ryansechrest / **php-style-guide.md**

Last active 20 hours ago

Star

Code Revisions 451 Stars 285 Forks 100

PHP style guide with coding standards and best practices.

php-style-guide.md

PHP Style Guide

All rules and guidelines in this document apply to PHP files unless otherwise noted. References to PHP/HTML files can be interpreted as files that primarily contain HTML, but use PHP for templating purposes.

> The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Most sections are broken up into two parts:

1. Overview of all rules with a quick example
2. Each rule called out with examples of do's and don'ts

Icon Legend:

· Space, ⇥ Tab, ↵ Enter/Return

Table of Contents

1. Files

This section describes the format and naming convention of PHP files.

File Format

1. **Character encoding** MUST be set to UTF-8 without BOM
   - Sublime.app → `File › Save with Encoding › UTF-8`
2. **Line endings** MUST be set to Unix (LF)

- Sublime.app → `View › Line Endings › Unix`

Filename

1. **Letters** MUST be all lowercase
   - e.g. `autoloader.php`
2. **Words** MUST be separated with a hyphen
   - e.g. `app-config.php`

▲ Table of Contents

2. Skeleton

This section showcases a barebones PHP file with its minimum requirements.

Line by line breakdown:

- **Line 1**: PHP open tag
- **Line 2**: Blank line
- **Line 3**: Your code
- **Line 4**: Blank line
- **Line 5**: End-of-file comment
- **Line 6**: Blank line

```
<?php

// your code

// EOF

```

▲ Table of Contents

3. PHP Tags

This section describes the use of PHP tags in PHP and PHP/HTML files.

1. **Open tag** MUST be on its own line and MUST be followed by a blank line
   - i.e. `<?php ↵ ↵ ...`
2. **Close tag** MUST NOT be used in PHP files
   - i.e. no `?>`
3. **Open/close tag** MUST be on one line in PHP/HTML files
   - i.e. `<?php ... ?>`

4. Short open tag MUST NOT be used
   - i.e. `<?` → `<?php`
5. Short echo tag SHOULD be used in PHP/HTML files
   - i.e. `<?php echo` → `<?=`

▲ Table of Contents

1. Open Tag

Open tag MUST be on its own line and MUST be followed by a blank line.

✖ Incorrect

```php
<?php print_welcome_message();
```

↳ Incorrect because `<?php` is not on its own line.

```php
<?php
print_welcome_message();
```

↳ Incorrect because `<?php` is not followed by a blank line.

✔ Correct

```php
<?php

print_welcome_message();
```

▲ PHP Tags

2. Close Tag

Close tag MUST NOT be used in PHP files.

✖ Incorrect

```php
<?php

print_welcome_message();

?>
```

↳ Incorrect because `?>` was used.

✔ Correct

```
<?php

print_welcome_message();
```

▲ PHP Tags

3. Open/Close Tag

Open/close tag MUST be on one line in PHP/HTML files.

✖ Incorrect

```
<div>
	<h1><?php
	print_welcome_message();
	?></h1>
</div>
```

↳ Incorrect because `<?php` and `?>` are not on one line.

✔ Correct

```
<div>
	<h1><?php print_welcome_message(); ?></h1>
</div>
```

▲ PHP Tags

4. Short Open Tag

Short open tag MUST NOT be used.

✖ Incorrect

```
<?

print_welcome_message();
```

↳ Incorrect because `<?` was used instead of `<?php` .

✔ Correct

```
<?php

print_welcome_message();
```

▲ PHP Tags

5. Short Echo Tag

Short echo tag SHOULD be used in PHP/HTML files.

~ Acceptable

```
<div>
	<p><?php echo get_welcome_message(); ?></p>
</div>
```

↳ Acceptable, but `<?=` should be used over `<?php echo` when possible.

✔ Preferred

```
<div>
	<p><?= get_welcome_message(); ?></p>
</div>
```

▲ PHP Tags

4. End of File

This section describes how every PHP file must end.

End-of-file comment:

- MUST be included at the end of a file
  - i.e. `// EOF`
- MUST be on its own line
  - i.e. `↵ // EOF`
- MUST be surrounded by blank lines
  - i.e. `... ↵ ↵ // EOF ↵`

✖ Incorrect

```
<?php
```

```
print_welcome_message();
```

↳ Incorrect because `// EOF` is missing.

```
<?php

print_welcome_message(); // EOF
```

↳ Incorrect because `// EOF` is not on its own line.

```
<?php

print_welcome_message();
// EOF
```

↳ Incorrect because `// EOF` is not surrounded by blank lines.

✓ **Correct**

```
<?php

print_welcome_message();

// EOF
```

▲ Table of Contents

5. Namespaces

This section describes how to use one or more namespaces and their naming convention.

1. **Namespace declaration** MUST be the first statement and MUST be followed by a blank line
   - i.e. `<?php ↵ ↵ namespace MyCompany; ↵ ↵ ...`
2. **Namespace name** MUST start with a capital letter and MUST be camelcase
   - e.g. `namespace MyCompany;`
3. **Multiple namespaces** MUST use the curly brace syntax
   - i.e. `namespace MyCompany { ... }`
4. **Magic constant** SHOULD be used to reference the namespace name
   - i.e. `__NAMESPACE__`

▲ Table of Contents

1. Namespace Declaration

Namespace declaration MUST be the first statement and MUST be followed by a blank line.

✖ Incorrect

```php
<?php

print_welcome_message();

namespace MyCompany;

// EOF
```

↳ Incorrect because `namespace MyCompany;` is not the first statement.

```php
<?php

namespace MyCompany;
print_welcome_message();

// EOF
```

↳ Incorrect because `namespace MyCompany;` is not followed by a blank line.

✔ Correct

```php
<?php

namespace MyCompany;

print_welcome_message();

// EOF
```

▲ Namespaces

2. Namespace Name

Namespace name MUST start with a capital letter and MUST be camelcase.

✖ Incorrect

```php
<?php

namespace myCompany;

// EOF
```

↳ Incorrect because `myCompany` does not start with a capital letter.

```php
<?php

namespace MyCOMPANY;

// EOF
```

↳ Incorrect because `MyCOMPANY` is not written in camelcase.

✔ Correct

```php
<?php

namespace MyCompany;

// EOF
```

▲ Namespaces

3. Multiple Namespaces

Multiple namespaces MUST use the curly brace syntax.

✖ Incorrect

```php
<?php

namespace MyCompany\Model;

namespace MyCompany\View;

// EOF
```

↳ Incorrect because there are two namespaces and the curly brace syntax was not used.

✔ **Correct**

```php
<?php

namespace MyCompany\Model {
	// model body
}

namespace MyCompany\View {
	// view body
}

// EOF
```

▲ Namespaces

4. Magic Constant

Magic constant SHOULD be used to reference the namespace name.

~ **Acceptable**

```php
<?php

namespace MyCompany\Model {
	// model body
}

namespace MyCompany\View {
	$welcome_message = MyCompany\View\get_welcome_message();
}

// EOF
```

↳ Acceptable, but using `__NAMESPACE__` instead of `MyCompany\View` is preferred.

✔ **Preferred**

```php
<?php

namespace MyCompany\Model {
	// ModuleOne body
}
```

```
namespace MyCompany\View {
	$welcome_message = __NAMESPACE__ . '\\' . get_welcome_message();
}

// EOF
```

▲ Namespaces

6. Comments

This section describes how comments should be formatted and used.

1. **Single-line comments** MUST use two forward slashes
   - e.g. `// My comment`
2. **Multi-line comments** MUST use the block format
   - i.e. `/** ↵ * My comment ↵ */`
3. **Header comments** SHOULD use the block format
   - i.e. `/** ↵ * Name of code section ↵ */`
4. **Divider comments** SHOULD use the block format with asterisks in between
   - i.e. `/** 75 asterisks */`
5. **Comments** MUST be on their own line
   - i.e. `↵ // My comment`
6. **Blocks of code** SHOULD be explained or summarized
   - e.g. `// Compare user accounts from export against expired accounts in system`
7. **Ambiguous numbers** MUST be clarified
   - e.g. `// 1,000 processable records per hour API limit`
8. **External variables** MUST be clarified
   - e.g. `// Database object included in file.php`

▲ Table of Contents

1. Single-line Comments

Single-line comments MUST use two forward slashes.

✖ **Incorrect**

```
<?php

/* This is a comment */
```

```
// EOF
```

↳ Incorrect because it uses `/*` and `*/` for a single-line comment.

✅ Correct

```php
<?php

// This is a comment

// EOF
```

▲ Comments

2. Multi-line Comments

Multi-line comments MUST use the block format.

❌ Incorrect

```php
<?php

// This is a
// multi-line
// comment

// EOF
```

↳ Incorrect because it uses `//` for a multi-line comment.

✅ Correct

```php
<?php

/**
 * This is a
 * multi-line
 * comment
 */

// EOF
```

▲ Comments

3. Header Comments

Header comments SHOULD use the block format.

```php
<?php

/**
 * Global application settings
 */

define('SETTING_ONE', '');
define('SETTING_TWO', '');
define('SETTING_THREE', '');

// EOF
```

▲ Comments

4. Divider Comments

Divider comments SHOULD use the block format with 75 asterisks in between.

✖ **Incorrect**

```php
<?php

/**#######################################################################*/

// EOF
```

↳ Incorrect because it uses `#` instead of `*`.

```php
<?php

/*************/

// EOF
```

↳ Incorrect because it uses 10 instead of 75 `*`.

✔ **Correct**

```php
<?php

/**
 * Beginning + Middle + End
 * 3 spaces + 75 spaces + 2 spaces = 80 character line limit
 */

/*****************************************************************************/

// EOF
```

▲ Comments

5. Comments

Comment MUST be on their own line.

✗ Incorrect

```php
<?php

print_welcome_message(); // Prints welcome message

// EOF
```

↳ Incorrect because `// Prints welcome message` is not on its own line.

✓ Correct

```php
<?php

// Prints welcome message
print_welcome_message();

// EOF
```

▲ Comments

6. Blocks of Code

Blocks of code SHOULD be explained or summarized.

~ Acceptable

```php
<?php

foreach ($users as $user) {
	if ($expr1) {
		// ...
	} else {
		// ...
	}
	if ($expr2) {
		// ...
	} elseif ($expr3) {
		// ...
	} else {
		// ...
	}
	// ...
}

// EOF
```

↳ Acceptable, but block of code should be explained or summarized.

✔ Preferred

```php
<?php

/**
 * Get active website bloggers with profile photo for author page.
 * If no photo exists on website, check intranet.
 * If neither location has photo, send user email to upload one.
 */
foreach ($users as $user) {
	if ($expr1) {
		// ...
	} else {
		// ...
	}
	if ($expr2) {
		// ...
	} elseif ($expr3) {
		// ...
	} else {
		// ...
	}
	// ...
}
```

```
// EOF
```

▲ Comments

7. Ambiguous Numbers

Ambiguous numbers MUST be clarified.

✖ Incorrect

```
<?php

while ($expr && $x < 1000) {
        // ...
}

// EOF
```

↳ Incorrect because `1000` is not clarified.

✔ Correct

```
<?php

// Script times out after 1,000 records
while ($expr && $x < 1000) {
        // ...
}

// EOF
```

▲ Comments

8. External Variables

External variables MUST be clarified.

✖ Incorrect

```
<?php

include_once 'some-file.php';
```

```
// ...

foreach($users as $user) {
	// ...
}

// EOF
```

↳ Incorrect because source of `$users` is not clear.

✔ **Correct**

```
<?php

include_once 'some-file.php';

// ...

// $users from some-file.php
foreach($users as $user) {
	// ...
}

// EOF
```

▲ Comments

7. Includes

This section describes the format for including and requiring files.

1. **Include/require once** SHOULD be used
   - i.e. `include` → `include_once` , `require` → `require_once`
2. **Parenthesis** MUST NOT be used
   - e.g. `include_once('file.php');` → `include_once 'file.php';`
3. **Purpose of include** MUST be documented with a comment
   - e.g. `// Provides WordPress environment` ↲ `require_once 'wp-load.php';`

▲ Table of Contents

1. Include/Require Once

Include/require once SHOULD be used.

~ Acceptable

```php
<?php

include 'some-file.php';
require 'some-other-file.php';

// EOF
```

↳ Acceptable, but `_once` should be appended to `include` and `require` if possible.

✔ **Preferred**

```php
<?php

include_once 'some-file.php';
require_once 'some-other-file.php';

// EOF
```

▲ Includes

2. Parenthesis

Parenthesis MUST NOT be used.

✖ **Incorrect**

```php
<?php

include_once('some-file.php');
require_once('some-other-file.php');

// EOF
```

↳ Incorrect because `include_once` and `require_once` are used with parenthesis.

✔ **Correct**

```php
<?php

include_once 'some-file.php';
require_once 'some-other-file.php';
```

```
// EOF
```

▲ Includes

3. Purpose of Include

Purpose of include MUST be documented with a comment.

❌ **Incorrect**

```
<?php

require_once 'some-file.php';

// EOF
```

↳ Incorrect because there is no comment as to what `some-file.php` does or provides.

✔ **Correct**

```
<?php

// Provides XYZ framework
require_once 'some-file.php';

// EOF
```

▲ Includes

8. Formatting

This section outline various, general formatting rules related to whitespace and text.

1. **Line length** MUST NOT exceed 80 characters, unless it is text
   - i.e. `|---- 80+ chars ----|` → refactor expression and/or break list values
2. **Line indentation** MUST be accomplished using tabs
   - i.e. `function func() { ↵ ⇥ ... ↵ }`
3. **Blank lines** SHOULD be added between logical blocks of code
   - i.e. `... ↵ ↵ ...`
4. **Text alignment** MUST be accomplished using spaces
   - i.e. `$var · · · = ...;`

5. Trailing whitespace MUST NOT be present after statements or serial comma break or on blank lines
   - i.e. no `...` `·` `·` `↲` `·` `↲` `...`
6. Keywords MUST be all lowercase
   - e.g. `false`, `true`, `null`, etc.
7. Variables MUST be all lowercase and words MUST be separated by an underscore
   - e.g. `$welcome_message`
8. Global variables MUST be declared one variable per line and MUST be indented after the first
   - e.g. `global $var1,` `↲` `⇥` `$var2;`
9. Constants MUST be all uppercase and words MUST be separated by an underscore
   - e.g. `WELCOME_MESSAGE`
10. Statements MUST be placed on their own line and MUST end with a semicolon
    - e.g. `welcome_message();`
11. Operators MUST be surrounded by a space
    - e.g. `$total = 15 + 7;`, `$var .= '';`
12. Unary operators MUST be attached to their variable or integer
    - e.g. `$index++`, `--$index`
13. Concatenation period MUST be surrounded by a space
    - e.g. `echo 'Read:' . $welcome_message;`
14. Single quotes MUST be used
    - e.g. `echo 'Hello, World!';`
15. Double quotes SHOULD NOT be used
    - e.g. `echo "Read: $welcome_message";` → `echo 'Read: ' . $welcome_message;`

▲ Table of Contents

1. Line Length

Line length MUST NOT exceed 80 characters, unless it is text.

✖ Incorrect

```
<?php

if(in_array('Slumdog Millionaire', $movies) && in_array('Silver Linings Playbook'
	// if body
}

// EOF
```

↳ Incorrect because expression exceeds 80 characters and should be refactored.

```php
<?php

$my_movies = array('Slumdog Millionaire', 'Silver Linings Playbook', 'The Lives c

// EOF
```

↳ Incorrect because arguments exceed 80 characters and should be placed on their own line.

~ Acceptable

```php
<?php

$text = 'Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec posuere r

// EOF
```

↳ Acceptable because line length was exceeded due to text, not code.

✓ Correct

```php
<?php

$my_movies = array(
	'Slumdog Millionaire',
	'Silver Linings Playbook',
	'The Lives of Others',
	'The Shawshank Redemption'
);

$has_all_movies = true;

foreach($my_movies as $my_movie) {
	if(!in_array($my_movie, $movies)) {
		$has_all_movies = false;
	}
}

if($has_all_movies) {
	// if body
}
```

```
$some_long_variable = get_something_else(
		'from_some_other_function',
		'another_long_argument'
);

// EOF
```

▲ Formatting

2. Line Indentation

Line indentation MUST be accomplished using tabs.

✖ Incorrect

```
<?php

function print_welcome_message() {
    echo WELCOME_MESSAGE;
}

// EOF
```

↳ Incorrect because spaces are used to indent `echo WELCOME_MESSAGE;` instead of a tab.

✔ Correct

```
<?php

function print_welcome_message() {
	echo WELCOME_MESSAGE;
}

// EOF
```

▲ Formatting

3. Blank Lines

Blank lines SHOULD be added between logical blocks of code.

~ Acceptable

```php
<?php

$my_movies = array(
	'Slumdog Millionaire',
	'Silver Linings Playbook',
	'The Lives of Others',
	'The Shawshank Redemption'
);
$has_all_movies = true;
foreach($my_movies as $my_movie) {
	if(!in_array($my_movie, $movies)) {
		$has_all_movies = false;
	}
}
if($has_all_movies) {
	// if body
}

// EOF
```

↳ Acceptable, but can make scanning code more difficult.

✔ **Preferred**

```php
<?php

$my_movies = array(
	'Slumdog Millionaire',
	'Silver Linings Playbook',
	'The Lives of Others',
	'The Shawshank Redemption'
);

$has_all_movies = true;

foreach($my_movies as $my_movie) {
	if(!in_array($my_movie, $movies)) {
		$has_all_movies = false;
	}
}

if($has_all_movies) {
	// if body
}

// EOF
```

▲ Formatting

4. Text Alignment

Text alignment MUST be accomplished using spaces.

✗ Incorrect

```
<?php

$movie_quotes = array(
		'slumdog_millionaire'		=> 'When somebody asks me a question, I t
		'silver_linings_playbook'	=> 'I opened up to you, and you judged me
		'the_lives_of_others'		=> 'To think that people like you ruled a
		'the_shawshank_redemption'	=> 'Get busy living, or get busy dying.'
);

// EOF
```

↳ Incorrect because tabs are used instead of spaces to vertically align `=>`.

```
<?php

$movie_quotes = array(
    'slumdog_millionaire'       => 'When somebody asks me a question, I tell them
    'silver_linings_playbook'   => 'I opened up to you, and you judged me.',
    'the_lives_of_others'       => 'To think that people like you ruled a country
    'the_shawshank_redemption'  => 'Get busy living, or get busy dying.'
);

// EOF
```

↳ Incorrect because spaces are used instead of tabs to indent array keys.

✓ Correct

```
<?php

$movie_quotes = array(
	'slumdog_millionaire'       => 'When somebody asks me a question, I tell
	'silver_linings_playbook'   => 'I opened up to you, and you judged me.',
	'the_lives_of_others'       => 'To think that people like you ruled a cou
	'the_shawshank_redemption'  => 'Get busy living, or get busy dying.'
);
```

```
// EOF
```

▲ Formatting

5. Trailing Whitespace

Trailing whitespace MUST NOT be present after statements or serial comma break or on blank lines.

✖ Incorrect

```php
<?php

$quotes_exist = false;  

print_welcome_message();

// EOF
```

↳ Incorrect because there are two spaces after `$quotes_exist = false;`.

```php
<?php

$my_movies = array(
	'Slumdog Millionaire', 
	'Silver Linings Playbook',
	'The Lives of Others',
	'The Shawshank Redemption'
);

// EOF
```

↳ Incorrect because there is a space after `,`.

```php
<?php

$quotes_exist = false;

print_welcome_message();

// EOF
```

↳ Incorrect because there are two spaces on the blank line below `$quotes_exist = false;` .

✔ Correct

```php
<?php

$quotes_exist = false;

print_welcome_message();

// EOF
```

```php
<?php

$my_movies = array(
        'Slumdog Millionaire',
        'Silver Linings Playbook',
        'The Lives of Others',
        'The Shawshank Redemption'
);

// EOF
```

▲ Formatting

6. Keywords

Keywords MUST be all lowercase.

✖ Incorrect

```php
<?php

$is_true = FALSE;
$is_false = TRUE:
$movie_quote = NULL;

// EOF
```

↳ Incorrect because `FALSE` , `TRUE` and `NULL` are not all lowercase.

✔ Correct

```php
<?php

$is_true = false;
$is_false = true:
$movie_quote = null;

// EOF
```

▲ Formatting

7. Variables

Variables MUST be all lowercase and words MUST be separated by an underscore.

❌ **Incorrect**

```php
<?php

$welcome_Message = '';
$Welcome_Message = '';
$WELCOME_MESSAGE = '';

// EOF
```

↳ Incorrect because `$welcome_Message` , `$Welcome_Message` and `$WELCOME_MESSAGE` are not all lowercase.

```php
<?php

$welcomemessage = '';

// EOF
```

↳ Incorrect because `welcome` and `message` are not separated with an underscore.

✔ **Correct**

```php
<?php

$welcome_message = '';
```

```
// EOF
```

▲ Formatting

8. Global Variables

Global variables MUST be declared one variable per line and MUST be indented after the first.

✖ Incorrect

```php
<?php

global $app_config, $cache, $db_connection;

// EOF
```

↳ Incorrect because `$app_config` , `$cache` and `$db_connection` are together on one line.

```php
<?php

global $app_config,
$cache,
$db_connection;

// EOF
```

↳ Incorrect because `$db_connection` and `$cache` are not indentend once.

✔ Correct

```php
<?php

global $app_config,
       $cache,
       $db_connection;

// EOF
```

▲ Formatting

9. Constants

Constants MUST be all uppercase and words MUST be separated by an underscore.

✖ Incorrect

```
<?php

define('welcome_Message', '');
define('Welcome_Message', '');
define('welcome_message', '');

// EOF
```

↳ Incorrect because `welcome_Message` , `Welcome_Message` and `welcome_message` are not all uppercase.

```
<?php

define('WELCOMEMESSAGE', '');

// EOF
```

↳ Incorrect because `WELCOME` and `MESSAGE` are not separated with an underscore.

✔ Correct

```
<?php

define('WELCOME_MESSAGE', '');

// EOF
```

▲ Formatting

10. Statements

Statements MUST be placed on their own line and MUST end with a semicolon.

✖ Incorrect

```
<?php
```

```
$quotes_exist = false; print_welcome_message();

// EOF
```

↳ Incorrect because `$quotes_exist = false;` and `print_welcome_message();` are on one line.

```
<div>
	<h1><?= print_welcome_message() ?></h1>
</div>
```

↳ Incorrect because `print_welcome_message()` is missing a semicolon.

✔ **Correct**

```
<?php

$quotes_exist = false;
print_welcome_message();

// EOF
```

```
<div>
	<h1><?= print_welcome_message(); ?></h1>
</div>
```

▲ Formatting

11. Operators

Operators MUST be surrounded a space.

✖ **Incorrect**

```
<?php

$total=3+14;
$string='Hello, World! ';
$string.='Today is a good day!';

// EOF
```

↳ Incorrect because there is no space surrounding the `=`, `+` or `.=` sign.

✔ **Correct**

```php
<?php

$total = 3 + 14;
$string = 'Hello, World! ';
$string .= 'Today is a good day!';

// EOF
```

▲ Formatting

12. Unary Operators

Unary operators MUST be attached to their variable or integer.

✖ **Incorrect**

```php
<?php

$index ++;
-- $index;

// EOF
```

↳ Incorrect because there is a space before `++` and after `--`.

✔ **Correct**

```php
<?php

$index++;
--$index;

// EOF
```

▲ Formatting

13. Concatenation Period

Concatenation period MUST be surrounded by a space.

✖ Incorrect

```php
<?php

echo 'Hello, World! Today is '.$date.'!';

// EOF
```

↳ Incorrect because there is no space surrounding `.`.

✔ Correct

```php
<?php

echo 'Hello, World! Today is ' . $date . '!';

// EOF
```

▲ Formatting

14. Single Quotes

Single quotes MUST be used.

✖ Incorrect

```php
<?php

echo "Hello, World!";

// EOF
```

↳ Incorrect because `"Hello, World!"` is not written with single quotes.

✔ Correct

```php
<?php

echo 'Hello, World!';

// EOF
```

▲ Formatting

15. Double Quotes

Double quotes SHOULD NOT be used.

~ Acceptable

```php
<?php

echo "Hello, World! Today is $date!";

// EOF
```

↳ Acceptable, but burries the `$date` variable, which is why single quotes are preferred.

```php
<?php

echo "Hello, World! He's watching movies and she's reading books.";

// EOF
```

↳ Acceptable when long pieces of text have apostrophies that would need to be escaped.

✔ Preferred

```php
<?php

echo 'Hello, World! Today is ' . $date . '!';

echo 'Hello, World! He\'s watching movies and she\'s reading books.';

// EOF
```

▲ Formatting

9. Functions

This section describes the format for function names, calls, arguments and declarations.

1. **Function name** MUST be all lowercase and words MUST be separated by an underscore
   - e.g. `function welcome_message() {`
2. **Function prefix** MUST start with verb
   - e.g. `get_`, `add_`, `update_`, `delete_`, `convert_`, etc.
3. **Function call** MUST NOT have a space between function name and open parenthesis
   - e.g. `func();`
4. **Function arguments**
   - MUST NOT have a space before the comma
   - MUST have a space after the comma
   - MAY use line breaks for long arguments
   - MUST then place each argument on its own line
   - MUST then indent each argument once
   - MUST be ordered from required to optional first
   - MUST be ordered from high to low importance second
   - MUST use descriptive defaults
   - MUST use type hinting
   - e.g. `func($arg1, $arg2 = 'asc', $arg3 = 100);`
5. **Function declaration** MUST be documented using phpDocumentor tag style and SHOULD include
   - Short description
   - Optional long description, if needed
   - @access: `private` or `protected` (assumed `public`)
   - @author: Author name
   - @global: Global variables function uses, if applicable
   - @param: Parameters with data type, variable name, and description
   - @return: Return data type, if applicable
6. **Function return**
   - MUST occur as early as possible
   - MUST be initialized prior at top
   - MUST be preceded by blank line, except inside control statement
   - i.e. `if (!$expr) { return false; }`

▲ Table of Contents

1. Function Name

Function name MUST be all lowercase and words MUST be separated by an underscore.

✖ **Incorrect**

```php
<?php

get_Welcome_Message();
Get_Welcome_Message();
GET_WELCOME_MESSAGE();

// EOF
```

↳ Incorrect because the function names are not all lowercase.

```php
<?php

getwelcomemessage();

// EOF
```

↳ Incorrect because `get`, `welcome` and `message` are not separated with an underscore.

✔ **Correct**

```php
<?php

get_welcome_message();

// EOF
```

▲ Functions

2. Function Prefix

Function prefix MUST start with verb.

✖ **Incorrect**

```php
<?php

active_users();
network_location($location1, $location2);
widget_form($id);
```

```
// EOF
```

↳ Incorrect because functions are not prefixed with a verb.

✓ Correct

```
<?php

get_active_users();
move_network_location($location1, $location2);
delete_widget_form($id);

// EOF
```

▲ Functions

3. Function Call

Function call MUST NOT have a space between function name and open parenthesis.

✗ Incorrect

```
<?php

print_welcome_message ();

// EOF
```

↳ Incorrect because there is a space between `get_welcome_message` and `()`.

✓ Correct

```
<?php

print_welcome_message();

// EOF
```

▲ Functions

4. Function Arguments

Function arguments:

- MUST NOT have a space before the comma
- MUST have a space after the comma
- MAY use line breaks for long arguments
- MUST then place each argument on its own line
- MUST then indent each argument once
- MUST be ordered from required to optional first
- MUST be ordered from high to low importance second
- MUST use descriptive defaults
- MUST use type hinting

✖ Incorrect

```php
<?php

my_function($arg1 , $arg2 , $arg3);

// EOF
```

↳ Incorrect because there is a space before `,`.

```php
<?php

my_function($arg1,$arg2,$arg3);

// EOF
```

↳ Incorrect because there is no space after `,`.

```php
<?php

my_other_function($arg1_with_a_really_long_name,
        $arg2_also_has_a_long_name,
        $arg3
);

// EOF
```

↳ Incorrect because `$arg1_with_a_really_long_name` is not on its own line.

```php
<?php

my_other_function(
$arg1_with_a_really_long_name,
$arg2_also_has_a_long_name,
$arg3
);

// EOF
```

↳ Incorrect because arguments are not indented once.

```php
<?php

function get_objects($type, $order = 'asc', $limit) {
	// ...
}

// EOF
```

↳ Incorrect because `$type` , `$order` and `$limit` are not in order of required to optional.

```php
<?php

function get_objects($limit, $order, $type) {
	// ...
}

// EOF
```

↳ Incorrect because `$limit` , `$order` and `$type` are not in order of importance.

```php
<?php

function get_objects($type, $order = true, $limit = 100) {
	// ...
}

// EOF
```

↳ Incorrect because `true` is not a descriptive default for `$order` .

```php
<?php

function add_users_to_office($users, $office) {
	// ...
}

// EOF
```

↳ Incorrect because `$users` and `$office` are missing their data type.

✔ Correct

```php
<?php

my_function($arg1, $arg2, $arg3);

my_other_function(
	$arg1_with_a_really_long_name,
	$arg2_also_has_a_long_name,
	$arg3
);

function get_objects($type, $order = 'asc', $limit = 100) {
	// ...
}

function add_users_to_office(array $users, Office $office) {
	// ...
}

// EOF
```

▲ Functions

5. Function Declaration

Function declaration MUST be documented using phpDocumentor tag style and SHOULD include:

- Short description
- Optional long description, if needed
- @access: `private` or `protected` (assumed `public`)
- @author: Author name
- @global: Global variables function uses, if applicable

- @param: Parameters with data type, variable name, and description
- @return: Return data type, if applicable

✖ Incorrect

```php
<?php

function my_function($id, $type, $width, $height) {
	// ...
	return $Photo;
}

// EOF
```

↳ Incorrect because `my_function` is not documented.

✔ Correct

```php
<?php

/**
 * Get photo from blog author
 *
 * Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut id volutpat
 * orci. Etiam pharetra eget turpis non ultrices. Pellentesque vitae risus
 * sagittis, vehicula massa eget, tincidunt ligula.
 *
 * @access private
 * @author Firstname Lastname
 * @global object $post
 * @param int $id Author ID
 * @param string $type Type of photo
 * @param int $width Photo width in px
 * @param int $height Photo height in px
 * @return object Photo
 */
function my_function($id, $type, $width, $height) {
	// ...
	return $Photo;
}

// EOF
```

▲ Functions

6. Function Return

Function return:

- MUST occur as early as possible
- MUST be initialized prior at top
- MUST be preceded by blank line, except inside control statement

✖ Incorrect

```php
<?php

function get_object() {
	$var = false;
	if($expr1) {
		// ...
		if($expr2) {
			// ...
		}
	}

	return $var;
}

// EOF
```

↳ Incorrect because `get_object` does not return as early as possible.

```php
<?php

function get_movies() {
	// ...

	return $movies;
}

// EOF
```

↳ Incorrect because `$movies` is not initialized at top.

```php
<?php

function get_movies() {
	$movies = array();
	// ...
	return $movies;
}
```

```
// EOF
```

↳ Incorrect because `return $movies` is not preceded by blank line.

✅ **Correct**

```php
<?php

function get_object() {
	$var = false;
	if (!$expr1) {
		return $var;
	}
	if (!$expr2) {
		return $var;
	}
	// ...

	return $var;
}

// EOF
```

```php
<?php

function get_movies() {
	$movies = array();
	// ...

	return $movies;
}

// EOF
```

▲ Functions

10. Control Structures

This section defines the layout and usage of control structures. Note that this section is separated into rules that are applicable to all structures, followed by specific rules for individual structures.

- **Keyword** MUST be followed by a space
  - e.g. `if (`, `switch (`, `do {`, `for (`

- **Opening parenthesis** MUST NOT be followed by a space
  - e.g. `($expr`, `($i`
- **Closing parenthesis** MUST NOT be preceded by a space
  - e.g. `$expr)`, `$i++)`, `$value)`
- **Opening brace** MUST be preceded by a space and MUST be followed by a new line
  - e.g. `$expr) {`, `$i++) {`
- **Structure body** MUST be indented once and MUST be enclosed with curly braces (no shorthand)
  - e.g. `if ($expr) { ↵ ⇥ ... ↵ }`
- **Closing brace** MUST start on the next line
  - i.e. `... ↵ }`
- **Nesting** MUST NOT exceed three levels
  - e.g. no `if ($expr1) { if ($expr2) { if ($expr3) { if ($expr4) { ... }}}}`

In addition to the rules above, some control structures have additional requirements:

1. If, Elseif, Else
   - `elseif` MUST be used instead of `else if`
   - `elseif` and `else` MUST be between `}` and `{` on one line
2. Switch, Case
   - Case statement MUST be indented once
     - i.e. `⇥ case 1:`
   - Case body MUST be indented twice
     - i.e. `⇥ ⇥ func();`
   - Break keyword MUST be indented twice
     - i.e. `⇥ ⇥ break;`
   - Case logic MUST be separated by one blank line
     - i.e. `case 1: ... break; ↵ ↵ case 2: ... break;`
3. While, Do While
4. For, Foreach
5. Try, Catch
   - `catch` MUST be between `}` and `{` on one line

▲ Table of Contents

1. If, Elseif, Else

- `elseif` MUST be used instead of `else if`
- `elseif` and `else` MUST be between `}` and `{` on one line

❌ **Incorrect**

```php
<?php

if ($expr1) {
	// if body
} else if ($expr2) {
	// elseif body
} else {
	// else body
}

// EOF
```

↳ Incorrect because `else if` was used instead of `elseif`.

```php
<?php

if ($expr1) {
	// if body
}
elseif ($expr2) {
	// elseif body
}
else {
	// else body
}

// EOF
```

↳ Incorrect because `elseif` and `else` are not between `}` and `{` on one line.

```php
<?php

$result1 = if ($expr1) ? true : false;

if($expr2)
	$result2 = true;

// EOF
```

↳ Incorrect because structure body is not wrapped in curly braces.

✅ **Correct**

```php
<?php

if ($expr1) {
	// if body
} elseif ($expr2) {
	// elseif body
} else {
	// else body
}

if ($expr1) {
	$result1 = true;
} else {
	$result1 = false;
}

if ($expr2) {
	$result2 = true;
}

// EOF
```

▲ Control Structures

2. Switch, Case

- Case statement MUST be indented once
- Case body MUST be indented twice
- Break keyword MUST be indented twice
- Case logic MUST be separated by one blank line

✖ Incorrect

```php
<?php

switch ($expr) {
case 0:
	echo 'First case, with a break';
	break;

case 1:
	echo 'Second case, which falls through';
	// no break
case 2:
case 3:
case 4:
	echo 'Third case, return instead of break';
```

```
		return;

	default:
		echo 'Default case';
		break;
}

// EOF
```

↳ Incorrect because `case 0` thru `default` are not indented once.

```php
<?php

switch ($expr) {
	case 0:
	echo 'First case, with a break';
	break;

	case 1:
	echo 'Second case, which falls through';
	// no break
	case 2:
	case 3:
	case 4:
	echo 'Third case, return instead of break';
	return;

	default:
	echo 'Default case';
	break;
}

// EOF
```

↳ Incorrect because `echo` , `break` and `return` are not indented twice.

```php
<?php

switch ($expr) {
	case 0:
		echo 'First case, with a break';
		break;
	case 1:
		echo 'Second case, which falls through';
		// no break
	case 2:
	case 3:
	case 4:
```

```
                echo 'Third case, return instead of break';
                return;
        default:
                echo 'Default case';
                break;
}

// EOF
```

↳ Incorrect because `case 0`, `case 1` thru `case 4`, and `default` are not separated by one blank line.

✔ **Correct**

```
<?php

switch ($expr) {
        case 0:
                echo 'First case, with a break';
                break;

        case 1:
                echo 'Second case, which falls through';
                // no break
        case 2:
        case 3:
        case 4:
                echo 'Third case, return instead of break';
                return;

        default:
                echo 'Default case';
                break;
}

// EOF
```

▲ Control Structures

3. While, Do While

✔ **Correct**

```
<?php

while ($expr) {
		// structure body
}

do {
		// structure body;
} while ($expr);

// EOF
```

▲ Control Structures

4. For, Foreach

✔ **Correct**

```
<?php

for ($i = 0; $i < 10; $i++) {
		// for body
}

foreach ($iterable as $key => $value) {
		// foreach body
}

// EOF
```

▲ Control Structures

5. Try, Catch

✖ **Incorrect**

```
<?php

try {
		// try body
}
catch (FirstExceptionType $e) {
		// catch body
}
catch (OtherExceptionType $e) {
		// catch body
}
```

```
// EOF
```

↳ Incorrect because `catch` is not between `}` and `{` on one line.

✔ Correct

```php
<?php

try {
	// try body
} catch (FirstExceptionType $e) {
	// catch body
} catch (OtherExceptionType $e) {
	// catch body
}

// EOF
```

▲ Control Structures

11. Classes

This section describes class files, names, definitions, properties, methods and instantiation.

1. **Class file** MUST only contain one definition and MUST be prefixed with `class-`
    - i.e. `class User` → `class-user.php`, `class Office` → `class-office.php`
2. **Class namespace** MUST be defined and MUST include vendor name
    - e.g. `namespace MyCompany\Model;`, `namespace MyCompany\View;`, `namespace MyCompany\Controller;`
3. **Class name** MUST start with a capital letter and MUST be camelcase
    - e.g. `MyCompany`
4. **Class documentation** MUST be present and MUST use phpDocumentor tag style
    - i.e. `@author`, `@global`, `@package`
5. **Class definition** MUST place curly braces on their own line
    - i.e. `class User ↵ { ↵ ... ↵ }`
6. **Class properties**
    - MUST follow variable standards
    - MUST specify visibility
    - MUST NOT be prefixed with an underscore if private or protected
    - e.g. `$var1;`, `private $var2;`, `protected $var3;`

7. Class methods
    - MUST follow function standards
    - MUST specify visibility
    - MUST NOT be prefixed with an underscore if private or protected
    - e.g. `func1()` , `private func2()` , `protected func3()`
8. Class instance
    - MUST start with capital letter
    - MUST be camelcase
    - MUST include parenthesis
    - e.g. `$user = new User();` , `$OfficeProgram = new OfficeProgram();`

▲ Table of Contents

1. Class File

Class file MUST only contain one definition and MUST be prefixed with `class-` .

✖ Incorrect

Filename: `class-user.php`

```
<?php

namespace MyCompany\Model;

class User
{
	// ...
}

class Office
{
	// ...
}

// EOF
```

↳ Incorrect because `User` and `Office` are defined in one file.

Filename: `user.php`

```
<?php

namespace MyCompany\Model;
```

```
class User
{
	// ...
}

// EOF
```

↳ Incorrect because filename is not prefixed with `class-` .

✔ Correct

Filename: `class-user.php`

```
<?php

namespace MyCompany\Model;

class User
{
	// ...
}

// EOF
```

Filename: `class-office.php`

```
<?php

namespace MyCompany\Model;

class Office
{
	// ...
}

// EOF
```

▲ Classes

2. Class Namespace

Class namespace MUST be defined and MUST include vendor name.

✖ Incorrect

```php
<?php

class User
{
	// ...
}

// EOF
```

↳ Incorrect because there is no namespace defined.

```php
<?php

namespace Model;

class User
{
	// ...
}

// EOF
```

↳ Incorrect because vendor name is missing in the namespace name.

✔ **Correct**

```php
<?php

namespace MyCompany\Model;

class User
{
	// ...
}

// EOF
```

▲ Classes

3. Class Name

Class name MUST start with a capital letter and MUST be camelcase.

✖ **Incorrect**

```php
<?php

namespace MyCompany\Model;

class officeProgram
{
        // ...
}

// EOF
```

↳ Incorrect because `officeProgram` does not start with a capital letter.

```php
<?php

namespace MyCompany\Model;

class Officeprogram
{
        // ...
}

// EOF
```

↳ Incorrect because `Officeprogram` is not camelcase.

✔ **Correct**

```php
<?php

namespace MyCompany\Model;

class OfficeProgram
{
        // ...
}

// EOF
```

▲ Classes

4. Class Documentation

Class documentation MUST be present and MUST use phpDocumentor tag style.

✖ Incorrect

```php
<?php

namespace MyCompany\Model;

class User
{
	// ...
}

// EOF
```

↳ Incorrect because `User` is missing documentation.

```php
<?php

namespace MyCompany\View;

/**
 * User View
 */
class User
{
	// ...
}

// EOF
```

↳ Incorrect because `User` is missing phpDocumentor tags.

✔ Correct

```php
<?php

namespace MyCompany\View;

/**
 * User View
 *
 * @author Firstname Lastname
 * @global object $post
 * @package MyCompany\API
 */
class User
{
	// ...
}

// EOF
```

▲ Classes

5. Class Definition

Class definition MUST place curly braces on their own line.

❌ **Incorrect**

```
<?php

namespace MyCompany\Model;

class User {
	// ...
}

// EOF
```

↳ Incorrect because `{` is not on its own line.

✔ **Correct**

```
<?php

namespace MyCompany\Model;

class User
{
	// ...
}

// EOF
```

▲ Classes

6. Class Properties

Class properties:

- MUST follow variable standards

- MUST specify visibility
- MUST NOT be prefixed with an underscore if private or protected

✗ Incorrect

```php
<?php

namespace MyCompany\Model;

class User
{
	// Public
	$var1;

	// Protected
	$var2;

	// Private
	$var3;
}

// EOF
```

↳ Incorrect because visibility is not specified for `$var1`, `$var2` and `$var3`.

```php
<?php

namespace MyCompany\Model;

class User
{
	public $var1;
	protected $_var2;
	private $_var3;
}

// EOF
```

↳ Incorrect because `protected` and `private` properties are prefixed with `_`.

✓ Correct

```php
<?php

namespace MyCompany\Model;
```

```
class User
{
	public $var1;
	protected $var2;
	private $var3;
}

// EOF
```

▲ Classes

7. Class Methods

Class methods:

- MUST follow function standards
- MUST specify visibility
- MUST NOT be prefixed with an underscore if private or protected

✖ **Incorrect**

```
<?php

namespace MyCompany\Model;

class User
{
	// ...

	// Public
	function get_var1() {
		return $this->var1;
	}

	// Protected
	function get_var2() {
		return $this->var2;
	}

	// Private
	function get_var3() {
		return $this->var3;
	}
}

// EOF
```

↳ Incorrect because visibility is not specified for `get_var1()`, `get_var2()` and `get_var3()`.

```php
<?php

namespace MyCompany\Model;

class User
{
	// ...

	public function get_var1() {
		return $this->var1;
	}

	protected function _get_var2() {
		return $this->var2;
	}

	private function _get_var3() {
		return $this->var3;
	}
}

// EOF
```

↳ Incorrect because `protected` and `private` methods are prefixed with `_`.

✔ **Correct**

```php
<?php

namespace MyCompany\Model;

class User
{
	// ...

	public function get_var1() {
		return $this->var1;
	}

	protected function get_var2() {
		return $this->var2;
	}

	private function get_var3() {
		return $this->var3;
	}
}

// EOF
```

▲ Classes

8. Class Instance

Class instance:

- MUST follow variable standards
- MUST include parenthesis

❌ **Incorrect**

```
<?php

$office_program = new OfficeProgram;

// EOF
```

↳ Incorrect because `new OfficeProgram` is missing parenthesis.

✔ **Correct**

```
<?php

$office_program = new OfficeProgram();

// EOF
```

▲ Classes

12. Best Practices

1. **Variable initialization** SHOULD occur prior to use and SHOULD occur early
   - e.g. `$var1 = '';` , `$var2 = 0;`
2. **Initialization/declaration order**
   - MUST lead with globals, follow with constants, conclude with local variables
   - MUST lead with properties and follow with methods in classes

   - MUST lead with `public`, follow with `protected`, conclude with `private` methods in classes
   - SHOULD be alphabetical within their type
   - i.e. `global $var1;`, `define('VAR2', '');`, `$var3 = 0;`
3. Globals SHOULD NOT be used
   - i.e. no `global $var;`
4. Explicit expressions SHOULD be used
   - e.g. `if ($expr === false)`, `while ($expr !== true)`
5. E_STRICT reporting MUST NOT trigger errors
   - i.e. do not use deprecated functions, etc.

▲ Table of Contents

1. Variable Initialization

Variable initialization SHOULD occur prior to use and SHOULD occur early.

~ Acceptable

```
<?php

$movies = get_movies();

// EOF
```

↳ Acceptable, but `$movies` should be initialized prior to use.

```
<?php

if ($expr) {
	// ....
}

$movies = array();
$movies = get_movies();

// EOF
```

↳ Acceptable, but `$movies` should be initialized earlier.

✓ Preferred

```php
<?php

$movies = array();

if ($expr) {
        // ....
}

$movies = get_movies();

// EOF
```

▲ Best Practices

2. Initialization/Declaration Order

Initialization/declaration order:

- MUST lead with globals, follow with constants, conclude with local variables
- MUST lead with properties and follow with methods in classes
- MUST lead with `public` , follow with `protected` , conclude with `private` methods in classes
- SHOULD be alphabetical within their type

✖ Incorrect

```php
<?php

define('ENVIRONMENT', 'PRODUCTION');

$id = 0;

global $app_config;

// EOF
```

↳ Incorrect because `$app_config` is not first, `ENVIRONMENT` not second, and `$id` not third.

```php
<?php

namespace MyCompany\Model;

class Office
```

```
{
	public function get_name() {
		// ...
	}

	private $name;
}

// EOF
```

↳ Incorrect because `get_name()` is declared before `$name` .

```
<?php

namespace MyCompany\Model;

class Office
{
	private $id;
	private $name;
	private $status;

	private function get_name() {
		// ...
	}

	public function get_id() {
		// ...
	}

	protected function get_status() {
		// ...
	}
}

// EOF
```

↳ Incorrect because `get_id()` is not first, `get_status()` not second, and `get_name()` not third.

~ **Acceptable**

```
<?php

global $db_connection,
	$app_config,
	$cache;
```

```php
define('MODE', 1);
define('ENVIRONMENT', 'PRODUCTION');

$id = 0;
$firstname = '';
$lastname = '';

// EOF
```

↳ Acceptable, but the globals and constants should be in alphabetical order.

```php
<?php

function get_movies() {
	// ...
}

function get_actors() {
	// ...
}

// EOF
```

↳ Acceptable, but `get_actors` should be declared before `get_movies`.

✔ **Correct**

```php
<?php

global $app_config,
	$cache,
	$db_connection;

define('ENVIRONMENT', 'PRODUCTION');
define('MODE', 1);

$id = 0;
$firstname = '';
$lastname = '';

// EOF
```

```php
<?php
```

```php
namespace MyCompany\Model;

class Office
{
	private $id;
	private $name;
	private $status;

	public function get_id() {
		// ...
	}

	protected function get_status() {
		// ...
	}

	private function get_name() {
		// ...
	}
}

// EOF
```

✔ Preferred

```php
<?php

function get_actors() {
	// ...
}

function get_movies() {
	// ...
}

// EOF
```

▲ Best Practices

3. Globals

Globals SHOULD NOT be used.

~ Acceptable

```php
<?php
```

```
$pdo = new PDO('mysql:host=localhost;dbname=test', $user, $pass);

function get_user($id) {
	global $pdo;
	// ...
}

// EOF
```

↳ Acceptable, but `global` variables should be avoided.

✔ **Preferred**

```
<?php

function get_database_object() {
	return new PDO('mysql:host=localhost;dbname=test', $user, $pass);
}

function get_user($id) {
	$pdo = get_database_object();
	// ...
}

// EOF
```

▲ Best Practices

4. Explicit Expressions

Explicit expressions SHOULD be used.

~ Acceptable

```
<?php

if ($expr == true) {
	// ...
}

// EOF
```

↳ Acceptable, but `===` could be used here instead.

✓ Preferred

```
<?php

if ($expr === true) {
	// ...
}

// EOF
```

▲ Best Practices

5. E_STRICT Reporting

E_STRICT reporting MUST NOT trigger errors.

✗ Incorrect

```
<?php

$firstname = call_user_method('get_firstname', $User);

// EOF
```

↳ Incorrect because `call_user_method` (deprecated) will cause E_STRICT warning.

✓ Correct

```
<?php

$firstname = call_user_func(array($User, 'get_firstname'));

// EOF
```

▲ Best Practices

▲ Table of Contents

Inspired in part by style guides from:
CodeIgniter, Drupal, Horde, Pear, PSR-1, PSR-2, Symfony, and WordPress.

iyawa-revo commented on Aug 16, 2018

Thank a lot for this great work

andrewkimbowa commented on Aug 28, 2018 • edited ▾

Thank you very much for this work.

shamimmoeen commented on Sep 1, 2018

👍

joemaller commented on Sep 24, 2018 • edited ▾

Thank you for this. One note: Semicolons are redundant in short-echo tags since closing php tags `?>` imply a semicolon. ref

ItsJustATypo commented on Jan 26, 2019 • edited ▾

This guide is helpful. I think there's a typo in section 8/Formatting, rule 10/Statements, The rule states:

> 10. Statements
>   Statements MUST be placed on their own line and **MUST end with a semicolon**.

and gives this as an example as being **INCORRECT** because *it's missing a semicolon*:

```
<div>
<h1><?= print_welcome_message() ?></h1>
</div>
↳ Incorrect because print_welcome_message() is missing a semicolon.
```

but the example given as **CORRECT** is *also missing a semicolon*:

```
<h1><?= print_welcome_message() ?></h1>
```

Following the rule, it should be:

```
<h1><?= print_welcome_message(); ?></h1>
```

hwmatthewa commented on Dec 14, 2019 • edited ▾

Function name MUST be all lowercase and words MUST be separated by an underscore
e.g. function welcome_message() {

This directly contradicts PSR1:
'Method names MUST be declared in camelCase.' https://www.php-fig.org/psr/psr-1/#1-overview

Edit: This is not actually true. Methods and functions are different. Methods are on classes, and as such, have separate naming conventions.

ypicot commented on May 4, 2020

Great work, with usefull examples.
One small correction : acording to PSR-12, "Code MUST use an indent of 4 spaces for each indent level, and MUST NOT use tabs for indenting."
https://www.php-fig.org/psr/psr-12/#24-indenting

arcanisgk commented on Jun 20, 2020

> Great work, with usefull examples.
> One small correction : acording to PSR-12, "Code MUST use an indent of 4 spaces for each indent level, and MUST NOT use tabs for indenting."
> https://www.php-fig.org/psr/psr-12/#24-indenting

hey, then what do we do; we follow the psr-12; or we omit it; and continue to use references from different standards;

It would be convenient to add the reference at the bottom of each Guideline:

REF-> Use Reference from PSR-1.
REF-> Use Reference from Sinfony Recommendation.
REF-> Use Reference from Wordpress Recommendation.

Instead of putting it at the end:

> Inspired in part by style guides from:
> CodeIgniter, Drupal, Horde, Pear, PSR-1, PSR-2, Symfony, and WordPress.

notnian commented on Jun 26, 2020

What a nice guide 😃 👍

But tell me why using `// EOF` comment at the end of the file is a great thing ?

lutfiddin-ux commented on Jan 14, 2021

that was a great help, thanks a lot

kuroyza commented on May 8, 2021

Thank you so much for sharing this great guide 👍👍

arcanisgk commented on May 8, 2021

I think this should already have an update. As good as it is, it was last updated in 2014 (it's already been 7 years wow) ...

ryansechrest commented on May 8, 2021

Oh yeah, this is very old, and I've changed my mind on a few things since I wrote this. 🤣

kurahaupo commented on Jun 13, 2021 • edited

@hwmatthewa
(function names must be lower_snake_case)

> This directly contradicts PSR1:
> 'Method names MUST be declared in camelCase.' https://www.php-fig.org/psr/psr-1/#1-overview

Style guides don't all have to be the same; if they were, there would be no point, "the one true way" would simply be part of the language specification.

The purpose of a style guide is to make the code easier to read for its intended audience. For people who are accustomed to say, Java, camelCase makes perfect sense, but to people accustomed to say, traditional C, it looks ugly-crazy-awful.

It makes more sense to have the same guideline for choosing names used across all the languages in a project; that's the reason you find some C projects written using camelCase (when traditionally it's snake_case), and some PHP projects using snake_case (when PSR says to use camelCase).

The important thing is that a project *has* a style guide, and that the code follows it.

kurahaupo commented on Jun 13, 2021 • edited

@notnian because PSR says so, but I'm blowed if I know why it does either.

Maybe they're worried about files being truncated? Seems like a weird way to 'fix' that though.

If you **really** want a syntactic closure at the end of each file, put a {} block around the whole file, and simply say "don't put anything after the last closing "}".

hwmatthewa commented on Jun 14, 2021

@kurahaupo When I initially wrote that comment two years ago, I misunderstood that they were referring to functions outside of classes and ones defined on classes - methods. With that context, my comment is irrelevant.

sontranduc commented on Apr 15, 2022

People still use camelcase for variables