

Convention générale de codage pour la programmation

par [Adrien Pellegrini \(Page d'accueil\)](#)

Date de publication : 28/08/2006

Dernière mise à jour : 28/08/2006

Bien coder est un art. Apprenons comment y parvenir.

I - Introduction.....	3
I.1 - Remerciements.....	3
I.2 - Préambule.....	3
II - En général.....	4
II.1 - Formatage du code.....	4
II.1.1 - Indentation.....	4
II.1.2 - Accolades.....	4
II.1.3 - Interlignes.....	5
II.1.4 - Espaces blancs.....	5
II.1.5 - Longueur d'une ligne.....	6
II.2 - Convention d'écriture.....	6
II.2.1 - Langue.....	6
II.2.2 - Convention d'écriture des variables.....	6
II.2.3 - Convention d'écriture des constantes.....	7
II.2.4 - Convention d'écriture des fonctions et des méthodes.....	7
II.2.5 - Convention d'écriture des commentaires.....	7
Commentaires sur une ligne.....	7
Commentaires en fin de ligne.....	7
Commentaires multi-lignes.....	7
Documentation du code.....	8
À éviter.....	8
III - Langages.....	9
III.1 - PHP.....	9
III.1.1 - Balises courtes d'ouverture.....	9
III.1.2 - TRUE et FALSE.....	9
III.1.3 - Imbrication de fonctions.....	9
III.1.4 - Concaténation de chaîne.....	9
III.1.5 - Les guillemets (quotes).....	9
III.1.6 - mysql_real_escape_string().....	10
III.1.7 - PDO - PHP Data Objects.....	10
III.1.8 - addslashes().....	10
III.2 - SQL.....	10
III.2.1 - Écriture d'une requête SQL.....	10
III.2.2 - Clause WHERE et JOIN.....	10
III.3 - HTML et CSS.....	11
III.3.1 - Clarté du CSS.....	11
III.3.2 - Nom des classes.....	11
III.3.3 - Table vs DIV.....	11
IV - Liens.....	12
V - Conclusion.....	13

I - Introduction

I.1 - Remerciements

Tous mes remerciements à **Yogui** pour sa relecture.

I.2 - Préambule

Ce tutorial a pour but de vous apprendre les bonnes pratiques pour "bien" coder. Mais sachez qu'il n'y a pas qu'une ni des dizaines de façon de coder.

Sachez aussi que la façon dont vous pouvez coder est assez personnelle mais il y a des "grand classiques" et des erreurs à éviter.

Un exemple de code "obfusqué" qui démontre un style assez particulier avec un code tout à fait correct.

Code obfusqué en C

```

char
    _3141592654[3141
], __3141[3141]; _314159[31415], _3141[31415]; main() {register char*
    _3_141,*_3_1415, *_3_1415; register int _314,_31415,__31415,*_31,
    _3_14159,_3_1415;*_3141592654=_31415=2,_3141592654[0][_3141592654
-1]=1[_3141]=5;__3_1415=1;do{__3_14159=_314=0,__31415++;for( _31415
=0;_31415<(3,14-4)*__31415;_31415++)_31415[_3141]=_314159[_31415]= -
1;_3141[*_314159=_3_14159]=_314;_3_141=_3141592654+_3_1415;_3_1415=
__3_1415+_3141;for
    ( _31415 = 3141-
    _31415;_31415--
    ,_3_141 ++,
    +=_314<<2 ;
    *_3_1415;_31
    if(!(*_31+1)
    _31415,_314
    _31415;*_ (
    )+= *_3_1415
    _3_1415 >=
    _3_1415+= -
    )++;_314=_314
    _3_14159 && *
    =1,__3_1415 =
    _314+(__31415
    while ( ++ *
    )*_3_141--=0
    ) ; { char *
    write((3,1),
    ), (_3_14159
    3.1415926; }
    _31415<3141-
    31415% 314-(
    _31415 ] +
    [ 3]+1)-_314;
    ,_3141592654))
    char
    _3141592654[3141
    _31415;_31415--
    _3_1415++){_314
    _314<<=1;_314+=
    =_314159+_314;
    )*_31=_314 /
    [_3141]=_314 %
    _3_1415=_3_141
    =*_31;while(*
    31415/3141 ) *
    10, (*--_3_1415
    [_3141]; if ( !
    _3_1415)_3_14159
    3141-_31415;}if(
    >>1)>=_31415 )
    _3_141==3141/314
    ;}while(_3_14159
    _3_14= "3.1415";
    (--*_3_14,__3_14
    ++,++_3_14159))+
    for ( _31415 = 1;
    1;_31415++)write(
    3,14),_3141592654[
    "0123456789","314"
    puts((*_3141592654=0
    ;_314= *_3.141592; }

```

II - En général

Cette partie est générale à la programmation dans la plupart des langages. Elle se doit donc être la plus générale possible sans rentrer dans certaines conventions des différents langages.

Pour vous rendre directement à la spécification de certains langages, reportez-vous au Chapitre III.

II.1 - Formatage du code

Un code source se présente comme un texte : groupement thématique en paragraphes, indentation pour montrer le niveau d'imbrication, etc. Chaque paragraphe peut être documenté par un commentaire bref.

II.1.1 - Indentation

Une bonne indentation du code permet de mieux s'y retrouver et d'éviter souvent de nombreuses erreurs.

Tous les éditeurs n'interprètent pas les tabulations de la même façon. Certains font qu'une tabulation vaut 3 espaces, d'autres 5 espaces. Donc pour faire une indentation, il vaut mieux utiliser les espaces (généralement 4) plutôt qu'une tabulation.

Comprenez aussi qu'un code écrit comme ci-dessous est quasi illisible !

```
if ($comment == FALSE)
{
echo 'Error';
}
else
{
echo 'Fine';
}
```

Un code sous cette forme est bien plus aéré et compréhensible :

```
if ($comment == FALSE)
{
    echo 'Error';
}
else
{
    echo 'Fine';
}
```

Toutefois, veillez à ne pas exagérer avec vos indentations.

II.1.2 - Accolades

Il existe deux façons très utilisées pour placer vos accolades.

Soit vous les placez après le nom de la fonction (comme dans l'exemple ci-dessous), soit vous les placez en dessous.

à côté

```
function checkpoints($x, $y) {
    ...
}
```

en dessous

```
function checkpoints($x, $y)
{
```

en dessous

```
...  
}
```

Nous vous recommandons la deuxième solution pour des raisons de clarté. Voici un exemple qui démontre bien que la deuxième solution est la plus adéquate.

```
if(!empty($_POST)  
    and !empty($_POST['prenom']) and trim($_POST['prenom']) != ''  
    and !empty($_POST['nom']) and trim($_POST['nom']) != ''  
    and !empty($_POST['adresse']) and trim($_POST['adresse']) != ''  
{  
    // traitement du formulaire...  
}
```

Ce qu'il faut éviter :

- Les accolades sur la même ligne. Il se peut que votre fonction ne fasse qu'une ligne de code. Dans ce cas, il ne faut pas les écrire toutes sur la même ligne.

à éviter

```
function checkpoints($x, $y) { return FALSE; }
```

- Les accolades facultatives.

à éviter

```
while ($x != $y) $x++;
```

II.1.3 - Interlignes

Aérer votre code avec des espaces blancs améliore souvent la structure de votre code. Prenons l'exemple de commentaires.

```
// A variable  
$myVar = 0;  
// An other variable  
$otherVar = 1;
```

Ce code peut devenir très vite chargé. Préférez ceci :

```
// A variable  
$myVar = 0;  
  
// An other variable  
$otherVar = 1;
```

Ainsi, on voit mieux à quelle ligne de code correspond le commentaire.

II.1.4 - Espaces blancs

Veillez à toujours bien espacer vos variables de vos opérateurs.

Ceci :

```
$total = $apple + $pear;
```

est bien plus lisible que cela :

```
$total=$apple+$pear;
```

II.1.5 - Longueur d'une ligne

Veillez à ne pas écrire des lignes de code trop longues. Être obligé d'utiliser la barre de défilement pour voir la fin du code n'est vraiment pas une bonne solution.

80 caractères sur une ligne semblent être la limite pour convenir à toutes les résolutions actuelles sans devoir utiliser la barre de défilement.

II.2 - Convention d'écriture

Afin que tout le monde puisse comprendre l'intérêt d'une telle variable, fonction à tel endroit du code, il est judicieux de bien choisir son nom. Cela permet une auto-documentation du code assez simpliste. Arrangez-vous toujours pour que quiconque lisant votre code puisse le comprendre simplement en voyant les noms.

Veillez aussi à réfléchir à deux fois avant d'écrire une abréviation.

II.2.1 - Langue

Veillez à ne coder qu'en une seule langue. C'est-à-dire qu'il faut éviter de mélanger français et anglais (par exemple). Ainsi, je vous recommande d'utiliser l'anglais pour la bonne et simple raison que c'est la langue universelle utilisée par les développeurs. Toutes les fonctions prédéfinies sont écrites en anglais.

Prenons un système de recherche dans un système de fichiers. Nous aurons probablement besoin d'une variable pour déterminer le résultat de la recherche. Naturellement, nous pensons à "trouvé"... Le souci est que ce mot comporte un accent, ce qui ne convient pas. Notre variable booléenne s'appellera donc "trouve". Maintenant, il risque d'y avoir confusion entre le participe passé et la première personne du singulier de ce verbe conjugué au présent... En regardant le nom de cette variable, s'agit-il de la chaîne à trouver ou bien d'un booléen me permettant de savoir si le résultat a été trouvé ?

Il existe une solution très simple : utiliser une langue ne disposant pas de ces accents que nos langues européennes complexes affectionnent tellement. Il s'agit d'une langue qui a subi de grands changements dont le dernier remonte à l'époque de William Shakespeare, l'un des plus grands contributeurs à l'avènement de l'anglais moderne. En bref : codez en anglais.

II.2.2 - Convention d'écriture des variables

Tout naturellement, pour écrire une variable d'un seul mot, écrivez la, sans majuscule, sous la forme : *variable*.

Pour écrire une variable qui contient plusieurs mots, deux écritures sont possibles.

Soit vous écrivez vos variables sous la forme : *attachedVars*.

Soit vous les écrivez sous cette forme : *spaced_vars*.

- Ce qu'il faut éviter :
- Les variables trop longues (3, 4 parties maximum) : *longVariableLikeThisIsWrong* ou *long_variable_like_this_is_wrong*.
- Les variables qui sont peu compréhensibles : *lVar* (au lieu de *lengthVar*).
- Les variables quasi-semblables : *myVar* et *myVars*.
- Mettre une majuscule au début des variables : *MyVar*.

Personnellement et sûrement certains d'entre vous ont pour habitude de nommer les classes avec le premier caractère en majuscule. Ceci afin de bien distinguer les fonctions des classes.

```
$template = template($variable);
```

```
$template = new Template($variable);
```

À titre indicatif, il existe un système appelé "Hungarian Notation", inventé par Charles Simonyi. Ce système consiste à écrire écrire son type avant chaque variable.

Par exemple, pour écrire une variable qui contiendra un entier, on peut écrire *iVar* (i pour integer).

II.2.3 - Convention d'écriture des constantes

Par convention, les constantes s'écrivent entièrement en caractères majuscules : *CONSTANT*.

Pour écrire une constante de plusieurs mots, on utilise l'underscore : *MY_CONSTANT*.

II.2.4 - Convention d'écriture des fonctions et des méthodes

Pour différencier vos propres fonctions des fonctions prédéfinies, je vous conseille de les écrire sous cette forme : *myFunction()* ou *functions()*.

Pour les noms de classes en PHP, on utilise souvent : *MyClass* ou *Class*.

II.2.5 - Convention d'écriture des commentaires

Les commentaires permettent d'expliquer au lecteur de votre code le fonctionnement de votre programme. Il peut vous sembler ennuyeux de les écrire mais ils vous seront d'une précieuse aide si vous devez vous relire après quelques mois ou si quelqu'un essaye de comprendre votre code.

Considérez aussi que la pause de commentaires fait partie du temps développement.

Veillez éviter aussi des commentaires inutiles. Il ne servent pas de décorations !

Commentaires sur une ligne

```
// My comment
if ($comment == FALSE)
{
    echo 'Error';
}

// My other comment
if ($variable)
{
    echo 'Bon';
}
```

Commentaires en fin de ligne

```
$myVar = array(); // Array
$test = 0; // Test variable
```

Ce commentaire peut souvent remplacer le commentaire sur une ligne mais, si votre code est assez complexe et long, et si vous avez besoin de mettre beaucoup de lignes en commentaire, veillez à utiliser le type de commentaire précédent.

Commentaires multi-lignes

```
/* My comment is too long to be written on a simple line
then I think that I'll write it on multi-line */

// else ...
```

```
/*
$variable = 1;
$test = false;

if ($test)
{
    $variable = 0;
}
*/
echo $variable;
```

Documentation du code

Nombreux développeurs ne connaissent pas cette pratique pourtant bien utile. Elle consiste à documenter son code de telle façon qu'un programme externe puisse fournir les informations utiles concernant le code rien qu'en examinant les commentaires. Pour cela, les commentaires doivent s'écrire d'une certaine façon, comme par exemple pour les fonctions :

```
/**
 * Check if points are in the ship
 * @param int x the height
 * @param int y the width
 * @return bool TRUE / FALSE
 */
function checkpoints($x, $y)
{
    ...
}
```

Comparatif des générateurs de documentation PHP par Hugo Etievant

À éviter

- Placer le code en commentaire de cette façon :

```
if ($comment == FALSE)
{
    echo 'Error' . $myVar /*.' !!!!'*/;
}
```

III - Langages

III.1 - PHP

III.1.1 - Balises courtes d'ouverture

L'utilisation des balises courtes d'ouverture `<? ?>` est déconseillée.

Si vous utilisez les balises courtes d'ouverture et que la directive `short_open_tag` est à *off* sur le server, aucune de vos pages ne pourra être exécutée.

Si les short tags sont désactivés, cela peut donner lieu à une situation cocasse (ou catastrophique, suivant le point de vue) : puisque le code PHP n'est pas interprété, il est envoyé au navigateur Web de l'internaute. Il n'est cependant pas affiché dans la page puisque le tag commence par `<`, ce que le navigateur comprend comme un début de balise HTML, or une balise HTML inconnue du navigateur est ignorée. Ainsi, tout ce qui se trouve jusqu'au `>` suivant est caché à l'internaute.

III.1.2 - TRUE et FALSE

Il serait bon d'utiliser `TRUE` et `FALSE` à la place de 1 et 0.

Voici un lien intéressant sur la [Comparaison de types en PHP](#).

III.1.3 - Imbrication de fonctions

Veillez à limiter l'imbrication de fonctions à 3. Cela vous permettra de ne pas faire d'erreur stupide en vous trompant de paramètre ou en oubliant une parenthèse fermante.

```
$randomStr = md5(uniqid(microtime() * 100000, TRUE));
```

III.1.4 - Concaténation de chaîne

Pour concaténer une chaîne en PHP on utilise le point.

Merci de ne pas utiliser la virgule et d'éviter de mettre un espace après le point. Cela ne fait qu'augmenter la taille de la ligne et ce n'est vraiment pas indispensable pour la bonne compréhension du code.

```
$variable = 'Bonjour';  
$text = $variable.' moi';
```

III.1.5 - Les guillemets (quotes)

Il est bon de savoir la différence entre les guillemets simples (apostrophes) et les doubles (guillemets anglais).

D'une part, les guillemets simples s'exécutent plus vite que les guillemets doubles (voir tableau).

D'autre part, les guillemets doubles interprètent les variables, ce qui a pour conséquence d'allonger le temps d'exécution.

Voici un tableau reprenant le temps d'exécution du code

```
echo 'Vive Développez'.$variable.'!!<br />'; et $variable = '.com';
```

exécuté 100 fois dans une boucle. Seul le temps minimum et maximum y sont affichés.

Simple guillemets	Double Guillemets
echo 'Vive Développez'.\$variable.' !! ';	echo "Vive Développez \$variable !! ";
0.476837158203	0.691413879395
0.715255737305	1.47819519043

III.1.6 - `mysql_real_escape_string()`

`mysql_real_escape_string()` protège les caractères spéciaux d'une requête MySQL. Cette fonction permet d'éviter les attaques par injection SQL.

Utiliser `mysql_real_escape_string()` de la façon suivante est complètement absurde.

```
$variable = htmlentities(mysql_real_escape_string(intval($variable)));
```

Tout d'abord, la fonction `htmlentities()` n'est pas destinée à protéger quelque chose prévu pour être stocké dans une base de données.

Ensuite, la fonction `intval()` permet d'obtenir une valeur numérique entière, ce qui empêche de manière très effective toute tentative d'injection SQL.

Pour finir, `mysql_real_escape_string()` ne sert strictement à rien ici puisque l'appel à `intval()` est suffisant.

III.1.7 - PDO - PHP Data Objects

L'extension PDO (PHP Data Objects) est une interface d'abstraction pour les accès à une base de données. Quel que soit le type de base de données que vous utilisez, le code SQL restera le même.

Pour plus d'informations, reportez vous [ici](#).

III.1-8 - `addslashes()`

Il est préférable d'utiliser les fonction comme `mysql_real_escape_string()` (MySQL) ou `sqlite_escape_string` (SQLite) à la place de `addslashes()` pour préparer une requête SQL.

III.2 - SQL

III.2.1 - Écriture d'une requête SQL

Il est préférable de mettre les mot-clés SQL en majuscules pour bien les distinguer de ce qui est défini par l'utilisateur.

```
select id, nom from etudiant where id < 30;
```

```
SELECT id, nom FROM etudiant WHERE id < 30;
```

III.2.2 - Clause WHERE et JOIN

Vous aurez parfois besoin de sélectionner des données contenues dans plusieurs tables. Pour cela, il est préférable d'utiliser les jointures que la clause `WHERE`.

Pour anecdote, la prochaine version de SQL Server (qui en est actuellement à la version 2005) interdira les jointures à l'aide du mot clef `WHERE`.

Voici un lien explicatif sur les jointures par [SQLpro](#)

III.3 - HTML et CSS

III.3.1 - Clarté du CSS

Au lieu d'écrire quelque chose du genre, il serait bon d'utiliser les feuilles de style !

```
<DIV style='overflow: auto; height: 250; width: 179; scrollbar-face-color:
#EAEDF4; scrollbar-highlight-color: #EAEDF4; scrollbar-shadow-color: #EAEDF4;
scrollbar-3dlight-color: #EAEDF4; scrollbar-arrow-color: #5F107C;
scrollbar-track-color: #EAEDF4; scrollbar-darkshadow-color: #EAEDF4'>
<p style="margin-top: 0; margin-bottom: 0">
<font color="#6801F5" face="Verdana" size="1">
```

III.3.2 - Nom des classes

Il ne faut pas nommer une classe en fonction de l'apparence qu'elle donne à l'élément.

Si l'on prend l'exemple d'une classe nommée "yellow-box", qui comme son nom l'indique, mettrait une couleur de fond jaune à la boîte. Plus tard, si vous voulez changer la couleur de fond, non seulement il faudra modifier le code CSS de la classe mais en plus modifier son nom. Ça n'aurait plus aucun sens de la nommer "yellow-box" si la couleur de fond est orange.








Normalement, toute classe CSS est destinée à être appliquée à un type particulier d'élément visuel selon ce qu'il contient, par exemple une réponse dans le forum (la case complète avec la partie pseudo + le titre + le contenu) de classe "forum-answer" ou simplement le message de cette réponse "forum-answer-message". Tout est dans la sémantique.

Au-delà des problèmes évidents de maintenance, il est tout à fait probable que les moteurs de recherche, lors de leurs opérations de référencement, examinent ces classes CSS de manière à donner de bons ou de mauvais points selon leur utilisation.

III.3.3 - Table vs DIV

Les tableaux servent à afficher des données tabulaires et non à structurer une page. Pour structurer une page, vous devez utiliser la balise *div* et pour le style, CSS.

IV - Liens

-  **CSS : Notions de base**
-  **Bonnes pratiques de SEO ('white hat techniques')**
-  **Div et CSS : une mise en page rapide et facile**
-  **Apostrophes ou guillemets : lesquels choisir ?**
-  **Standards de codage du projet PHP::PEAR**
-  **Unmaintainable code**
-  **Google : "coding practices"**

V - Conclusion

Adopter une façon de coder "standard" permet à tout le monde de se comprendre en lisant le même code. Il n'est pas nécessaire de la respecter à la lettre mais du moins ce qui est énoncé ici paraît le strict minimum à savoir. De plus, coder d'une façon ou d'un autre n'a aucun effet sur le comportement du code, mais le faire proprement n'aura que des avantages.