



## Conventions d'écriture du code en PHP

### 1 Préalables

Voici quelques conventions d'écriture du code PHP. Ces conventions sont minimalistes et doivent (devraient) donc au minimum être respectées.

Ce texte est à destination du programmeur débutant ne se destinant pas particulièrement à la programmation. Le lecteur qui voudrait aller plus avant consultera par exemple

- ↪ PHP, conventions d'écriture [\[lien\]](#)
- ↪ PHP, coding guidelines [\[lien\]](#)
- ↪ Les conventions d'écriture du code Java par SUN, [\[lien\]](#) [\[pdf\]](#)

Certains pourraient se demander pourquoi écrire et pourquoi respecter des conventions d'écriture du code. Certains sont également tenté de définir leurs propres conventions ... probablement dans un soucis d'affirmation de soi. En programmation, c'est une mauvaise idée d'essayer de se démarquer de cette façon.

Voici quelques (bonnes) raisons de respecter des conventions :

- ↪ Le programmeur passe une bonne partie de son temps à faire de la maintenance ... et pas spécialement la maintenance de son propre code. Il est donc important qu'il puisse lire rapidement le code écrit par un autre. *Le respect des conventions va l'y aider.*
- ↪ Les conventions ont été réfléchies, elles ont une raison. Par exemple, si l'on demande de couper les lignes à 80 caractères c'est parce que ; tout le monde n'a pas la même largeur d'écran, certains impriment les documents, c'est plus rapide à lire (moins de mouvement de l'œil), ... Le programmeur veut pouvoir lire le code sur le support qui lui convient. *Le respect des conventions va l'y aider.*

## 2 Conventions d'écriture

### 2.1 Choix des noms

#### 2.1.1 Les fichiers

Les noms de fichiers porteront l'extension `.php`. Si le fichier PHP contient un **script**, il s'écrira en minuscule, les mots séparés par un *underscore* (barre de soulignement) <sup>1</sup>. Si le fichier PHP représente une **classe**, son nom commencera par une majuscule et l'on utilisera toujours *underscore* pour séparer les mots.

Les noms de fichiers seront représentatifs de leur contenu, on n'hésitera pas à nommer un fichier `db_utils.php` s'il contient des informations propres à une base de données. Il sera sans doute utile — pour des projets plus conséquents — de regrouper logiquement les scripts dans des répertoires.

```
mon_joli_script.php
authentication.php
Web_User.php
template_liste_contacts.php
```

#### 2.1.2 Les variables

Les noms de variables s'écrivent en **minuscules** et les différents mots sont séparés par une majuscule (*camelCase*). Ils sont représentatifs du contenu de la variable.

Lorsqu'une variable ne l'est plus, c'est-à-dire qu'elle est une **constante**, son nom s'écrit en majuscules, les mots séparés par un *underscore*.

```
$maJolieVariable
$_maJolieVariablePrivee
$NOMBRE_LIGNES
```

**Remarque** Les chaînes de caractères (*String*) sont soit délimitées par un guillemet simple (*single quote*) `'`, soit par un guillemet double (*double quote*) `"`.

On privilégiera l'utilisation du *single quote* afin d'éviter l'expansion des variables.

<sup>1</sup>En PHP, on n'utilise pas la *CamelCase* pour les noms de fichiers.

### 2.1.3 Les fonctions, les méthodes

Les noms de fonctions s'écrivent en *camelCase* et commencent par une minuscule. Ils sont représentatifs de leur action.

Lors d'un appel de fonction, on n'ajoute pas d'espace blanc sauf après la virgule séparant les arguments. Un peu comme suit ;

```
$var = foo($one, 'two', 3);
```

En orienté objet, les accesseurs (*getters*) commencent par `get` et les mutateurs (*setters*) par `set`.

**Remarque** On fera précéder d'un *underscore* les variables et les fonctions ayant une portée `private` ou `protected`.

```
function maJolieFonction() {...}
function getNom() {...}
function displayContacts() {...}
```

## 2.2 Format du fichier

### 2.2.1 Longueur des lignes

La longueur d'une ligne n'excède pas 80 caractères<sup>2</sup>.

On coupe les lignes — en règle général — après une virgule (si l'on doit couper dans les paramètres d'une fonction) et avant un opérateur.

```
function myPrettyFunction($prettyString,
    $prettyObject,
    $defaultStringVeryImportant='importantValue') {
    $value = false;
    if ($firtsCondition
        && $prettyObject->foo() == 'Second_condition'
        && !$prettyObject->bar()) {
        $value = true;
    }
    return $value;
}
```

<sup>2</sup>Certains éditeurs sont capable de tronquer automatiquement les lignes à 80 caractères et d'autres indique — par une ligne rouge par exemple — la limite à ne pas dépasser.

## 2.2.2 Indentation

L'indentation est l'action qui consiste à laisser quelque espace (ou l'une ou l'autre tabulation) en début de ligne. Les éditeurs destinés au code indentent seuls le code ... il suffit de les surveiller.

Chaque indentation fait 4 espaces blancs. On n'utilisera pas la tabulation.

Il existe deux manières de **délimiter les blocs**. On peut choisir celle qui nous convient tant que l'on reste cohérent au sein d'un même code.

Première manière

```
foreach ( $members as $member ) {
    // Instructions here
}
```

Seconde manière

```
foreach ( $members as $member )
{
    // Instructions here
}
```

## 2.2.3 Fichier source et sortie html

Si l'on prend la bonne habitude d'écrire proprement le code source php, il est également bon de fournir une sortie html propre<sup>3</sup>. Pour se faire l'on s'arrange pour passer à la ligne dans le code html généré à l'aide de \n bien placés.

Si je veux générer une liste à puce en html, je peux écrire ceci en php

```
<ul>
<?php
foreach ( $members as $member ) {
    echo '    <li>' . $member . '</li>' . "\n";
}
?>
</ul>
```

Sans le caractère \n en fin de ligne, je n'aurais pas de passage à la ligne dans le code html généré et tous les *item* seraient sur la même ligne<sup>4</sup>.

<sup>3</sup>Même si ce source html est rarement lu, un programmeur pourra si référer dans une phase de *debugging* par exemple. Et c'est mieux s'il y a des passage à la ligne adéquats.

<sup>4</sup>Avec probablement une ligne de plus de 80 caractères.

## 2.3 Structures de contrôle

Comme nous l'avons dit, lors de l'utilisation d'une structure répétitive (`foreach`, `while`, ...) ou d'une structure de contrôle (`if`), il est bon d'indenter « le » bloc d'instructions. Lorsque l'on rencontre des `if else` imbriqués, on évite une indentation exagérée, c'est-à-dire celle qui consiste à indenter chaque sous bloc, comme suit :

```
// Dark side
if ( <cond> ) {
    <instruction>
} else {
    if ( <cond> ) {
        <instruction>
    } else {
        if ( <cond> ) {
            <instruction>
        } else {
            <instruction>
        }
    }
}
```

Dans ce cas, on préférera écrire

```
if ( <cond> ) {
    <instruction>
} else if ( <cond> ) {
    <instruction>
} else if ( <cond> ) {
    <instruction>
} else {
    <instruction>
}
```

## 2.4 Les commentaires

Lorsque l'on documente son code, il faut distinguer les commentaires destinés aux utilisateurs et les commentaires destinés aux programmeurs.

Les premiers sont là afin que l'utilisateur d'un code PHP sache ce que fait le script ou la classe. Ces commentaires sont placés en début de script PHP et en

début de méthode lorsque l'on se trouve dans une optique OO. Ces commentaires peuvent passer à travers une moulinette <http://www.phpdoc.org/> afin de générer une version lisible des commentaires en html destinée aux utilisateurs.

Les seconds, destinés aux programmeurs sont ajoutés dès qu'une partie du code semble plus compliquées et non triviale à la première lecture.

En php, on utilisera les commentaires de la forme `/* */` ou `//`.

Le début d'un script PHP pourra avoir l'allure suivante

```
/**
 * Short description
 *
 * Long description if any
 *
 * PHP versions 4 and 5
 *
 * LICENSE: ... <your license>
 *
 * @author      Author <author@example.com>
 * @copyright   ...
 * ... <other tags>
 */
```

## 2.5 Inclusion de fichiers

Pour l'inclusion de fichier, on privilégiera les instructions `require_once` et `include_once` à leur équivalent `require` et `include`.

## 2.6 Persistance de données

Si des requêtes SQL apparaissent dans les scripts PHP, on écrira ces requêtes en MAJUSCULES afin qu'elle soient facilement repérables. On prendra également l'habitude de bien respecter la casse. En effet si une table d'une base de données, s'appelle `f00`, on utilisera `f00` dans le code PHP et pas `F00` ou `FOO`.