la boucle) est 1, soit x^0. La méthode récursive n'utilise pas de boucle, puisque le calcul est exprimé en fonction d'un autre qui n'est pas encore connu, sauf pour la récursion terminale (x^0) dont on peut connaître la valeur (la récursion terminale est la dernière étape du traitement récursif). Le traitement est donc formalisé par une fonction (ou une procédure) qui décrit le calcul plutôt que de le faire. On parle de *fonction récursive* car elle s'appelle elle-même. Le fonctionnement d'une telle fonction se passe en deux phases : la descente en profondeur par appels récursifs, puis la remontée du résultat de chaque appel. Voici le détail de ce fonctionnement sur le calcul de la puissance.

Le premier appel de la fonction `puissance(x,N)` effectue le calcul de `x*puissance(x,N-1)`. Ce calcul reste en « suspens », car il déclenche l'appel à la fonction `puissance(x,N-1)` qui effectue le calcul de `x*puissance(x,N-2)`. Ce calcul reste lui-même en attente du résultat du nouvel appel à la fonction `puissance(x,N-2)`, qui calcule `x*puissance(x,N-3)`, etc. On voit que, s'il n'y a pas de boucle, il y a bien une répétition de calculs par une suite d'appels en cascade de la même fonction avec des exposants de plus en plus petits. Chaque appel est suspendu par un nouvel appel, dont il attend le résultat. Si l'enchaînement des appels n'est pas contrôlé, on aboutit à une fonction récursive infinie qui provoque un arrêt brutal du programme. Il faut donc prévoir le cas de la récursion terminale, c'est-à-dire de l'appel dont le résultat est obtenu de manière itérative. Dans notre exemple, cela se produit quand la puissance est égale à 0. Alors, le résultat de la fonction est 1. Quand le « fond » des appels récursifs est atteint (la récursion terminale donne un résultat et ne fait plus d'appel), l'appel le plus profond se termine et retourne son résultat à l'appel

précédent. Cet appel, qui était en attente d'un résultat, reçoit la valeur de retour, poursuit son calcul, et retourne son résultat à l'appel précédent, qui reprend le traitement suspendu et retourne son résultat, et ainsi de suite. Cette phase constitue la remontée du résultat de chaque appel. Enfin, le premier appel reçoit le résultat final. Le fonctionnement des appels récursifs, avec une phase de descente et une phase de remontée, est présenté à la figure 8.4, qui calcule x^4. Pour simplifier la présentation, `puissance(x,N)` est noté `P(x,N)`.

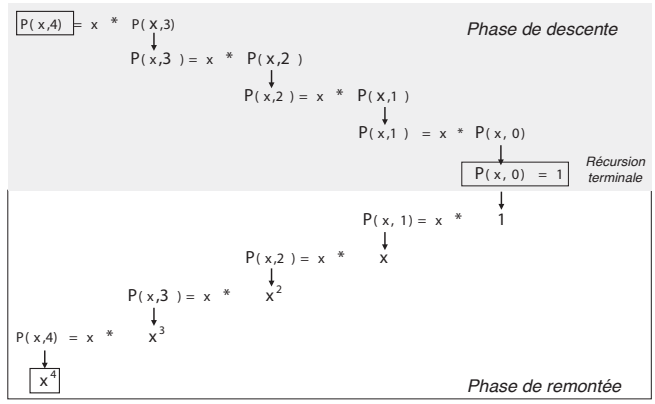

Figure 8.4 – Fonctionnement des appels récursifs.

L'élégance de cette méthode de programmation tient au fait que le calcul s'effectue de lui-même avec, comme seules informations, la valeur de la récursion finale et la description du calcul. C'est à la fois simple et puissant. Concevoir une fonction récursive suppose le respect de deux règles : définir la *récursion terminale* et décrire la *méthode de calcul* à partir de la fonction elle-même.

Syntaxe

Les syntaxes suivantes présentent la fonction `puissance(x,N)` itérative utilisant une boucle. C'est la méthode de programmation utilisée jusque-là dans cet ouvrage.

```
function puissance($x,$n)
{$res=1; // Valeur initiale
 for ($i=0 ; $i<$n; $i++) // boucle de calcul
 {$res=$res*$x;}
 return $res;
}
```

Une fonction `puissance(x,N)`récursive est principalement constituée d'un test :

- Si le résultat du calcul est connu, par exemple *x0* qui vaut 1, on retourne la valeur connue. C'est la récursion terminale !
- Sinon on appelle la fonction pour effectuer calcul suivant : `puissance(x,N-1)`.

Voici le programme `fonction_puissance_recursive_shell.php` qui utilise la fonction `puissance(x,N)`récursive :

Listing 8.17 – Programme fonction_puissance_recursive_shell.php

```
<?php
function puissance($x,$n)
{if ($n == 0) $res=1;
 else   $res=$x*puissance($x,$n-1);
 return $res;
}
echo "Entrez 2 entiers : "; // Saisie de deux entiers
fscanf(STDIN,"%d %d",$a,$b);
$resultat=puissance($a,$b); // Appel de la fonction
echo "$a puissance $b = $resultat".PHP_EOL ;
?>
```

Voici son exécution :

Listing 8.18 – Exécution de fonction_puissance_recursive_shell.php

```
$ php fonction_puissance_recursive_shell.php
Entrez 2 entiers : 2 4
2 puissance 4 = 16
```

Exercices

8.1 Cet exercice regroupe différents traitements vus dans ce chapitre et dans les chapitres précédents. Chaque procédure ou fonction peut être développée séparément. Une fois le programme principal développé, chaque développement d'une nouvelle fonctionnalité (fonction) constitue un exercice à part entière.

Faire un programme qui gère une liste de personnes dans un environnement shell. Chaque personne est caractérisée par son *identifiant*, son *nom*, son *prénom* et son *âge*. Le programme principal présentera le menu suivant :

```
-1- Saisie d'une liste de personnes
-2- Affichage de toutes les personnes
-3- Sauvegarde dans un fichier
-4- Chargement d'un fichier
-0- Quitter
Choix (1,2,3,...) :
```

Chaque choix appellera une fonction ou une procédure qui effectuera le traitement demandé. Le tableau des personnes et le nombre de personnes seront définis comme des variables globales. Le tableau des personnes sera toujours trié et les doublons seront supprimés.

Choix 1 : voici un exemple de la saisie des personnes :

```
Choix (1,2,3,...) : 1
Entrez un ID, un nom, un prénom et un âge (ex : 1;Dupont;Jean;28) :
22;de la fontaine ; jean ; 110
Entrez un ID, un nom, un prénom et un âge (ex : 1;Dupont;Jean;28) : 33
; martin ; jean christophe ; 21
Entrez un ID, un nom, un prénom et un âge (ex : 1;Dupont;Jean;28) :
10;dupont;jean;28
Entrez un ID, un nom, un prénom et un âge (ex : 1;Dupont;Jean;28) :
```

Les contraintes à respecter sont :

- Une ligne vide provoque une fin de saisie.
- La saisie des noms et des prénoms composés devra être possible.
- Les noms et les prénoms seront normalisés. Pour cela, faites une fonction de normalisation des noms et des prénoms. Dans un premier temps, cette fonction convertira uniquement en majuscule. Dans un deuxième temps, les noms et prénoms seront normalisés en majuscules, sans accent, ni valeur numérique, ni apostrophe. Les espaces multiples seront remplacés par un seul espace, puis chaque espace d'un nom ou d'un prénom composé sera remplacé par un tiret « – ». Par exemple « Léry » sera normalisé en « LERY », « dE la fONtaine » sera normalisé en « DE–LA–FONTAINE », ou « d'ortal » sera normalisé en « D–ORTAL ». De même pour les prénoms, « André pieRRe » sera transformé en ANDRE–PIERRE.
- L'identifiant de la personne ainsi que son âge seront convertis en entier.
- L'identifiant de la personne devra être unique, c'est-à-dire non utilisé par une autre personne.
- Si un des éléments est vide, ou si l'identifiant ou l'âge ne sont pas des numériques, un message d'erreur apparaît, et la personne n'est pas rangée dans le tableau des personnes.
- Les informations d'une personne saisie sont rangées dans un tableau associatif `$une_personne[]` avec les étiquettes « ID », « nom », « prenom », « age », puis chaque personne sera ensuite rangée dans un tableau numérique de personnes, `$tab_personnes[]`.
- Le tableau des personnes, `$tab_personnes[]`, sera trié à la fin de la saisie, et les doublons supprimés.

La figure 8.5 présente le tableau des personnes.

$tab_personnes

Lignes	colonnes ID	nom	prenom	age
0	10	DUPONT	JEAN	28
1	22	DE-LA-FONTAINE	JEAN	110
2	33	MARTIN	JEAN-CHRISTOPHE	21

Figure 8.5 – Tableaux des personnes.

Choix 2 : voici un exemple de l'affichage des personnes

```
Choix (1,2,3,...) : 2
-------------------------------------------------------------
Numéro :     ID Nom                  Prénom               Age
-------------------------------------------------------------
     0 :     10 DUPONT               JEAN                  28
     1 :     22 DE-LA-FONTAINE       JEAN                 110
     2 :     33 MARTIN               JEAN-CHRISTOPHE       21
-------------------------------------------------------------
```

Les contraintes à respecter sont :

- avant chaque affichage le tableau sera trié (par défaut selon l'identifiant=ID), et les doublons supprimés ;
- la présentation de l'affichage d'un tableau sera réutilisable par d'autres modules comme lors d'une recherche multiple : faire une procédure d'affichage ayant un tableau de personnes à afficher en argument.

Choix 3 : voici un exemple de la sauvegarde dans un fichier de personnes

```
Choix (1,2,3,...) : 3
Nom du fichier de sauvegarde : liste.txt
3 personne(s) sauvegardée(s) dans le fichier Sauvegardes/liste.txt
```

Les contraintes à respecter sont :

- Le nom du fichier sera normalisé *via* une fonction de normalisation :
  - Le nom du fichier sera en minuscules, sans accent. Les valeurs numériques seront autorisées. Les espaces multiples seront remplacés par un seul espace, puis chaque espace sera remplacé par un « _ ».
  - L'extension sera toujours .txt. Si le nom saisi ne contient pas d'extension, elle sera ajoutée, sinon l'extension saisie sera remplacée.
  - Le répertoire de sauvegarde sera « Sauvegardes/ ».
- Chaque ligne du fichier produit contiendra les informations d'une personne : l'identifiant (ID), le nom, le prénom et l'âge de la personne. Le caractère séparateur sera la tabulation.

Voici un exemple de fichier.

```
$ cat liste.txt
10  DUPONT          JEAN             28
22  DE-LA-FONTAINE  JEAN             110
33  MARTIN          JEAN-CHRISTOPHE  21
```

Choix 4 : voici un exemple du chargement de personnes

```
Choix (1,2,3,...) : 4
Nom du fichier à charger : liste personnes
Lecture de : Sauvegardes/liste_personnes.txt
15 personne(s) lue(s)
```

Les contraintes à respecter sont :

- Le nom du fichier sera normalisé de la même manière que pour la sauvegarde (même fonction de normalisation).
- Le chargement du fichier remplace toutes les données actuellement dans le tableau des personnes.
- L'identifiant de la personne devra être unique, c'est-à-dire non utilisé par une autre personne. Dans le cas contraire, la personne en conflit n'est pas chargée.
- Le tableau des personnes sera trié à la fin du chargement et les doublons supprimés.

Choix 0 : voici un exemple de la fin du programme

```
Choix (1,2,3,...) : 0
Au revoir !
```

Solutions

8.1 Cet exercice gère une liste de personnes. Les personnes sont conservées dans un tableau `$tab_personnes[]` qui est défini en global. Une variable globale `$sauvegarde_a_faire` indique s'il y a eu insertions de données. Elle est utilisée par `Chargement()` et `Quitter()` afin de faire une sauvegarde avant de charger un nouveau fichier ou de quitter le programme. Toutes les fonctions sont dans le même fichier. Certaines fonctions comme `affichage_tab_personnes()`ont été conçues pour être facilement réutilisables dans d'autres fonctions. Un ensemble de fonctions « outils » permettent d'effectuer les traitements de base comme la normalisation des noms et des prénoms. Dans la fonction `supprime_accents()`, certains accents sont remplacés par des « … ».

Voici le programme `fonction_liste_personnes_shell.php`.

Listing 8.19 – Programme fonction_liste_personnes_shell.php

```
<?php
define("NON_TROUVE",-1)  ;
$choix=-1                ;
$tab_personnes=array()   ;
```

```
$nbpersonnes=0           ;
$sauvegarde_a_faire=false;
// --- Boucle de menu    ---
while ($choix !=0)
{ $choix=-1;
  echo "-1- Saisie d'une liste de personnes".PHP_EOL;
  echo "-2- Affichage de toutes les personnes".PHP_EOL;
  echo "-3- Sauvegarde dans un fichier".PHP_EOL;
  echo "-4- Chargement d'un fichier".PHP_EOL;
  echo "-0- Quitter".PHP_EOL;
  echo "Choix (1,2,3,...) : ";
  fscanf(STDIN,"%d",$choix);
  switch($choix)
  { case 1 : Saisie()    ;
             break;
    case 2 : Affichage() ;
             break;
    case 3 : Sauvegarde();
             break;
    case 4 : Chargement();
             break;
    case 0 : quitter();
             break;
    default : echo "Choix impossible !".PHP_EOL;
             break;
  }
}
// --- Saisie d'une liste de personnes ---
function Saisie()
{global $nbpersonnes       ;
 global $tab_personnes     ;
 global $sauvegarde_a_faire;
 $entree="saisie non vide";
 while (!empty($entree))
 {echo "Entrez un ID, un nom, un prénom et un âge (ex :
 1;Dupont;Jean;28) : ";
  $entree=fgets(STDIN) ;
  // On supprime les espaces (début et fin) et saut de ligne
  $entree=trim($entree);
  // On vérifie que la saisie n'est pas vide
  if (!empty($entree))
  {$erreur_saisie=false;
   // Rangement dans le tableau associatif
   list($une_personne['ID'],$une_personne['nom'],$une_
   personne['prenom'],$une_personne['age'])=explode(';',$entree);

   $indice=count($une_personne);
   $nb_champs=$indice+1;
   if ($indice <3)
   { echo "  Erreur : La saisie ne contient que $nb_champs champs !
   ".PHP_EOL;
```

```
      $erreur_saisie=true;
    }
    else
    { // Normalisation_nom et vérification de chaque champ
      foreach($une_personne as $etiquette => $val_champ)
      { if (($etiquette == 'nom') || ($etiquette == 'prenom') )
        { $une_personne[$etiquette]=normalisation_nom($val_champ);
        }
        if (($etiquette == 'ID') || ($etiquette == 'age'))
        { $val_champ=intval($val_champ)       ;
          $une_personne[$etiquette]=$val_champ;
          if ($etiquette == 'ID')
          {$ID_unique=true;
           $ID_unique=verif_unicite_ID($val_champ);
           if ($ID_unique == false)
           { echo "  Erreur : Valeur du champ $etiquette est déjà utilisé !
           ".PHP_EOL;
             $erreur_saisie=true;
           }
          }
          if ($val_champ <= 0)
          { echo "  Erreur : Valeur du champ $etiquette doit être un
          entier positif ! ".PHP_EOL;
            $erreur_saisie=true;
          }
        }
        if (empty($val_champ))
        { echo "  Erreur : Valeur du champ $etiquette  est non valide !
        ".PHP_EOL;
          $erreur_saisie=true;
        }
      }
    }
    if (!$erreur_saisie)
    { // On renverse le tableau $une_personne à cause de l'ordre inverse
    de list
      $tab_personnes[]=array_reverse($une_personne);
      $nbpersonnes++;
      $sauvegarde_a_faire=true;
    }
    else
    {echo "  Erreur : Merci de saisir à nouveau les informations !".PHP_EOL;
    }
   }
  }
}
// --- Affichage de la liste des personnes ---
function Affichage()
{global $nbpersonnes   ;
 global $tab_personnes ;
 // Tri du tableau en retirant les doublons
```

```
 sort($tab_personnes);
 $tab_personnes=array_unique($tab_personnes,SORT_REGULAR );
 if (count($tab_personnes) == 0)
 {echo "Aucune personne à afficher !".PHP_EOL;
 }
 else
 {affichage_tab_personnes($tab_personnes);
 }
}
// --- Sauvegarde dans un fichier ---
function Sauvegarde()
{global $nbpersonnes       ;
 global $tab_personnes     ;
 global $sauvegarde_a_faire;
 // Tri du tableau en retirant les doublons
 sort($tab_personnes);
 $tab_personnes=array_unique($tab_personnes,SORT_REGULAR );
 $sauvegarde_confirmee=false;
 if ($nbpersonnes == 0)
 {echo "Aucune personne à sauvegarder !".PHP_EOL;
  echo "La sauvegarde videra le fichier !".PHP_EOL;
  echo "Confirmez la sauvegarde (o/n) :";
  fscanf(STDIN,"%s",$reponse);
  if ($reponse == "o")
  {$sauvegarde_confirmee=true;
  }
 }
 if (($nbpersonnes != 0) || $sauvegarde_confirmee)
 {echo "Nom du fichier de sauvegarde : ";
  $NomFichier=fgets(STDIN);
  // On normalise le nom du fichier
  $NomFichier=normalisation_fichier($NomFichier);
  // -- Ouverture du fichier en mode écriture ---
  $f1 = @fopen($NomFichier, "wt");
  if (! $f1) // --- erreur d'ouverture ---
  { // -- On affiche les messages d'erreur ---
    echo "Erreur d'écriture du fichier $NomFichier".PHP_EOL;
  }
  else
  {// --- Boucle de sauvegarde ---
foreach ($tab_personnes as $etiquette => $une_personne)
   {// Importation des variables à partir de l'étiquette des champs
    extract($une_personne,EXTR_OVERWRITE);
    fprintf($f1,"%d\t%s\t%s\t%d%s",$ID,$nom,$prenom,$age,PHP_EOL);
   }
   fclose($f1);
   if ($nbpersonnes == 0)
    echo "le fichier $NomFichier a été vidé".PHP_EOL;
   else
    echo "$nbpersonnes personne(s) sauvegardée(s) dans le fichier
    $NomFichier".PHP_EOL;
```

```
   }
   $sauvegarde_a_faire=false;
  }
}
// --- Chargement d'un fichier ---
function Chargement()
{global $nbpersonnes       ;
 global $tab_personnes     ;
 global $sauvegarde_a_faire;
 verif_sauvegarde()        ;
 // Chargement remplace les données => on vide le tableau
 $nbpersonnes=0;
 // On supprime les éléments actuels du tableau
 array_splice($tab_personnes,0,count($tab_personnes));
 // On met à jour le nombre de personnes
 $nbpersonnes=0;
 // On charge le fichier
 echo "Nom du fichier à charger : ";
 $NomFichier=fgets(STDIN);
 $NomFichier=normalisation_fichier($NomFichier);
 echo "Lecture de : $NomFichier".PHP_EOL;
 // -- Ouverture du fichier en mode lecture ---
 $f1 = @fopen($NomFichier, "rt");
 if (! $f1) // --- erreur d'ouverture ---
 { // -- On affiche les messages d'erreur ---
   echo "Erreur de lecture du fichier $NomFichier".PHP_EOL;
 }
 else
 { // --- Boucle de chargement ---
   while ($Tab_Info=fscanf($f1,"%d\t%s\t%s\t%d"))
   {list ($ID,$nom,$prenom,$age) = $Tab_Info;
    $nom   =normalisation_nom($nom)   ;
    $prenom=normalisation_nom($prenom);
    // Vérification que cet ID n'est pas déjà utilisé
    $ID_unique=true;
    $ID_unique=verif_unicite_ID($ID);
    if ($ID_unique == false)
    { echo "Erreur : L'identifiant $ID est déjà utilisé par : ".PHP_EOL;
      $tab_numero=Recherche_sur_critere('a',$ID);
      foreach($tab_numero as $numero)
      {$tab_num_personnes[$numero]=$tab_personnes[$numero];
      }
      affichage_tab_personnes($tab_num_personnes);
     echo "==> Pas de chargement de : $ID\t$nom\t$prenom\t$age".PHP_EOL;
    }
    else
    { $tab_
personnes[]=array('ID'=>$ID,'nom'=>$nom,'prenom'=>$prenom,'age'=>$age);
      $nbpersonnes++;
    }
   }
```

```
    echo "$nbpersonnes personne(s) lue(s)".PHP_EOL;
    fclose($f1);
  }
}
// --- Fonction quitter ---
function Quitter()
{verif_sauvegarde();
 echo "Au revoir !".PHP_EOL;
}
// --- Fonction outil de suppression des accents ---
function supprime_accent($chaine)
{// Tableau des caractères accentués à remplacer
  $caracteres_a_remplacer = array('À','Á','Â','Ã','Ä','Å',
  'Æ','Ç','È','É','Ê','Ë', ... ,'Ǽ','ǽ','Ǿ','ǿ');
  // Tableau des caractères sans accent de remplacement
  $caracteres_de_remplacement=array('A','A','A','A','A','A',
  'AE','C','E','E','E','E', ... ,'AE','ae','O','o');
 return str_replace($caracteres_a_remplacer, $caracteres_de_
 remplacement, $chaine);
}
// --- Fonction outil de normalisation des noms  ---
function normalisation_nom($chaine)
{// Tableau des motifs de recherche
 $tab_motif=array('/[^a-zA-Z -]/', '/[ -]+/', '/^-|-$/');
 // Tableau des caractères de remplacement
 $tab_remplacement=array('', '-', '')     ;
 // Chaîne sur laquelle s'effectue le remplacement
 $chaine_contexte=supprime_accent($chaine);
 // Retour de la fonction
 return strtoupper(preg_replace($tab_motif,$tab_remplacement, $chaine_
 contexte));
}
// -- Fonction outil de normalisation des noms de fichiers  --
function normalisation_fichier($nomf)
{ // Tableau des motifs de recherche
  $tab_motif=array('/[^a-zA-Z0-9 ._-]/','/[ -]+/','/^-|-$/');
  // Tableau des caractères de remplacement
  $tab_remplacement=array('', '-', '')   ;
  // Remplacement des caractères non autorisés
  $nomf_retour=strtolower(preg_replace($tab_motif,$tab_
  remplacement,supprime_accent($nomf)));
  $nomf_retour=str_replace('-','_',$nomf_retour);
  // Suppression de l'éventuelle extension
  $elements_chemin=pathinfo($nomf_retour);
  $nomf_retour=$elements_chemin['filename'];
  // -- On prépare le nom du répertoire de sauvegarde ---
  $repertoire_courant=realpath(".")      ;
  $repertoire_sauvegarde="../Sauvegardes/"  ;
  // -- On traite le nom du fichier ---
  $nomf_retour=$repertoire_sauvegarde.$nomf_retour.".txt";
  return $nomf_retour;
```

```
}
// -- Fonction outil d'affichage d'un tableau de personnes  --
function affichage_tab_personnes($tpersonnes)
{// Entête de l'affichage
 echo "-----------------------------------------------".PHP_EOL;
 fprintf(STDOUT,"%6s : %6s %-20s %-20s %4s
 %s","Numéro","ID","Nom","Prénom"," Age",PHP_EOL);
 echo "-----------------------------------------------".PHP_EOL;
 foreach ($tpersonnes as $indice => $une_personne)
 {// Importation des variables à partir de l'étiquette des champs
  extract($une_personne,EXTR_OVERWRITE);
  fprintf(STDOUT,"%6d : %6d %-20s %-20s %4d
  %s",$indice,$ID,$nom,$prenom,$age,PHP_EOL);
 }
 echo "-----------------------------------------------".PHP_EOL;
}
// -- Fonction outil de vérification de sauvegarde ---
function verif_sauvegarde()
{global $sauvegarde_a_faire;
 // On vérifie d'abord s'il y a une sauvegarde à faire
 if ($sauvegarde_a_faire)
 {echo "Attention les données ont été modifiées. !".PHP_EOL;
  echo "Voulez-vous sauvegarder les données actuelles avant de charger
  le fichier (o/n) : ";
  fscanf(STDIN,"%s",$reponse);
  if ($reponse == "o")
  {Sauvegarde();
  }
 }
}
// -- Fonction outil pour l'unicité de l'identifiant --
function verif_unicite_ID($identifiant)
{global $tab_personnes     ;
 $colonne_ID = array_column($tab_personnes, 'ID');
 $identifiant_existe=in_array($identifiant,$colonne_ID);
 $identifiant_valide = ! $identifiant_existe;
 return $identifiant_valide ;
}
?>
```