Les tableaux de données

Sommaire

Les repères de lecture


Retour au chapitre


Définition


Objectif(s)


Espace Élèves


Vidéo / Audio


Point important / À retenir


Remarque(s)


Pour aller plus loin


Normes et lois


Quiz

I. Principe et fonctionnement

Un **tableau de données** permet de stocker des **valeurs** et de les ranger de façon ordonnée en les associant à des **clefs uniques** (que l'on nomme parfois des indices ou des index).

Créons, par exemple, le tableau suivant qui liste les saisons et attribue un numéro à chaque saison.

Tableau n°1

NUMÉROS	SAISONS
0	Été
1	Automne
2	Hiver
3	Printemps

Il est aisé de relier par exemple l'automne au numéro 1 et le printemps au numéro 3 car le rôle du tableau est justement d'associer les saisons aux numéros.

Il suffit de considérer que les saisons sont les valeurs et que les numéros sont les clefs pour avoir compris le principe du tableau de données.

Le tableau de données est parfois appelé un *array* car c'est la fonction *array* qui permet de créer ces tableaux.

Examinons la syntaxe en PHP en reprenant l'exemple du tableau précédent. Nous allons donc associer chaque saison à sa clef avec la fonction *array* comme ceci :

```
$saisons = array(0 => 'Été', 1 => 'Automne', 2 => 'Hiver', 3 =>'Printemps');
```

Fig. 1

Nous pouvons constater en premier lieu l'emploi de la flèche qui permet l'association clef/valeur. La flèche joue ici le rôle d'opérateur d'association. La flèche s'écrit simplement avec le signe = égal suivi du signe > supérieur à.

Puisque la flèche permet l'association clef/valeur nous pouvons ainsi relier 0 à Été grâce à l'instruction 0 => 'Été'.

De la même façon, nous relions 1 à Automne avec l'instruction 1 => 'Automne', 2 à Hiver avec l'instruction 2 => 'Hiver' et 3 à Printemps avec l'instruction 3 => 'Printemps'.

Après chacune de ces instructions, sauf la dernière, nous ajoutons une virgule pour bien les séparer.

Et bien sûr nous n'oublions pas d'entourer les noms de saisons avec des *quotes* puisqu'il s'agit de chaîne de caractères. Nous aurons également remarqué les parenthèses après l'instruction *array* qui entourent les quatre instructions.

Un point très important est de bien voir que nous affectons ce tableau à la variable *$saisons* et que donc la variable *$saisons* devient un tableau de données. Les tableaux de données sont donc des variables. De ce fait nous allons donc pouvoir utiliser ce tableau de données, et par exemple afficher une valeur du tableau ou bien toutes les valeurs.

Notons également que, en vertu de la pertinence de nommage des variables, nous avons mis au pluriel le nom de la variable *$saisons* pour la raison simple que cette variable contient plusieurs saisons.

A. Affichage d'une valeur du tableau

L'affichage d'une valeur du tableau ou de toutes les valeurs du tableau ne s'effectue pas directement avec la fonction *echo*, car un tableau n'est pas une chaîne de caractères ou un nombre. Un tableau est certes une variable, mais il s'agit d'une variable particulière qui est, comme nous l'avons vu, une association structurée de valeurs et de clefs.

Si nous faisons le test *echo $saisons*, nous constatons en effet que nous n'affichons rien d'autre que *array*. PHP nous indique ainsi que la variable que nous voulons afficher est un tableau de données.

Ce code :

```
echo $saisons; // Faisons le test et constatons que ceci affiche array.
```

Fig. 2

Affiche :

```
array
```

Fig. 3

Pour afficher une valeur du tableau, la **bonne méthode est d'utiliser sa clef**.

Puisque l'on sait que chaque valeur possède une clef unique, nous ne pouvons pas risquer une erreur de relation. Par exemple « Hiver » possède la clef 2. Cette clef est unique et nous allons donc l'utiliser pour afficher « Hiver » comme le montre l'exemple suivant :

Ce code :

```
echo $saisons[2]; // affiche Hiver.
```

Fig. 4

Affiche :

```
Hiver
```

Fig. 5

Ce code utilise les *crochets* [ ] pour sélectionner l'index (ou clé) souhaité. *Les crochets sont caractéristiques* de l'utilisation des tableaux de données. Ils permettent d'accéder à toutes les valeurs du tableau grâce à leur clef.

Si nous voulons afficher Printemps et Été, il suffit de retrouver leurs clefs, qui sont respectivement 3 et 0 et d'écrire ensuite le code suivant :

```
echo $saisons[3]; // affiche Printemps.
echo $saisons[0]; // affiche Été.
```

Fig. 6

Si en revanche nous voulons afficher toutes les valeurs du tableau, sans les afficher une par une, nous devons utiliser une boucle *foreach* qui va parcourir toutes les valeurs du tableau. La boucle *foreach* est une boucle spécialement réservée aux tableaux.

II. La boucle *foreach*

La boucle *foreach* est une boucle permettant de parcourir toutes les valeurs d'un tableau de données et d'en récupérer la structure.

Bien que sa syntaxe soit spécifique, son principe de fonctionnement est similaire à celui des autres boucles *while* et *for* étudiées précédemment : la boucle récupère la structure du tableau tant que des valeurs existent dans le tableau (cf. le chapitre : les conditions, les opérateurs et les boucles).

Dans l'exemple du tableau avec les saisons, la structure du tableau $saisons est la suivante :

Tableau n°2 $saisons

CLEF	VALEUR
0	Été
1	Automne
2	Hiver
3	Printemps

Ainsi, grâce à la boucle *foreach*, nous allons pouvoir récupérer les 4 associations du tableau pour les afficher comme ceci :

```
$saisons = array(0 => 'Été', 1 => 'Automne', 2 => 'Hiver', 3 =>'Printemps');
foreach ( $saisons as $clef => $valeur )
{
      echo $valeur // ou echo $saisons[$clef];
}
```

Fig. 7

Passons en revue la syntaxe utilisée sans revenir sur la ligne de code qui affecte le tableau à la variable $saisons ni sur l'emploi des parenthèses ou des accolades que nous commençons maintenant à bien savoir identifier.

La première ligne débute par le mot-clef *foreach* qui signifie littéralement « pour chaque » ou « pour chacun(e) ». Nous devinons que la boucle va ainsi s'effectuer pour chacune des valeurs contenues dans le tableau.

Ensuite nous rappelons la variable sur laquelle la boucle va s'effectuer : la variable $saisons.

La suite de l'instruction est as *$clef* => *$valeur*. Décomposons cette syntaxe :

– le code *$clef* => $valeur **donne des noms temporaires** aux clefs et aux valeurs du tableau. Ces noms temporaires vont nous permettre à chaque tour de boucle d'afficher les clefs et les valeurs du tableau.

– Ces noms temporaires sont appelés des **pseudos variables**. Elles n'ont d'existence qu'à l'intérieur de la boucle. Il n'est pas possible de les utiliser en dehors des accolades.

– le mot-clef *as*, qui signifie « comme », indique à la boucle qu'elle va pouvoir utiliser **les pseudos variables** indiquées pour agir sur le tableau *$saisons*.

Nous pouvons résumer comme suit ce principe :

Pour chaque élément du tableau $saisons qui **a pour structure** (as) **une clef** (*$clef*) **reliée à une valeur** (*$valeur*), *$clef* et *$valeur* sont les **pseudos variables** définies qui permettent de récupérer les valeurs réelles du tableau au sein de la boucle tant que ces valeurs existent.

La seconde ligne contient le *echo* qui va afficher chaque valeur du tableau grâce à la pseudo variable *$valeur*. **Ainsi la pseudo variable $valeur va prendre à chaque tour de boucle les différentes valeurs contenues dans le tableau $saisons** : Été, puis Automne, puis Hiver, puis Printemps.

```php
foreach ( $saisons as $clef => $valeur )
{
      // la pseudo variable $valeur prend les différentes valeurs du tableau :
      // Été, puis Automne, puis Hiver, puis Printemps.
      echo $valeur ;
}
```

Fig. 8

Si nous souhaitons afficher les clefs du tableau, il suffit de garder la boucle et d'effectuer un *echo* sur la pseudo variable *$clef* comme ceci :

```php
foreach ( $saisons as $clef => $valeur )
{
      echo $clef // affiche 0, puis 1, puis 2, puis 3.
}
```

Fig. 9

Si nous souhaitons afficher les clefs du tableau et les valeurs, il suffit de garder la boucle et d'effectuer un *echo* sur la pseudo variable *$clef* et sur la pseudo variable *$valeur* (avec une concaténation et un tiret entre les deux données) comme ceci :

```
foreach ( $saisons as $clef => $valeur )
{
      echo $clef .' - '. $valeur .'<br>' ;
   // affiche 0 - Été, puis 1 - Automne, puis 2 - Hiver, puis 3 - Printemps.
}
```

Fig. 10

Résultat :

```
0 - Été
1 - Automne
2 - Hiver
3 - Printemps.
```

Fig. 11

III. Remarques importantes sur la boucle *foreach*

La boucle *foreach* est très importante en PHP car les tableaux de données *array* sont toujours très nombreux dans un projet de développement PHP. Il convient donc d'avoir une bonne connaissance de cette boucle.

Passons en revue quelques remarques importantes.

A. La pseudo variable de la clef est facultative

Il n'est pas utile d'indiquer une pseudo variable pour la clef si nous n'en avons pas besoin, dans le cas où par exemple nous voudrions afficher uniquement les saisons mais pas les clefs.

Le code dans ce cas serait de la forme :

```
// notons l'absence de la pseudo variable de la clef
foreach ( $saisons as $valeur )
{
      echo $valeur ;
}
```

Fig. 12

B. Le nommage des pseudo variables est libre

Dans l'exemple précédent, nous avons nommé les pseudo variables *$clef* et *$valeur*.

Étant donné la prédominance de l'anglais dans le développement web, nous rencontrerons souvent leur traduction *$key* et $value.

Le caractère générique de ces noms de pseudo variables nous permet de les utiliser à propos de n'importe quel contenu d'un tableau de données (saisons, voitures, sport, etc.). **Cependant, il est utile de nommer ces pseudo variables en fonction de ce contenu.**

C. Importance du nommage des pseudo variables

Puisque, à chaque tour de boucle, la pseudo variable aura comme valeur le nom d'une saison, il peut être judicieux de la nommer $saison (au singulier) au lieu de la nommer *$valeur*, qui est trop générique, l'intérêt étant que l'on comprend immédiatement ce que contient la pseudo variable $saison.

Il est tout à fait pertinent lors de l'écriture d'un script de nommer les variables (et par là même les pseudo variables) en fonction des valeurs qu'elles contiennent.

Ainsi le code de la boucle pourrait être celui-ci :

```
foreach ( $saisons as $saison )
{
      echo $saison;
}
```

Fig. 13

Si notre tableau contenait des voitures, la boucle serait la suivante :

```
foreach ( $voitures as $voiture )
{
      echo $voiture;
}
```

Fig. 14

IV. *print_r* et *var_dump* : affichage du tableau en mode développeur

En plus de la boucle *foreach*, il existe deux fonctions permettant d'observer la structure et le contenu du tableau. Ces fonctions sont en revanche uniquement réservées au développement.

A. Fonction *print_r*

La fonction *print_r* s'utilise comme ceci :

```
print_r($saisons); // Afficher pour le développeur la structure du tableau.
```

Fig. 15

Elle va afficher le tableau $saisons dans le navigateur comme ceci :

```
Array ( [0] => Été [1] => Automne [2] => Hiver [3] => Printemps )
```

Fig. 16

La structure et le contenu du tableau sont ainsi aisés à observer.

Un moyen encore plus efficace de lire ce résultat et d'observer le code source de la page.

Le code source de la page s'obtient avec le raccourci clavier : **CONTROL + U**

Le code source va nous permettre d'obtenir ce résultat :

```
Array
(
  [0] => Été
  [1] => Automne
  [2] => Hiver
  [3] => Printemps
)
```

Fig. 17

Comme nous pouvons le constater, l'observation est facilitée par les sauts de ligne et l'indentation (tabulation) des lignes.

Cette différence de vue entre la page web et son code source s'explique par le fait que le navigateur n'affiche pas les sauts de ligne des chaînes de caractères sauf si un élément HTML *<pre>* est présent et qu'il peut être interprété. Pour rappel l'élément *<pre>* est l'élément dit de pré-formatage, la mise en forme d'un texte est respectée lors de la restitution de la page web par le navigateur.

Comme il peut être fastidieux d'afficher le code source pour consulter le résultat d'un *print_r*, il est judicieux d'utiliser un *<pre>* avant d'afficher le tableau (et de le refermer ensuite).

```
echo '<pre>' ; // élément HTML de pré-formatage
print_r($saisons);
echo '</pre>' ;
```

Fig. 18

Ainsi le code ci-dessus va-t-il afficher directement la structure voulue dans la page web :

```
Array
(
  [0] => Été
  [1] => Automne
  [2] => Hiver
  [3] => Printemps
)
```

Fig. 19

Dans les exemples ou exercices de ce cours où nous afficherons le résultat d'un *print_r*, nous utiliserons l'élément *<pre>* sans pour autant l'écrire.

B. Fonction *var_dump*

La fonction *var_dump* joue un rôle similaire en donnant directement plus d'informations.

```
var_dump($saisons);
```

Fig.20

En effet, la ligne ci-dessus affiche :

```
array
 0 => string 'Été' (length=3)
 1 => string 'Automne' (length=7)
 2 => string 'Hiver' (length=5)
 3 => string 'Printemps' (length=9)
```

Fig.21

La structure du tableau est visible directement et pour chaque valeur du tableau, nous obtenons :

- le type de valeur, ici : *string*, c'est à dire chaîne de caractères (donc du texte) ;
- la longueur de la chaîne de caractères (length = 3).

Les fonctions *print_r* et *var_dump* sont donc très utiles pour le développement lorsque nous travaillons avec les tableaux de données.

V. À propos des clefs du tableau

A. Clefs numériques

Nous avons créé le tableau $saisons avec des clefs qui sont des nombres : 0, 1, 2 et 3.

Lorsqu'une clef est un nombre, les clefs sont dites des **clefs numériques**.

Il faut savoir que par défaut, PHP crée des tableaux avec des clefs numériques. Il est donc facultatif d'indiquer ces clefs lors de la création d'un tableau.

Ainsi le tableau que nous avons créé de la façon suivante :

```
$saisons = array(0 => 'Été', 1 => 'Automne', 2 => 'Hiver', 3 =>'Printemps');
```

Fig.22

Ce tableau aurait pu être créé sans indiquer les clefs :

```
$saisons = array('Été', 'Automne', 'Hiver', 'Printemps');
// clefs non indiquées
```

Fig.23

Et le résultat aurait été identique :

```
Array
(
  [0] => Été
  [1] => Automne
  [2] => Hiver
  [3] => Printemps
)
```

Fig.24

Il est, dans ce cas très important, de se rappeler que PHP commence toujours pas affecter la clef 0 au premier élément du tableau.

B. Clefs associatives

Il est possible lors de la création d'un tableau d'utiliser des clefs qui ne soient pas des nombres mais des chaînes de caractères.

Soit par exemple le tableau suivant qui décrit une voiture :

```php
$voiture = array(
    'couleur' => 'bleu',
    'modèle' => '608',
    'annee' => '2018'
);
```

Fig.25

Il est tout à fait possible de le traduire en tableau PHP en utilisant les propriétés comme clefs.

Naturellement les clefs étant des chaînes de caractères, nous devons les entourer avec des quotes.

Un tableau avec des clefs qui sont des chaînes de caractères est appelé un **tableau associatif**.

VI. Ajouter ou modifier des valeurs dans un tableau de données

A. Ajout de valeurs

Soit le tableau suivant qui contient des couleurs.

Tableau n°3

CLEF	VALEUR
0	Bleu
1	Rouge
2	Vert

Nous avons créé ce tableau comme suit :

```php
$couleurs = array('bleu', 'rouge', 'vert');
```

Fig. 26

Consultation du tableau avec *print_r* :

```
Array
(
  [0] => bleu
  [1] => rouge
  [2] => vert
)
```

Fig. 27

Les tableaux de données sont des variables.

Pour ajouter des valeurs à un tableau, il suffit d'utiliser la syntaxe basée sur l'emploi des crochets puis d'assigner la valeur voulue comme à une variable.

Pour ajouter par exemple la couleur orange, nous affectons simplement avec les crochets la valeur « orange » à la variable $couleurs.

Concrètement nous écrivons le nom de la variable tableau suivi des crochets, puis du signe égal et enfin de la valeur que nous souhaitons ajouter.

```php
$couleurs[] = 'orange';
```

Fig. 28

Résultat :

```
Array
(
  [0] => bleu
  [1] => rouge
  [2] => vert
  [3] => orange
)
```

Fig. 29

Notons que nous pouvons choisir la clef que nous voulons en l'insérant entre les crochets.

Par exemple, ajoutons la valeur rose avec la clef 11 :

```php
$couleurs[11] = 'rose';
```

Fig.30

Résultat :

```
Array
(
  [0] => bleu
  [1] => rouge
  [2] => vert
  [3] => orange
  [11] => rose
)
```

Fig.31

B. Modification des valeurs d'un tableau

Les crochets permettent d'accéder à n'importe quelle valeur du tableau. Si nous voulons modifier une valeur, il suffit donc d'y accéder avec les crochets puis de lui affecter une nouvelle valeur.

Exemple
Dans le tableau $couleurs, si nous voulons remplacer la valeur orange (qui possède la clef 3) par la valeur violet, nous allons affecter la nouvelle valeur à la clef 3.

```php
$couleurs[3] = 'violet';
// Array
// (
//     [0] => bleu
//     [1] => rouge
//     [2] => vert
//     [3] => violet
//     [11] => rose
// )
```

Fig.32

Le fonctionnement est tout à fait similaire à celui de la modification d'une variable. Cela signifie que les éléments d'un tableau se comportent comme des variables.

Retenons que les crochets permettent d'accéder à n'importe quelle valeur du tableau et nous permettent d'ajouter ou de modifier des valeurs.

Cette simplicité d'ajout et de modification de valeur avec les crochets est un point fort du fonctionnement des tableaux en PHP.

C. Déclarer un tableau de données avec les crochets

Les crochets permettent également de créer un tableau de données, la fonction *array* n'étant pas le seul moyen. Si nous voulons créer un tableau avec une valeur, il suffit de procéder comme si nous ajoutions cette valeur à ce tableau.

Par exemple, si nous créons un tableau dont les valeurs sont des noms de fruits, la syntaxe suivante permet directement :
– de créer le tableau *$fruits* ;
– d'affecter une première valeur.

```php
$fruits[] = 'pomme';
// Array
// (
//     [0] => pomme
// )
```

Fig. 33

Le tableau est donc créé et par la suite, il suffit d'ajouter d'autres valeurs, toujours en utilisant les crochets.

```php
$fruits[] = 'cerise';
$fruits[] = 'banane';
```

Fig. 34

D. Supprimer des valeurs d'un tableau de données

Il est possible de supprimer des valeurs d'un tableau en utilisant la fonction *unset*.

La fonction *unset* est une fonction native de PHP qui permet de supprimer des variables et nous avons vu qu'un élément de tableau est une variable, il suffit donc d'effectuer un *unset* sur cet élément. Ainsi si nous souhaitons supprimer la valeur rose du tableau couleur :

```php
unset($couleurs[11]);
```

Fig. 35

Le tableau contiendra désormais les valeurs suivantes :

```
Array
(
  [0] => bleu
  [1] => rouge
  [2] => vert
  [3] => violet
)
```

Fig. 36

E. Supprimer un tableau de données

La fonction *unset* permet également de supprimer un tableau de données car pour mémoire un tableau de données est une variable.

Il suffit d'affecter *unset* au tableau, comme ceci :

```php
unset($couleurs);
```

Fig. 37

F. Ajout de variables dans un tableau

Il n'existe aucune difficulté à ajouter des variables dans un tableau. Cela revient à ce que les valeurs de ces variables deviennent les valeurs du tableau.

```php
$a = 'texte';
$b[] = $a; // affectation de la valeur de la variable $a au tableau $b
```

Fig.38

La structure du tableau sera alors :

```
Array (
    [0] => texte
)
```

Fig.39

G. Tableaux multidimensionnels : les tableaux de tableaux

Puisque les tableaux sont des variables, il est également possible d'ajouter un tableau dans un tableau.

```php
$a = 'texte';
$b[] = $a; // affectation de la valeur de la variable $a au tableau $b
$c[] = $b; // affectation du tableau $b au tableau $c
print_r($c);
```

Fig.40

Le *print_r($c)* va retourner :

```
Array
(
  [0] => Array
    (
      [0] => texte
    )
)
```

Fig.41

Nous sommes en présence d'un tableau contenu dans un tableau. Nous parlons alors de tableau multidimensionnel, c'est-à-dire qu'il possède plusieurs dimensions ou niveaux. On parle également de tableaux imbriqués ou de tableaux de tableaux (un tableau contenant des tableaux).

Représentons le tableau PHP en tableau classique pour en visualiser la structure :

Tableau n°4 Tableau $c

CLEF	VALEUR	
0	Tableau $b	
	Clef	Valeur
	0	texte

La clef 0 du tableau $c est associée à une valeur qui est le tableau $b.

La clef 0 du tableau $b est associé à une valeur qui est chaîne de caractères « texte ».

Pour manipuler ou modifier le tableau, nous allons comme précédemment utiliser les crochets.

Pour accéder au tableau $b, nous utiliserons la syntaxe $c[0] puisque la clef 0 correspond à ce tableau $b.

Pour accéder à la valeur « texte », et puisque cette valeur possède la clef 0 du tableau $b, et que ce tableau $b est une valeur du tableau $c, nous utilisons deux niveaux de crochets : $c[0][0].

Affichons la valeur texte avec la bonne syntaxe :

```php
echo $c[0][0];
```

Fig. 42

Nous parcourons ainsi les tableaux multidimensionnels avec les crochets.

Tableau n°5 Tableau $c

CLEF	VALEUR	
0	Tableau $b	
	Clef	Valeur
	0	texte
[clef]	[clef]	valeur

Les crochets permettent d'ajouter également des valeurs dans ces tableaux multidimensionnels.

Si nous voulons ajouter un nouveau tableau $b2 au tableau $c, la syntaxe sera :

```php
$b2 = array('texte 2', 'texte 3');
$c[] = $b2;
print_r($c);
```

Fig. 43

Le print_r($c) nous retourne la nouvelle structure :

```
Array
(
  [0] => Array
    (
      [0] => texte
    )
  [1] => Array
    (
      [0] => texte 2
      [1] => texte 3
    )
)
```

Fig. 44

Nous pouvons visualiser cette structure avec un tableau classique :

Tableau n°6 *Tableau $c*

CLEF	VALEUR	
0	Tableau $b	
	Clef	Valeur
	0	Texte
[clef]	Tableau $b2	
	Clef	Valeur
	0	texte 1
	1	texte 2

Affichons la valeur texte 2 avec la bonne syntaxe :

```
echo $c[1][0]; // affiche texte 2
```

Fig. 45

Le premier crochet récupère la valeur de la clef 1, c'est-à-dire le tableau *$b2*.

Le second crochet récupère dans le tableau *$b2* la valeur de la *clef 0*, c'est-à-dire la valeur « texte 2 ».

Selon le même principe, nous procédons comme suit pour afficher « texte 3 » :

```
echo $c[1][1]; // affiche texte 3
```

Fig. 46

Remarquons que nous aurions pu ajouter au tableau $c une valeur qui ne soit pas un tableau, par exemple une chaîne de caractères.

Les principes d'accès aux informations contenues dans un tableau restent toujours les mêmes.

Notons que nous ne sommes pas limités dans l'imbrication de tableaux. Nous pouvons fréquemment dans certains projets concevoir des tableaux à trois, quatre ou cinq niveaux. Dans ce cas il suffit de se souvenir que les crochets permettent de naviguer dans la profondeur du tableau et que chaque crochet correspond à un niveau de tableau.

La modification ou la suppression de valeurs dans un tableau multidimensionnel repose également sur l'utilisation des crochets.

Pour modifier par exemple la valeur « texte 2 » il suffit avec les crochets de récupérer sa clef : la valeur « texte 2 » a pour *clef $c[1][0]* ;

Modifions sa valeur en lui affectant une autre valeur (comme vu précédemment) :

```
$c[1][0] = 'autre texte';
print_r($c);
```

Fig. 47

Le *print_r* va retourner la nouvelle structure modifiée :

```
Array
(
  [0] => Array
    (
      [0] => texte
    )

  [1] => Array
    (
      [0] => autre texte
      [1] => texte 3
    )
)
```

Fig. 48

Supprimons maintenant cette valeur avec la fonction *unset* :

```
unset($c[1][0]);
print_r($c);
```

Fig. 49

Le *print_r* va retourner la nouvelle structure. La valeur « autre texte » a été supprimé.

```
Array
(
  [0] => Array
    (
      [0] => texte
    )

  [1] => Array
    (
      [1] => texte 3
    )
)
```

Fig. 50

Nous avons vu des exemples de tableaux multidimensionnels théoriques. Voici un exemple complet qui illustre le besoin de tableaux multidimensionnels.

Si nous souhaitons par exemple afficher pour les quatre saisons le nom de la saison et une image, il peut être intéressant de stocker l'information avec un tableau de données pour gérer et afficher plus aisément les contenus.

Il serait en revanche moins pertinent d'utiliser deux tableaux distincts qui pourraient ne pas garantir la relation nom de la saison / image de la saison.

Exemple

Tableau n°7 Tableau des noms

$NOM_SAISONS	
CLEF	VALEUR
0	Été
1	Automne
2	Hiver
3	Printemps

Tableau n°8 Tableau des images (noms de fichiers .jpg)

$IMAGE_SAISONS	
CLEF	VALEUR
0	Hiver.jpg
1	Automne.jpg
2	Printemps/jpg
3	Eté.jpg

Dans cet exemple, le nom de saison « *Hiver* » possède la *clef 2* tandis que l'image hiver.jpg possède la *clef 0*. La relation entre ces deux éléments pose une difficulté de type logique. Il serait bien plus simple que les deux éléments nom et image soient accessibles de la même façon.

Pour ce faire, le tableau multidimensionnel est une bonne solution.

Formalisons le tableau que nous souhaitons :

Tableau n°9 Tableau multidimensionnel $saisons

CLEF	VALEUR	
0	Clef	Valeur
	Nom	Eté
	Image	Ete.jpg
1	Clef	Valeur
	Nom	Automne
	Image	Automne.jpg
2	Clef	Valeur
	Nom	Hiver
	Image	Hiver.jpg
3	Clef	Valeur
	Nom	Printemps
	Image	Printemps.jpg
[clef]	[clef]	[valeur]

La clef 0 a pour valeur un tableau. Ce tableau est structuré avec deux clefs : nom et image.

Nous constatons que les relations nom/image sont claires et ne comportent pas de problème logique car les deux valeurs sont rangées dans le même tableau :

– le nom « *hiver* » est accessible avec la syntaxe *$saisons[2]['nom']* ;

– tandis que l'image « *hiver.jpg* » est accessible avec la syntaxe *$saisons[2]['image']*.

À ce propos nous aurions pu concevoir le tableau $saisons comme tableau associatif, en remplaçant les clefs numériques 0, 1, 2 et 3 par les noms de saison.

Nous aurions alors la syntaxe *$saisons['hiver']['nom']* et *$saisons['hiver']['nom']* pour accéder aux valeurs souhaitées.

Il peut s'agir d'une bonne décision dans le cadre de ce projet qui contient peu de données. En revanche dans le cadre d'un projet plus vaste, comme par exemple, la liste des 36 000 communes de France, il serait peu opportun de créer un tableau associatif. Nous utiliserions sans doute les codes postaux des communes comme clefs.

Gardons, dans le cadre de cet exercice, un tableau avec des clefs numériques.

La syntaxe PHP pour créer ce tableau est la suivante :

```php
$saisons = array(
    0 => array('nom' => 'Été', 'image' => 'ete.jpg'),
    1 => array('nom' => 'Automne', 'image' => 'automne.jpg'),
    2 => array('nom' => 'Hiver', 'image' => 'hiver.jpg'),
    3 => array('nom' => 'Printemps', 'image' => 'printemps.jpg'),
);
echo $saisons[2]['nom']; // Hiver
echo $saisons[3]['image']; // printemps.jpg
```

Fig.51

Pour terminer l'exercice et afficher les noms et les images des saisons, nous utilisons une boucle *foreach* en nous rappelant que nous ne pouvons pas afficher un tableau avec *echo* et que nous ne pouvons donc pas effectuer ceci :

```php
foreach ($saisons as $saison) {
    echo $saison; // impossible car $saison est un array !
}
```

Fig.52

En revanche, puisque les tableaux contenus dans $saisons sont construits avec les clefs '*nom* ' et '*image'*, nous allons utiliser ces clefs comme suit :

```php
foreach ($saisons as $saison) {
    echo $saison['nom'];
    echo $saison['image'];
}
```

Fig.53

Naturellement, nous voulons élaborer la page HTML en intégrant correctement ces donnés.

Nous pouvons par exemple intégrer un titre « Les saisons » avec l'élément *<h1>* et intégrer les noms de saison avec l'élément *<h2>*.

Les noms de fichier image seront quant à eux intégrés avec l'élément *<img>*.

Le code final du projet sera donc :

```
echo '<h1>Les saisons</h1>';
foreach ($saisons as $saison) {
    echo '<h2>' . $saison['nom'] . '</h2>';
    echo '<img src="' . $saison['image'] . '"/>';
}
```

Fig. 54

Voici le rendu de la page web dans le navigateur avec une mise en forme CSS minimale :

Fig. 55

H. Quelques fonctions PHP utiles pour les tableaux de données

– *current* : retourne l'élément courant du tableau ;

– *end* : positionne le pointeur de tableau en fin de tableau ;

– *array_key_exists* : vérifie si une clé existe dans un tableau ;

– *in_array* : indique si une valeur appartient à un tableau ;

– *asort* : trie un tableau et conserve l'association des index ;

– *array_multisort* : trie les tableaux multidimensionnels ;

– *count* : affiche le nombre d'entrées du tableau.

Toutes les fonctions relatives aux tableaux sont disponibles dans le manuel PHP : http://fr2.php.net/manual/fr/ref.array.php

Après avoir pris connaissance du chapitre de ce cours dédié à la présentation du manuel PHP, vous êtes invité à lire les informations qu'il contient sur les tableaux de données.