LE LANGAGE PHP

TODO :

-

v1.2.0.0 – 07/05/2010
peignotc(at)arqendra(dot)net / peignotc(at)gmail(dot)com

TABLE DES MATIÈRES

0 PROLOGUE 4
1 INTRODUCTION AU LANGAGE PHP 5
2 NOTIONS DE BASE 6
2.1 GÉNÉRALITÉS 6
2.1.1 Écriture de code PHP 6
2.1.2 Interprétation du code PHP 7
2.2 LES VARIABLES 8
2.2.1 Les types de variables 8
2.2.2 Les variables numériques 8
2.2.3 Les chaînes de caractères 9
2.2.4 Les tableaux 9
2.3 LES STRUCTURES DE CONTRÔLE 10
2.3.1 Les opérateurs 10
2.3.2 Les tests 10
2.3.3 Les boucles 10
3 LES FONCTIONS 12
3.1 FONCTIONS STANDARD 12
3.1.1 Fonctions numériques 12
3.1.2 Fonctions de chaînes 12
3.1.3 Fonctions de tableaux 12
3.2 FONCTIONS UTILISATEUR 13
4 GESTION DES BASES DE DONNÉES 14
4.1 INTRODUCTION AUX BASES DE DONNÉES 14
4.1.1 Principes et structure d'une base de données 14
4.1.2 Exploitation d'une base de données 15
4.1.3 Le langage SQL 15
4.2 COMMANDES D'ACCÈS À LA BASE 16
4.2.1 Commandes principales 16
4.2.2 Commandes de gestion de base 17
4.3 GESTION EN PHP 17
4.3.1 Connexion à une base de données 17
4.3.2 Création d'une base de données 18
4.3.3 Fermeture d'une base de données 18
4.3.4 Écriture dans une base de données 18
4.3.5 Lecture dans une base de données 18
4.3.6 Gestion des erreurs 19
5 GESTION DES FICHIERS 21
5.1 GESTION DE L'ACCÈS 21
5.1.1 Pointeur sur un fichier et ouverture du fichier 21
5.1.2 Fermeture du fichier 21
5.2 LECTURE ET ÉCRITURE 21

5.2.1 Écriture dans un fichier 21
5.2.2 Lecture dans un fichier 22

TABLE DES ANNEXES

A BIBLIOGRAPHIE 23

TABLE DES ILLUSTRATIONS

Figure 1.1 : communication client-serveur avec traitement PHP 5
Figure 2.1 : exemple simple de codage PHP 6
Figure 2.2 : principe de génération de page web 7
Figure 2.3 : exemple du traitement effectué par l'interpréteur PHP 8
Figure 4.1 : exemple de lecture dans une base de données et de traitement des résultats 19
Figure 5.1 : exemple de lecture de fichier 22

0 PROLOGUE

Le présent document n'a pas pour but de présenter de manière exhaustive tous les secrets du langage PHP, mais d'offrir un aperçu rapide du champ d'application, des possibilités, et de la syntaxe de ce langage.

Il tient pour acquis un certain nombre de concepts et d'éléments de base de la programmation et communs à différents langages. La maîtrise de notions comme les principes de « mot-clef », d'« instruction » et de « variable », les tableaux, les structures de contrôle et les fonctions, telles qu'on peut les appréhender notamment dans l'étude du langage C, C++, Java, JavaScript, ou C#, etc. est impérative pour comprendre les informations de ce document.

Il tient également pour acquis l'ensemble des éléments du langage HTML dans le cadre de la programmation multimédia. Là aussi, la maîtrise des fondamentaux et de la syntaxe spécifiques à ce langage (par ailleurs très différents du PHP) est impérative pour bien saisir l'intérêt du langage PHP et comment et pourquoi les deux langages s'associent afin de produire du contenu multimédia dynamique (nda : voir le cours *Le langage HTML* du même auteur).

Enfin, veuillez noter que le contenu technique du présent document date du 2nd semestre 2004 [1]. L'essor fulgurant de la planète Internet ainsi que l'avancée inéluctable des technologies informatiques rendent une partie de ces informations obsolètes. Les éléments décrits demeurent fonctionnels, mais certains sont déconseillés dans le cadre d'activités à caractère professionnel, notamment par souci de sécurité et de respect des standards.

On veillera donc à compulser d'autres documents dédiés au langage PHP afin de parfaire ses connaissances quant aux us et coutumes du moment du bon développeur web : évolution du HTML 4 vers le XHTML [2], création de classes et d'objets en PHP5, etc.

---

[1] La mise en page peut être plus récente. La date spécifiée en page de garde correspond à la date de dernière modification du fond et/ou de la forme.

[2] XHTML : eXtensible HyperText Markup Language (eng) ≡ Langage de Description de l'HyperTexte Extensible (fr), successeur du HTML qui mélange les syntaxes du langage HTML et du langage XML nécessitant plus de rigueur dans son écriture.

1 INTRODUCTION AU LANGAGE PHP

Le langage **PHP** [1] est un langage de script open source, qui est interprété, et qui permet de réaliser de la programmation multimédia dynamique.

Les avantages de ce langage sont :

- intégration facile dans une page HTML ;
- syntaxe proche du langage C/C++ ;
- gestion simplifiée de multiples types de base de données.

PHP est un langage de script, et permet donc de réaliser des traitements. Ceux-ci sont réalisés au niveau de la machine serveur, c'est-à-dire l'ordinateur proposant l'accès à des pages web [2].

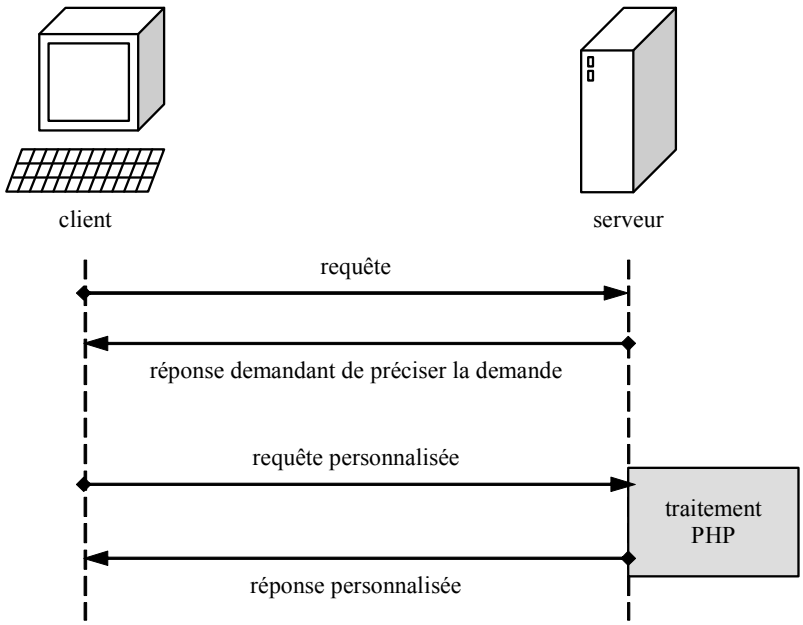


Figure 1.1 : communication client-serveur avec traitement PHP

Le langage PHP est un langage interprété, il n'a donc pas besoin d'être compilé ; en revanche, il est nécessaire de disposer d'un interpréteur PHP pour pouvoir exécuter des scripts PHP. Comme c'est la machine serveur qui se charge du traitement PHP, c'est elle qui doit donc disposer d'un interpréteur PHP [3].

Au contrario des langages de scripts qui s'exécutent au niveau du client, l'utilisation d'un langage de script serveur comme le PHP est totalement transparent pour la machine cliente, et donc il n'y a pas besoin de changer, ou de mettre à jour l'interpréteur HTML [4]. Cependant, comme le traitement est effectué par le serveur, qui est censé pouvoir répondre aux requêtes de plusieurs clients en même temps, celui-ci doit être suffisamment puissant pour pouvoir exécuter les différents traitements en parallèle. Il y a donc un risque de surcharge de traitements au niveau du serveur.

---

[1] PHP : Php is a Hypertext Preprocessor (eng) ≡ Php est un Préprocesseur Hypertexte (fr).
[2] Certains langages de script réalisent des traitements au niveau du client.
[3] Généralement, l'interpréteur PHP est constitué d'un module additionnel (plugin) à rajouter au serveur web.
[4] L'interpréteur HTML est le navigateur web.

2 NOTIONS DE BASE

2.1 GÉNÉRALITÉS

2.1.1 Écriture de code PHP

Généralement on écrit des scripts PHP en association avec le langage HTML. C'est-à-dire qu'un même fichier contiendra du code HTML ainsi que du code PHP [1].

Pour que l'interpréteur PHP puisse reconnaître du code PHP, on l'insère à l'intérieur de la balise `<? ?>`.

```
<?
   code PHP
?>
```

Il existe 2 syntaxes alternatives :

```
<?php                                  <script language="php">
   code PHP              OU               code PHP
?>                                     </script>
```

Cette syntaxe, utilisant une pseudo-balise, permet de mélanger très facilement du code HTML avec du code PHP. Là où, dans du code HTML, on veut insérer du code PHP, on utilise simplement la balise `<? ?>`. L'interpréteur PHP différencie ainsi facilement le code PHP (qu'il doit interpréter) du code HTML (qu'il n'a pas à interpréter).

Ex. :
```
<html>
   <head><title>PHP test</title></head>
   <body>
      Voici un test simple
      <? print(" de codage PHP"); ?>
   </body>
</html>
```

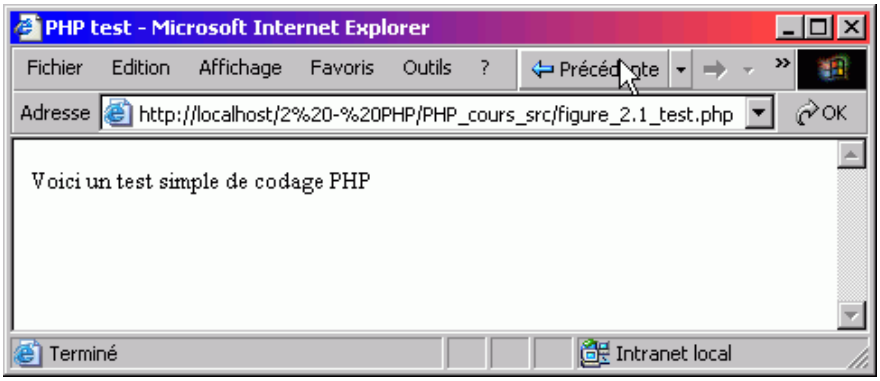


Figure 2.1 : exemple simple de codage PHP

Un fichier contenant du code PHP, même s'il est mélangé à du code HTML, doit avoir l'extension *.php* pour pouvoir être reconnu comme fichier PHP (les extensions *.php3*, *.php4* [2] et *.phtml* sont aussi possibles).

[1] Mais ce n'est pas obligatoire.
[2] .php3 ou .php4 permettent de préciser le numéro de version de PHP utilisé (version 3 ou 4) ; en septembre 2004, nous en sommes à PHP5.

Le langage PHP a une syntaxe très proche des langages C/C++ :

- toute ligne de code se termine par le caractère '`;`' (point-virgule) ;
- les niveaux d'imbrication de boucle sont indiquées avec les caractères '`{`' et '`}`' (accolades) ;
- les expressions sont évaluées en étant encadrées par les caractères '`(`' et '`)`' (parenthèses) ;
- les opérateurs (opération, affectation, comparaison) ont été repris du langage C ;
- les fonctions standard du langage C ont été recodées en PHP en conservant le même prototype [1] ;
- on insère des commentaires avec les caractères '`//`' (une seule ligne) ou bien `/* */` (plusieurs lignes).

2.1.2 Interprétation du code PHP

On a vu que c'était le serveur web qui avait en charge l'interprétation du code PHP. Le client, pour exploiter alors le résultat des traitements PHP, ne peut le faire que si celui-ci est fourni sous forme de code HTML, étant donné qu'il ne comprend pas la syntaxe PHP. Ce qui veut dire qu'un résultat issu d'un traitement PHP, doit être fourni sous la forme de code HTML, avec donc éventuellement des balises pour la mise en forme.

Le langage PHP opère ainsi ce qu'on appelle une **génération de page web**.

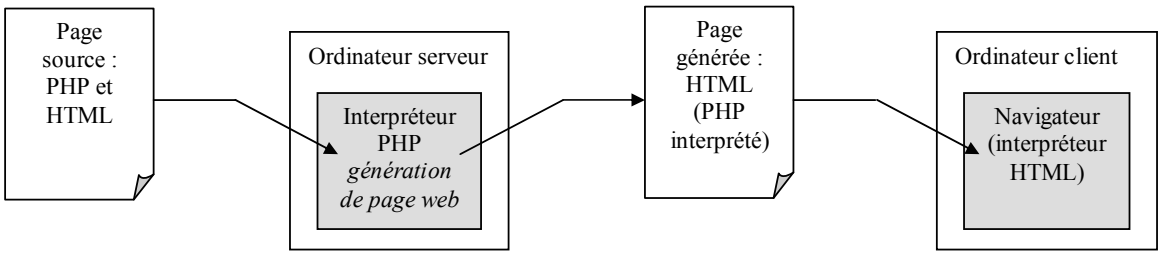


Figure 2.2 : principe de génération de page web

Ex. : Un site constitué de 2 pages, une première intégrant un formulaire simple avec un bouton de validation (*addition.htm*), la seconde affichant le résultat d'un traitement PHP (*addition_resultat.php*).

addition.htm

```
<html>
   <head><title>Addition</title></head>
   <body>
      <h1 align="center">Addition</h1>
      Combien font 4 + 2 ?<br>
      <form action="addition_resultat.php">
         Cliquez sur le bouton pour voir le résultat<br>
         <input type="submit" value="Cliquez-moi">
      </form>
   </body>
</html>
```

addition_resultat.php (source, vu par l'interpréteur PHP)

```
<html>
   <head><title>Addition - résultat</title></head>
   <body>
      <h1 align="center"> Addition - résultat </h1>
      4 + 2 font :<br>
      <?
         $res = 4 + 2;
         print("<b>$res</b>");
      ?>
      <br><hr>
   </body>
</html>
```

code PHP source

[1] Le prototype d'une fonction a pour caractéristiques : nom de la fonction, noms des paramètres en entrée : valeur_retour = nom_fonction(paramètre1, paramètre2, …).

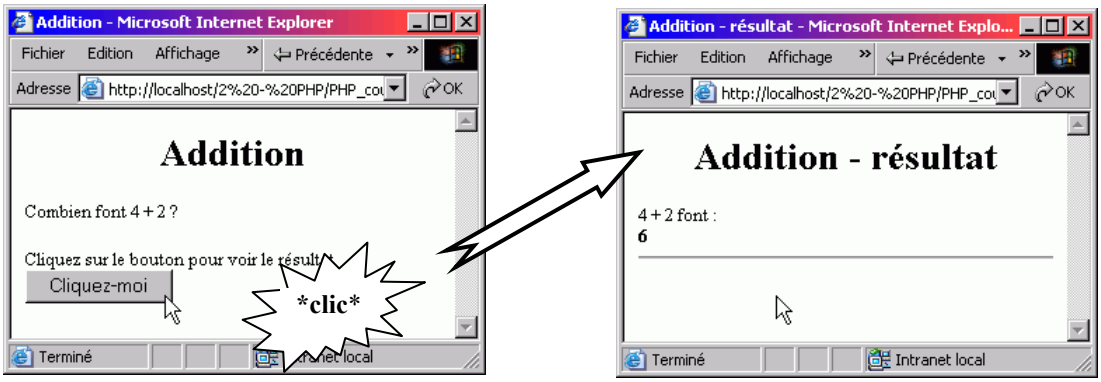


Figure 2.3 : exemple du traitement effectué par l'interpréteur PHP

Le fichier *addition_resultat.php* est le fichier source disponible sur le serveur web. Avant d'être envoyé au client, l'interpréteur PHP doit le traiter et le modifier afin qu'il ne contienne plus de code PHP, qui n'est pas compris par le navigateur. Voici donc le fichier *addition_resultat.php* tel qu'il sera vu par le client :

addition_resultat.php (interprété, vu par l'interpréteur HTML)

```
<html>
   <head><title>Addition - résultat</title></head>
   <body>
      <h1 align="center">Addition - résultat</h1>
      4 + 2 font :<br>
      <b>6</b>
      <br><hr>
   </body>
</html>
```

code PHP interprété

Ainsi, le traitement PHP est remplacé, dans le fichier envoyé au client, par le résultat des différents affichages réalisés lors de ce traitement. Ces affichages sont donc lus par l'interpréteur HTML comme toute autre ligne de code HTML ; il convient donc d'insérer des balises de mise en forme du texte dans ces affichages.

2.2 LES VARIABLES

Contrairement aux langages C/C++, il n'y a pas besoin de déclarer les variables. En revanche, afin de les reconnaître, leur nom commence toujours par le caractère '`$`' (dollar).

Comme la déclaration n'est pas nécessaire, une variable n'est donc pas typée : par exemple, une variable peut contenir un entier à sa première initialisation, puis, par affectation, contenir ensuite une chaîne.

2.2.1 Les types de variables

Les types disponibles en PHP sont classiques :

- entier : nombre entier, ex. : `$N = 15;` ;
- réel : nombre réel (flottant), ex. : `$X = -4.43;` ;
- chaîne : chaîne de caractère, ex. : `$nom = "Jean";` ;
- tableau : liste de valeur, ex. : `$tab = array(12,4,2);` ;
- objet : objet (au sens programmation orientée objet).

2.2.2 Les variables numériques

Les opérateurs pour les variables numériques sont les opérateurs classiques :

- opérateurs arithmétiques : `+`, `-` , `*`, `/`, `%` ;
- opérateurs d'incrémentation/décrémentation (pré– et post–) : `++`, `--` ;
- opérateurs combinés : `+=`, `-=`, `*=`, `/=`, `%=`, `&=`, `|=`, `^=`, `<<=`, `>>=` ;
- opérateurs logiques : `&` (ET), `|` (OU), `^` (XOR), << (décalage à gauche), >> (décalage à droite).

On peut afficher la valeur de la variable numérique directement grâce à l'instruction `print()` avec la syntaxe `print(chaîne);`.

Ex. :
```
$nb = 24;
print("il y a $nb élèves présents"); // il y a 24 élèves présents
```

On peut aussi afficher la valeur, mais de manière formatée (comme en langage C).

Ex. :
```
$nb = 10/3;
printf("10 divisé par 3 fait %1.2f", $nb); // 10 divisé par 3 fait 3.33
```

Ce formatage peut être appliqué directement à la variable, en modifiant ainsi sa valeur, mais sans l'afficher.

Ex. :
```
$nb = 9/7;
$nb = sprintf("%1.3f", $nb);
print("9 divisé par 7 fait $nb"); // 9 divisé par 7 fait 1.286
```

2.2.3 Les chaînes de caractères

Les chaînes de caractères doivent être encadrées par le caractère '`"`' (guillemets).

On peut concaténer les chaînes facilement avec l'opérateur '`.`' (point).

Ex. :
```
$nom = "Dupont";
$prenom = "Jean";
$id = "Nom: " . $nom . " – Prénom: " . $prenom;
print("$id"); // Nom: Dupont – Prénom: Jean
```

Il existe les caractères spéciaux connus du langage C : `\r` (retour chariot), `\n` (saut de ligne), `\t` (tabulation), `\b` (retour arrière), `\f` (saut de page).

Le caractère `\` est le caractère d'échappement [1] et permet par exemple d'insérer le caractère '`"`' (guillemets) dans une chaîne.

Ex. :
```
print("en écrivant \\\", le caractère guillemets : \" s'affiche");
```

2.2.4 Les tableaux

Comme toute autre variable PHP, le contenu d'un tableau n'est pas typé ; celui-ci peut contenir une liste de valeurs hétérogènes, soit donc de plusieurs types différents.

Ex. :
```
$tab = array(34, "jean", -76.2);
```

Il existe deux sortes de tableaux :

- tableau simple : l'index est un entier (identique au langage C), le 1er élément commence à l'indice 0 ;
  Ex. :
  ```
  $tab = array("Jean", 20);
  print("$tab[0] a $tab[1] ans"); // Jean a 20 ans
  ```
- tableau associatif : l'index est une chaîne.
  Ex. :
  ```
  $age["Paul"] = 21;
  print("Paul a $age[Paul] ans"); // Paul a 21 ans
  ```

On peut définir des tableaux à plusieurs dimensions.

Ex. : Tableau à 2 dimensions.

```
$tab = array (
     array("Jean", "Dupont"),
     array("Paul", "Durand")
  );

print("le nom de {$tab[0][0]} est {$tab[0][1]}"); // le nom de Jean est Dupont
print("le nom de {$tab[1][0]} est {$tab[1][1]}"); // le nom de Michel est Durand
```

[1] Appelé *quote* en anglais (to quote (eng) ≡ citer (fr)).

Notez l'utilisation des accolades dans le cas d'affichage de valeurs de tableaux multi-dimensions ; cela permet de remplacer par sa valeur l'élément du tableau pointé par les indices.

On peut définir explicitement un indice à une valeur en utilisant la syntaxe `indice => valeur`. Cela peut être plus pratique dans le cas de tableaux associatifs, ainsi que pour les tableaux simples afin d'affecter le 1er indice à 1 plutôt qu'à 0.

Ex. :
```
$tab = array("Paul", "Jean"); // idem $tab = array(0 => "Paul", 1 => "Jean");
$tab = array(1 => "Paul",2 => "Jean");
print("$tab[1]"); // Paul

$age = array("Jean" => 20,"Paul" => 21);
print("Jean a $age[Jean] ans"); // Jean a 20 ans
```

2.3 LES STRUCTURES DE CONTRÔLE

Elles sont identiques au langage C.

2.3.1 Les opérateurs

- opérateurs de tests : ==, !=, >, <, >=, <= ;
- opérateurs logiques : `AND` ou `&&`, `OR` ou `||`, `NOT` ou `!`, `XOR` ou `^`.

2.3.2 Les tests

2.3.2.1 Test simple

On réalise un test simple avec l'instruction `if ()` éventuellement associée avec l'instruction `else`, suivant la syntaxe :
```
if (condition) {
   /* instructions si condition validée */
}
else {
   /* instructions si condition invalidée */
}
```

2.3.2.2 Test multiple

On réalise un test multiple avec les instructions `switch` et `case` éventuellement associées avec l'instruction `default`, suivant la syntaxe :
```
switch ($variable) {
   case valeur1 :  /* instructions si $variable vaut valeur1 */
           break
   case valeur2 :  /* instructions si $variable vaut valeur2 */
           break
   ...
   default :  /* instructions si $variable ne vaut aucune des valeurs */
}
```

2.3.3 Les boucles

2.3.3.1 Boucle « tant que… faire… »

On réalise une boucle « tant que » (répétition avec aucune exécution au minimum) avec l'instruction `while ()`, suivant la syntaxe :
```
while (condition) {
   /* instructions */
}
```

2.3.3.2 Boucle « répéter… tant que »

On réalise une boucle « répéter tant que » (répétition avec aucune exécution au minimum) avec les instructions `do` et `while ()`, suivant la syntaxe :

```
do {
   /* instructions */
}
while (condition);
```

2.3.3.3 Boucle « pour… faire »

On réalise une boucle « pour » (répétition contrôlée) avec l'instruction `for ()`, suivant la syntaxe :

```
for (initialisation ; condition ; incrémentation ou décrémentation) {
   /* instructions */
}
```

3 LES FONCTIONS

3.1 FONCTIONS STANDARD

Le langage PHP comprend de nombreuses bibliothèques de fonctions standard, dont certaines ont été reprises du langage C. Voici une liste non exhaustive de quelques-unes d'entre elles.

3.1.1 Fonctions numériques

- `round(nb, dec)` : arrondit le nombre `nb` a 10^{-dec} près ;
- `ceil(nb)` : arrondit le nombre `nb` à l'entier supérieur ;
- `floor(nb)` : arrondit le nombre `nb` à l'entier inférieur ;
- `abs(nb)` : donne la valeur absolue du nombre `nb` ;
- `srand(nb)` : initialise le générateur de nombres aléatoires (on utilise `(double) microtime()*1000000` comme valeur d'initialisation généralement) ;
- `rand(min, max)` : renvoie un nombre aléatoire entier compris entre les nombres `min` et `max` (toujours avoir appelé le générateur de nombres aléatoires au préalable).

3.1.2 Fonctions de chaînes

- `trim(chn)` : enlève les espaces au début et/ou à la fin de la chaîne `chn` ;
- `urlencode(chn)` : convertit la chaîne de caractères `chn` en chaîne url-encodée (conversion des espaces, accents, caractères spéciaux, etc.) ;
- `urldecode(chn)` : convertit la chaîne url-encodée `chn` en chaîne de caractères ;
- `crypt(chn)` : crypte la chaîne `chn` (attention, l'opération inverse est impossible) ;
- `strtok(chn, car)` : extrait la sous-chaîne de la chaîne `chn` jusqu'au caractère `car` (non compris) ;
- `substr(chn, dep, lng)` : extrait la sous-chaîne de la chaîne `chn` longue de `lng` caractères à partir du `dep`-ième caractère depuis le début (si `dep` est positif) ou depuis la fin (si `dep` est négatif), (le 1er caractère a l'indice *0*, comme en langage C) ;
- `strlen(chn)` : donne la longueur de la chaîne `chn`.

3.1.3 Fonctions de tableaux

- `count(tab)` : donne le nombre d'éléments du tableau `tab` ;
- `array_merge(tab1, tab2)` : concatène les tableaux `tab1` et `tab2` en un seul autre tableau ;
- `each(tab)` : parcourt le tableau `tab` en renvoyant un tableau de 2 éléments, le 1er étant l'indice (indice *0* ou *key*), et le second la valeur (indice *1* ou *value*) (à utiliser dans une boucle) ;
- `sort(tab)` : trie les valeurs du tableau `tab` uniquement suivant les valeurs (ordre numérique pour des nombres, ordre alphabétique pour des chaînes) ;
- `rsort(tab)` : trie les valeurs du tableau `tab` uniquement suivant les valeurs inverses (ordre numérique inversé pour des nombres, ordre alphabétique inversé pour des chaînes) ;
- `asort(tab)` : trie les couples (indice, valeur) du tableau `tab` suivant les valeurs ;
- `arsort(tab)` : trie les couples (indice, valeur) du tableau `tab` suivant les valeurs inverses ;
- `ksort(tab)` : trie les couples (indice, valeur) du tableau `tab` suivant les indices ;
- `krsort(tab)` : trie les couples (indice, valeur) du tableau `tab` suivant les indices inverses ;
- `shuffle(tab)` : trie le tableau `tab` de manière aléatoire ;
- `reset(tab)` : replace le tableau `tab` dans l'ordre dans lequel il se trouvait avant le dernier tri effectué ;

- `implode(sep, tab)` : renvoie la chaîne de caractères correspondant à la concaténation de tous les éléments du tableau `tab` et en les séparant par le motif `sep` ;
- `explode(sep, chn)` : renvoie un tableau en découpant la chaîne `chn` suivant le motif `sep`.

3.2 FONCTIONS UTILISATEUR

En PHP, on peut écrire ses propres fonctions en utilisant le mot-clef `function` suivant la syntaxe suivante :

```
function nom_fonction($paramètre1, $paramètre2)
{
   ...
   return ($valeur_retour);
}
...
$res = nom_fonction($aa, $bb);
```

Nb : Une fonction peut ne renvoyer aucune valeur de retour, tout comme elle peut ne pas avoir de paramètres d'en entrée.

Ex. :
```
<?
      function addition($nb1, $nb2)
      {
         $val = $nb1 + $nb2;
         return ($val);
      }

      $nombre = 6;
      $res = addition($nombre, 5);
      print("le résultat est $res"); // le résultat est 11
   ?>
```

Les variables sont locales, sauf si déclarées avec le mot-clef `global`.

On peut définir des valeurs par défaut pour les arguments d'une fonction [1].

Ex. :
```
<?
      function addition($nb1, $nb2=4)
      {
         return($nb1 + $nb2);
      }
   ?>
```

On peut alors écrire :

```
$res = addition($nombre); // idem $res = addition($nombre, 4);
print("le résultat est $res"); // le résultat est 10
```

[1] Cette syntaxe peut s'apparenter au principe de polymorphisme des méthodes en programmation orientée objet.

4 GESTION DES BASES DE DONNÉES

4.1 INTRODUCTION AUX BASES DE DONNÉES

4.1.1 Principes et structure d'une base de données

Une base de données est un objet informatique qui permet de stocker et gérer de manière optimale un ensemble d'éléments d'informations en les structurant.

L'intérêt des bases de données est de permettre d'effectuer assez simplement des dénombrements, des statistiques et/ou des recherches à l'intérieur des données stockées, suivant des critères précis.

Ex. : Sans base de données : informations brutes.

> Jean, qui est un jeune homme de 20 ans, habite à Montpellier, dans l'Hérault.
> En Haute-Garonne, à Toulouse, habite Nathalie qui est une jeune fille de 18 ans.
> C'est à Toulouse, en Haute-Garonne, que vit Sonia, adolescente de 16 ans.
> Michel vit à Montpellier, ville de l'Hérault ; c'est un homme de 27 ans.
> Paul, adolescent de 14 ans, habite le département de la Haute-Garonne, à Toulouse.

Avec base de données : reformulation et restructuration des informations brutes en ne gardant que les données pertinentes.

Table *infos* :

prenom	age	sexe	ville	departement
Jean	20	M	Montpellier	Hérault
Nathalie	18	F	Toulouse	Haute-Garonne
Sonia	19	F	Toulouse	Haute-Garonne
Michel	27	M	Montpellier	Hérault
Paul	14	M	Toulouse	Haute-Garonne

Dans la terminologie des bases de données, chaque ligne (ex. : Jean – 20 – M – Montpellier – Hérault) est appelée **enregistrement**, et chaque colonne (*prenom*, a*ge*, *sexe*, *ville*, *departement*) est appelée **champ**.

En général, l'un de ces champs [1] permet d'identifier de manière unique chacun des enregistrements et ainsi éviter les doublons ; ce champ est alors appelé **clef primaire**, ne peut être vide [2] et ne peut avoir la même valeur pour 2 enregistrements différents.

Ex. : Dans la table *infos*, la clef primaire est l'information du prénom contenue dans le champ *prenom*.

L'ensemble des champs constitue ce qu'on appelle une **table** et se représente généralement sous forme de tableau à 2 dimensions [3]. Une **base de données** est un ensemble constitué de 1 ou plusieurs tables.

Si un enregistrement ne peut être identifié de manière unique grâce à la clef primaire, on utilise un second champ ; ce champ est alors appelé **clef secondaire**, ne peut être vide et ne peut avoir la même valeur pour 2 enregistrements différents ayant même clef primaire.

[1] 1 ou plus.
[2] Un champ autre qu'une clef primaire peut être vide.
[3] Cette représentation bidimensionnelle est uniquement visuelle (adaptée à l'humain) et ne peut être assimilée à l'objet informatique correspondant. Autrement dit, une table n'est pas stockée informatiquement sous forme de tableau bidimensionnel.

La clef secondaire permet de distinguer les enregistrements en cas de doublon sur la clef primaire ; le couple (clef primaire / clef secondaire) permet alors d'identifier assurément de manière unique les enregistrements d'une même table.

Ex. : Dans la table *infos*, la clef primaire fait référence au prénom. Si on accepte de traiter les cas où des personnes différentes ont le même prénom, il faut donc envisager d'utiliser une clef secondaire pour les différencier. Cela pourrait être par exemple l'information d'âge contenue dans le champ *age*.

En général, si une base comprend plusieurs tables, c'est pour minimiser la taille occupée par celle-ci, ainsi que d'optimiser la gestion ; les différentes tables possèdent alors des champs communs afin de permettre de recouper les informations stockées.

Ex. : La table précédente peut être scindée en deux tables avec un champ commun (*ville*).

Table *etatcivil* :

prenom	age	sexe	ville
Jean	20	M	Montpellier
Nathalie	18	F	Toulouse
Sonia	19	F	Toulouse
Michel	27	M	Montpellier
Paul	14	M	Toulouse

Table *depts* :

ville	departement
Montpellier	Hérault
Toulouse	Haute-Garonne

4.1.2 Exploitation d'une base de données

Utiliser une base de données permet d'effectuer des dénombrements, des statistiques, des recherches au sein des données stockées selon des critères.

Ex. : Savoir qui habite Toulouse ;
Savoir quels sont les hommes de plus de 19 ans ;
Savoir quelles femmes, et quel est leur âge, qui habitent Montpellier ;
etc.

Comme il est difficile de répondre aux questions par une simple observation des données des tables, une base de données peut être manipulée et interrogée [1] directement en lui soumettant les questions, que l'on formule en utilisant le langage SQL. Une question, mise en forme par le langage SQL, est appelée **requête**.

Le rôle du gestionnaire de base de données (SGBD [2]), outre l'optimisation du stockage des informations ainsi que des accès en lecture et écriture au sein des bases, est de proposer le support du langage SQL afin d'interroger une base de données.

Il existe différents SGBD (Access (suite Microsoft Office), Oracle, SQL Server, PostgreSQL, MySQL, HSQL, etc.), parmi lesquels, on retiendra MySQL car il est gratuit [3], standard et supporté de manière native par le langage PHP [4].

4.1.3 Le langage SQL

Le langage SQL [5] permet de communiquer avec une base de données afin de la gérer ou de l'interroger. Il s'agit d'un langage déclaratif à la syntaxe très simple, et qui figure parmi les plus utilisés pour l'accès aux bases de données.

SQL permet l'interaction avec le serveur et les informations qu'il héberge en soumettant une commande au SGBD sous la forme d'une requête en suivant la syntaxe :

```
commandeSQL [mot-clef1] paramètre1 [mot-clef2 paramètre2 [...]]
```

[1] On parle parfois d'*attaquer une base* lorsque l'on désire l'interroger.
[2] SGBD : Système de Gestion de Base de Données – logiciel gérant les bases de données et permettant de les interroger en utilisant le langage SQL (on parle aussi de SGBDR, Système de Gestion de Base de Données Relationnelles).
[3] Dans une utilisation personnelle ou éducative.
[4] Mais PHP intègre aussi le support pour beaucoup d'autres SGBD.
[5] SQL : Structured Query Language (eng) ≡ Langage de Requête Structurée (fr).

Un paramètre peut désigner le nom d'une base, d'une table, d'un champ, une valeur, un critère, etc. Lorsqu'il s'agit d'une valeur alphanumérique, on les encadre par des apostrophes [1] (ex. : `'chaine'`).

4.2 COMMANDES D'ACCÈS À LA BASE

4.2.1 Commandes principales

Le langage comprend 4 commandes principales [2] :

- `SELECT` : parcourt la base et lit des enregistrements ;
- `INSERT` : insère un nouvel enregistrement ;
- `UPDATE` : modifie un enregistrement existant ;
- `DELETE` : efface un enregistrement existant.

4.2.1.1 La commande Select

Pour interroger la base de données, on utilise la commande `SELECT` suivant la syntaxe :

```
SELECT champAExtraire1 [, champAExtraire2 [, ...]]
FROM nomTable
WHERE criteres
```

Le ou les critère(s) de recherche sont définis par la clause `WHERE criteres` suivant la syntaxe `WHERE critere1 [operateurLogique critere2 [...]]` où un critère suit la syntaxe `champCritere operateur valeur`.

La clause `WHERE` peut être omise ; en ce cas les champs indiqués de tous les enregistrements sont lus.
Si on veut lire tous les champs correspondant au(x) critère(s), on peut écrire `*` à la place de la liste des champs.

Ex. : Savoir qui habite Toulouse.

```
SELECT prenom FROM infos WHERE ville='Toulouse'
```

Savoir quel âge ont les hommes de plus de 19 ans.

```
SELECT age FROM infos WHERE sexe='M' AND age>19
```

Savoir quelles femmes, et quel est leur âge, qui habitent Montpellier.

```
SELECT prenom,age FROM infos WHERE sexe='F' AND ville='Montpellier'
```

Connaître toutes les informations concernant les personnes majeures.

```
SELECT * FROM infos WHERE age>=18
```

Connaître tous les prénoms des personnes (sans restriction).

```
SELECT prenom FROM infos
```

Connaître toutes les informations concernant toutes les personnes (soit donc l'intégralité de la table).

```
SELECT * FROM infos
```

Les résultats obtenus correspondent à une portion de la table, que l'on appelle alors **sous-table** ; il s'agit donc d'un tableau à 2 dimensions (même dans le cas où une seule colonne est extraite).

Ex. : Savoir qui habite Toulouse

```
SELECT prenom FROM infos WHERE ville='Toulouse'
```

prenom	age	sexe	ville	departement
Jean	20	M	Montpellier	Hérault
Nathalie	18	F	Toulouse	Haute-Garonne
Sonia	19	F	Toulouse	Haute-Garonne
Michel	27	M	Montpellier	Hérault
Paul	14	M	Toulouse	Haute-Garonne

La sous-table résultante est donc :

Nathalie
Sonia
Paul

[1] Appelées aussi *simple cotes*, en opposition aux guillemets (doubles cotes).
[2] Les commandes SQL sont généralement écrites en majuscules ; mais cela n'est pas une obligation, car SQL ne différencie pas la casse.

4.2.1.2 La commande Insert

Pour insérer un nouvel enregistrement dans la base de données, on utilise la commande `INSERT` suivant la syntaxe :

```
INSERT INTO nomTable
VALUES (valeurChamp1, valeurChamp2, ...)
```

Ex. : `INSERT INTO infos VALUES ('Jean', 20, 'M', 'Montpellier', 'Hérault')`

Il faut insérer autant de valeurs qu'il y a de champs dans la table, sinon il faut indiquer les champs que l'on remplit à la suite du nom de la table suivant la syntaxe :

```
INSERT INTO nomTable (nomChamp1 [, nomChamp2 [, ...])
VALUES (valeurChamp1 [, valeurChamp2 [, ...]])
```

Ex. : `INSERT INTO infos (prenom, age, ville) VALUES ('Jean', 20, 'Montpellier')`

4.2.1.3 La commande Delete

Pour effacer un enregistrement, on utilise la commande `DELETE` suivant la syntaxe :

```
DELETE FROM nomTable
WHERE criteres
```

Ex. : `DELETE FROM infos WHERE prenom='Nathalie'`

4.2.1.4 La commande Update

Pour mettre à jour un enregistrement, on utilise la commande `UPDATE` suivant la syntaxe :

```
UPDATE nomTable
SET champAModifier1=nouvelleValeur1 [, champAModifier2=nouvelleValeur2 [, ...]]
WHERE criteres
```

Ex. : `UPDATE infos SET age=22 WHERE prenom='Jean'`

4.2.2 Commandes de gestion de base

4.2.2.1 La commande Create

La commande `CREATE` permet de créer une base suivant la syntaxe `CREATE DATABASE nomBase`.

Elle permet également de créer une table en définissant le nom des champs ainsi que le type de chacun d'entre eux suivant la syntaxe :

```
CREATE TABLE nomTable (nomChamp1 type_champ1 [, nomChamp2 type_champ2 [, ...]]).
```

Les différents types de champs possibles sont : `CHAR`, `VARCHAR`, `INT`, `FLOAT`, `BOOLEAN`, `DATETIME`, etc.

Ex. :
```
CREATE DATABASE cours_php
CREATE TABLE etatcivil
   (prenom VARCHAR(50), age INT, sexe CHAR(1), ville VARCHAR(50))
CREATE TABLE depts (ville VARCHAR(50), departement INT)
```

4.2.2.2 La commande Drop

La commande `DROP`, utilisée avec la syntaxe `DROP DATABASE nomBase`, permet d'effacer une base ; utilisée avec la syntaxe `DROP TABLE nomTable`, elle permet d'effacer une table.

Ex. :
```
DROP TABLE etatcivil
DROP DATABASE cours_php
```

4.3 GESTION EN PHP

4.3.1 Connexion à une base de données

Pour pouvoir gérer une base de données, il faut se connecter à celle-ci, c'est-à-dire définir un objet informatique (référence) du langage PHP qui représente la base de données dans laquelle on veut lire et/ou écrire.

On utilise pour cela l'instruction `mysql_connect()` selon la syntaxe suivante : `référence = mysql_connect(serveur_bdd, login, mot_de_passe);`

Ex. : `$bddnoms = mysql_connect("serveurbdd", "root", "abracadabra");`

Le serveur de base de données correspond à la machine sur lequel le SGBD est installé (une machine du réseau local généralement) ; c'est sur cette machine que sont stockées les bases.

Le login et le mot de passe permettent d'accéder à la base de données selon des droits spécifiques (typiquement, accès lecture/écriture ou lecture seule).

4.3.2 Création d'une base de données

On crée une base de données avec l'instruction `mysql_query()` selon la syntaxe suivante : `mysql_query(requête);` où `requête` correspond à une requête SQL créant une base de données [1].

Ex. : `mysql_query("CREATE DATABASE cours_php");`

4.3.3 Fermeture d'une base de données

Après utilisation, il faut fermer l'accès à la base de données avec l'instruction `mysql_close()` selon la syntaxe suivante : `mysql_close(référence);`.

Ex. : `mysql_close($bddnoms);`

4.3.4 Écriture dans une base de données

Une fois la connexion établie (et la base créée si besoin est), on écrit directement dans la base grâce à des requêtes écrites en langage SQL en utilisant pour cela l'instruction `mysql_db_query()` selon la syntaxe suivante : `mysql_db_query(base, requête, référence);`.

Ex. :
```
$rqt = "CREATE TABLE etatcivil
   (prenom VARCHAR(50), age INT, sexe CHAR(1), ville VARCHAR(50))";
mysql_db_query("cours_php", $rqt, $bddnoms);

$p = array("Jean", 20, "M", "Montpellier");
$rqt = "INSERT INTO etatcivil VALUES ('$p[0]', '$p[1]', '$p[2]', '$p[3]')";
mysql_db_query("cours_php", $rqt, $bddnoms);
```

4.3.5 Lecture dans une base de données

Une fois la connexion établie, on lit directement dans la base grâce à des requêtes écrites en SQL et en utilisant là encore l'instruction `mysql_db_query()` mais cette fois-ci avec la syntaxe suivante : `idrésultats = mysql_db_query(base, requête, référence);`.

Ex. : Savoir qui habite Toulouse, et leur âge
```
SELECT prenom,age FROM etatcivil WHERE ville='Toulouse'

$rqt = "SELECT prenom,age FROM etatcivil WHERE ville='Toulouse'";
$idres = mysql_db_query("cours_php", $rqt, $bddnoms);
```

Le résultat obtenu est une sous-table :

Nathalie	18
Sonia	16
Paul	14

Pour extraire chacune des lignes de cette sous-table, on utilise l'instruction `mysql_fetch_array()` suivant la syntaxe : `ligne = mysql_fetch_array(idrésultats);`.

[1] Sans écrire de requête SQL, il existe aussi la fonction `mysql_create_db(base,référence)`, mais celle-ci est maintenant obsolète.

Ex. :
```
$rqt = "SELECT prenom,age FROM etatcivil WHERE ville='Toulouse'";
$idres = mysql_db_query("cours_php", $rqt, $bddnoms);

print("<table border=1>\n");
print("<tr>\n");
print("<th>prénom</th>\n");
print("<th>âge</th>\n");
print("</tr>\n");

while ($line = mysql_fetch_array($idres)) {
   print("<tr>\n");
   print("<td>$line[0]</td>\n");
   print("<td>$line[1]</td>\n");
   print("</tr>\n");
}

print("</table>\n");
```

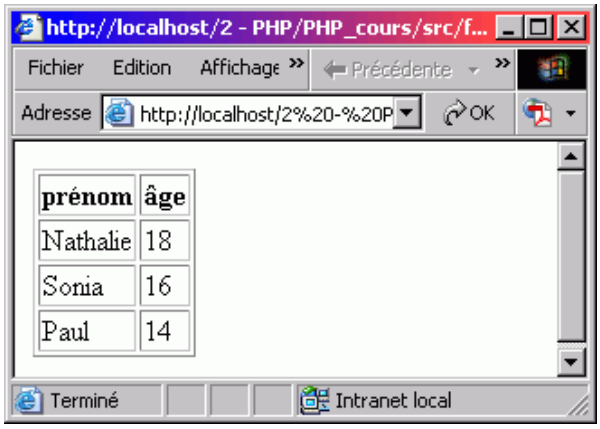


Figure 4.1 : exemple de lecture dans une base de données et de traitement des résultats

4.3.6 Gestion des erreurs

Toutes les instructions énumérées ci-dessus permettent la gestion d'erreurs. En effet, il suffit de lire le code de retour de l'instruction (même si celle-ci semble ne pas en renvoyer un). Il vaut `false` si l'instruction ne s'est pas déroulée correctement ; dans le cas contraire, il vaut la valeur renvoyée par l'instruction ou bien `true` si l'instruction n'en renvoie pas.

Ex. : Code complet.

```
// connexion à la base
if (!$bddnoms = mysql_connect("serveurbdd", "user", "motdepasse"))
   print("erreur connexion");
//else print("connexion réussie");

// création de la base
$rqt = "CREATE DATABASE cours_php"; // ceci ne doit être réalisé qu'une seule fois
if (!mysql_query($rqt))
   print("erreur création base");
//else print("création base réussie");

// création de la table
$rqt = "CREATE TABLE etatcivil
   (prenom VARCHAR(50), age INT, sexe VARCHAR(1), ville VARCHAR(50))";
if (!mysql_db_query("cours_php", $rqt, $bddnoms))
   print("erreur création table");
//else print("création table réussie");
```

```
// remplissage de la table
$tab = array(
   array("Jean", 20, "M", "Montpellier"),
   array("Nathalie", 18, "F", "Toulouse"),
   array("Sonia", 16, "F", "Toulouse"),
   array("Michel", 27, "M", "Montpellier"),
   array("Paul", 14, "M", "Toulouse")
);

for ($i=0 ; $i<5 ; $i++) {
   $rqt = "INSERT INTO etatcivil VALUES
      ({$tab[$i][0]}', '{$tab[$i][1]}', '{$tab[$i][2]}', '{$tab[$i][3]}')";
   if (!mysql_db_query("cours_php", $rqt, $bddnoms))
      print("erreur écriture");
   //else print("écriture réussie");
}

// extraction de la table
$rqt = "SELECT prenom,age FROM etatcivil WHERE ville='Toulouse'";
if (!$idres = mysql_db_query("cours_php", $rqt, $bddnoms))
   print("erreur lecture");
else {
   print("<table border=1>\n");
   print("<tr>\n");
   print("<th>prénom</th>\n");
   print("<th>âge</th>\n");
   print("</tr>\n");

   while ($line = mysql_fetch_array($idres)) {
      print("<tr>\n");
      print("<td>$line[0]</td>\n");
      print("<td>$line[1]</td>\n");
      print("</tr>\n");
   }

   print("</table>\n");
}

if (!mysql_close($bddnoms))
   print("erreur fermeture connexion");
//else print("fermeture connexion réussie");
```

5 GESTION DES FICHIERS

La gestion des fichiers en PHP est calquée sur celle déjà existante en langage C.

5.1 GESTION DE L'ACCÈS

5.1.1 Pointeur sur un fichier et ouverture du fichier

Avant toute utilisation d'un fichier, il faut définir un objet informatique qui symbolisera le fichier, et permettra de le manipuler. Cet objet est appelé **pointeur** de fichier ou **flux** sur fichier.

On définit un pointeur avec l'instruction `fopen()`, suivant la syntaxe : `pointeur = fopen(nom_fichier, mode_ouverture);`.

Ex. : `$fichier1 = fopen("test.txt", "r+");`

Les différents modes d'ouverture sont :

symbole	accès	remarques
`r`	lecture seule	
`w`	écriture seule	(1) (2)
`a`	écriture	(1) (3)
`r+`	lecture et écriture	
`w+`	lecture et écriture	(1) (2)
`a+`	lecture et écriture	(1) (3)

(1) : crée le fichier s'il n'existe pas
(2) : si le fichier existe, le contenu est écrasé
(3) : si le fichier existe, on écrit à la suite

La création du pointeur « ouvre » en même temps le fichier.

5.1.2 Fermeture du fichier

Tout fichier ouvert doit être fermé après usage en utilisant l'instruction `fclose()` avec la syntaxe : `fclose(pointeur);`.

Ex. : `fclose($fichier1);`

5.2 LECTURE ET ÉCRITURE

5.2.1 Écriture dans un fichier

Pour écrire dans un fichier, on utilise l'instruction `fwrite()`, avec la syntaxe : `fwrite(pointeur, données);`.

Ex. : `fwrite($fichier1, "ceci est un test");`

Il existe d'autres fonctions [1] d'écriture dans un fichier : `fputs()`, `putc()`, etc.

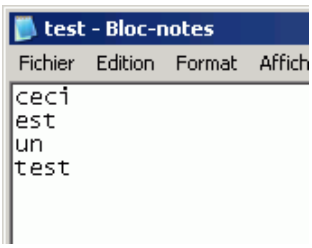
[1] Reprises du langage C.

5.2.2 Lecture dans un fichier

Pour lire dans un fichier, on utilise l'instruction `file()` avec la syntaxe : `tableau = file(pointeur);`. Chaque élément du fichier séparé par le caractère <*entrée*> sera un élément différent du tableau.

Ex. : `$tableau1 = file($fichier1);`

Si le fichier pointé par `$fichier1` contient le texte *ceci*↵*est*↵*un*↵*test*, le tableau résultant contiendra donc 4 éléments : *ceci*, *est*, *un* et *test*.



$tableau1
ceci
est
un
test

Figure 5.1 : exemple de lecture de fichier

Il existe d'autres fonctions [1] de lecture dans un fichier : `fgets()`, `fgetc()`, `fread()`, etc.

---

[1] Là aussi, héritées du langage C.

A BIBLIOGRAPHIE

Wazir Dino, *Cours PHP*, TS IRIS – LEGT Louis-Modeste Leroy – Évreux, 2002 ;

CommentÇaMarche.net, http://www.commentcamarche.net/, 2004 ;

Wikipedia – l'encyclopédie libre, http://fr.wikipedia.org/, 2005 ;

Peignot Christophe, *Langage HTML*, *Langage C* et *Langage SQL*, http://info.arqendra.net/, 2010.