PHP Coding Guidelines

Why are guidelines important?

Coding Standards are an important factor for achieving a high quality code. A common visual style, naming conventions and other technical settings allow us to produce a homogenous code which is easy to read and maintain. However, not all important factors can be covered by rules and coding standards. Equally important is the style in which certain problems are solved programmatically - it's the personality and experience of the individual developer which shines through and ultimately makes the difference between technically okay code or a well considered, mature solution.

These guidelines try to cover both, the technical standards as well as giving incentives for a common development style. These guidelines must be followed by everyone

1. PHP tags:

Always use <?php ?> to delimit PHP code, not the shorthand version <? ?>. (<? ?> is now deprecated in latest PHP versions)

2. Adding Comments:

Whenever you are using comments for single lines at that time use "single line comments".
For example: //comment goes here

Whenever you are using comments for functions, classes,etc... use "multiline comments"

For example:

```
/**
   *
   * Comments (including variable return types, what function or class is used for etc...)
   *
   */
```

You must have to provide comments on each required lines with proper description
For example:

Functions:

```
/**
 * Does something interesting
 *
 * @param  varchar $where  Where something interesting takes place
 * @param  integer  $repeat How many times something interesting should happen
 * @throws Some_Exception_Class If something interesting cannot happen
 * @return Status
 */
```

Classes:

```
/**
 * Short description for class
 *
 * Long description for class (if any)...
 *
 */
```

3. Variable names:

Variable names should be all lowercase, with words separated by underscores. For example, $current_user is correct, but $currentuser, $currentUser and $CurrentUser are not.

4. Constants:

Same as "variables (above)" but should have all letters in capital.Use '_' as the word separator
For example:
1) "APPLE" is correct, but "Apple" is not.
2) define("A_CONSTANT", "Hello world!"); => correct

3) define("A CONSTANT", "Hello world!"); => not correct

5. Loop indices:

This is the only occasion where single character names are permitted. Unless you already have a specific counting variable, use $i as the variable for the outermost loop, then go onto $j for the next most outermost loop etc. However, do **not** use the variable $l(lowercase 'L') in any of your code as it looks too much like the number 'one'.

Example of nested loops using this convention:

```
for ( $i = 0; $i < 5; $i++ )
{
 for ( $j = 0; $j < 4; $j++ )
 {
  for ( $k = 0; $k < 3; $k++ )
  {
   for ( $m = 0; $m < 2; $m++ )
   {
    foo($i, $j, $k, $m);
   }
  }
 }
}
```

6. Classes:

Classes should be given descriptive names.Class names should always begin with an uppercase letter. The class hierarchy is also reflected in the class name, each level of the hierarchy separated. Examples of good class names are:
Log
XML_RPC
HTML_Upload_Error

7. Functions:

Function names should follow the same guidelines as variable names, although they should include a verb somewhere if at all possible. Examples include `get_user_data()` and`validate_form_data()`. Basically, make it as obvious as possible what the function does from its name, whilst remaining reasonably concise.

Since function arguments are just variables used in a specific context, they should follow the same guidelines as variable names.

It should be possible to tell the main purpose of a function just by looking at the first line, e.g. `get_user_data($username)`. By examination, you can make a good guess that this function gets the data of a user with the username passed as the `$username` argument.

Function arguments should be separated by spaces, both when the function is defined and when it is called. However, there should not be any spaces between the arguments and the opening/closing brackets.

Some examples of correct/incorrect ways to write functions:

```
get_user_data( $username, $password ); // incorrect: spaces next to brackets
get_user_data($username,$password); // incorrect: no spaces between arguments
get_user_data($a, $b); // ambiguous: what do variables $a and $b hold?
get_user_data($username, $password); // correct
```

8. Code Layout:

Including braces

Braces should **always** be included when writing code using `if`, `for`, `while` etc. blocks. There are **no exceptions** to this rule, even if the braces could be omitted. Leaving out braces

makes code harder to maintain in the future and can also cause bugs that are very difficult to track down.

Some examples of correct/incorrect ways to write code blocks using braces:

```
/* These are all incorrect */

if ( condition ) foo();

if ( condition )
 foo();

while ( condition )
 foo();

for ( $i = 0; $i < 10; $i++ )
 foo($i);

/* These are all correct */

if ( condition )
{
 foo();
}

while ( condition )
{
 foo();
}

for ( $i = 0; $i < 10; $i++ )
{
```

```
 foo($i);
}
```

Where to put braces

Braces should always be placed on a line on their own; again there are no exceptions to this rule. Braces should also align properly (use two spaces to achieve this) so a closing brace is always in the same column as the corresponding opening brace. For example:

```
if ( condition )
{
 while ( condition )
 {
  foo();
 }
}
```

Spaces between the tokens

There should always be one space on either side of a token in expressions, statements etc. The only exceptions are commas (which should have one space after, but none before), semi-colons (which should not have spaces on either side if they are at the end of a line, and one space after otherwise). Functions should follow the rules laid out already, i.e. no spaces between the function name and the opening bracket and no space between the brackets and the arguments, but one space between each argument.

Control statements such as `if`, `for`, `while` etc. should have one space on either side of the opening bracket, and one space before the closing bracket. However, individual conditions inside these brackets (e.g. `($i < 9) || ($i > 16)`) should **not** have spaces between their conditions and their opening/closing brackets.

In these examples, each pair shows the incorrect way followed by the correct way:

```
$i=0;
$i = 0;

if(( $i<2 )||( $i>5 ))
if ( ($i < 2) || ($i > 5) )

foo ( $a,$b,$c )
foo($a, $b, $c)

$i=($j<5)?$j:5
$i = ($j < 5) ? $j : 5
```

Tabs and spaces

Do not use spaces to indent: use tabs. Indent as much as needed, but no more.

For Example:

```
 if($abc == 1)

{

   <TAB>echo "abc";

   }
```

Quoting Strings

Strings in PHP can either be quoted with single quotes (') or double quotes (""). The difference between the two is that the parser will use variable-interpolation in double-quoted strings, but not with single-quoted strings. So if your string contains no variables, use single quotes and save the parser the trouble of attempting to interpolate the string for variables, like so:

```
$str = "Avoid this - it just makes more work for the parser.";
$str = 'This is much better.'
```

Likewise, if you are passing a variable to a function, there is no need to use double quotes:

```
foo("$bar"); // No need to use double quotes
foo($bar); // Much better
```

Finally, when using associative arrays, you should include the key within single quotes to prevent any ambiguities, especially with constants:

```
$foo = bar[example]; // Wrong: what happens if 'example' is defined as a constant elsewhere?
$foo = bar['example']; // Correct: no ambiguity as to the name of the key
```

However, if you are accessing an array with a key that is stored in a variable, you can simply use:

```
$foo = bar[$example];
```

Shortcut Operators

The shortcut operators ++ and -- should **always** be used on a line of their own, with the exception of `for` loops. Failure to do this can cause obscure bugs that are incredibly difficult to track down. For example:

```
$foo[$i++] = $j; // Wrong: relies on $i being incremented after the expression is evaluated
$foo[--$j] = $i; // Wrong: relies on $j being decremented before the expression is evaluated

$foo[$i] = $j;
$i++; // Correct: obvious when $i is incremented

$j--;
$foo[$j] = $i; // Correct: obvious when $j is decremented
```

9. SQL code layout:

When writing SQL queries, capitalise all SQL keywords (SELECT, FROM, VALUES, AS etc.) and leave everything else in the relevant case. If you are using WHERE clauses to return data corresponding to a set of conditions, enclose those conditions in brackets in the same way you would for PHP if blocks, e.g. SELECT * FROM users WHERE ((registered = 'y') AND ((user_level = 'administrator') OR (user_level = 'moderator'))).

10. General Guidelines:

File Names

Whenever you save new file give them proper descriptive names.

If your file is class file then give it name like "fileName.class.php"

If your file is normal file then give it name like "fileName.php"

Dynamic URLs

Make a habit to use Dynamic URL always. For example, if the project url is http://www.example.com then do not ever use this url statically in any of the php file.

Use a constant instead like,

*define("BASE_URL", "*http://www.example.com*");*

and use this constant in every file.

Take Backups

Whenever you do any changes on live server at that time take backup of whole file first then make changes in it. And also arrange backup of files by date wise.

Test what you've done

Client is not a tester so please test once before deliver to client.