Horus R&D Internship

**Coding Standards**

For PHP and JavaScript

horizontal line

# 

# Introduction

Markdown format is a lightweight markup language with plain text formatting syntax, designed to be easily converted to HTML and other formats. It is widely used for documentation, readme files, and writing messages in online forums due to its readability in its raw form. Coding standards, on the other hand, are a set of guidelines and best practices that ensure code consistency, readability, and maintainability across a development team. Adopting these standards can significantly enhance the quality of the codebase, making it easier for developers to understand, review, and collaborate on projects. This introduction will cover the importance of coding standards and provide an overview of best practices for PHP and JavaScript.

## Markdown Format

1. **Headers**: Use # for headers. The number of # signs indicates the level of the header.

# Header 1 ## Header 2 ### Header 3

1. **Emphasis**: Use \* or \_ for italics and \*\* or \_\_ for bold.

\*italic\* or \_italic\_

\*\*bold\*\* or \_\_bold\_\_

1. **Lists**: Use -, \*, or + for unordered lists, and numbers for ordered lists.

- Item 1

- Item 2

- Item 3

1. First item

2. Second item

3. Third item

1. **Links**: Use [link text](URL) for links.

[OpenAI](https://www.openai.com)

1. **Images**: Similar to links but with an exclamation mark in front.

![alt text](URL)

1. **Code**: Use backticks for inline code and triple backticks for code blocks.

`inline code`

1. **Blockquotes**: Use > for blockquotes.

> This is a blockquote.

1. **Tables**: Use pipes and hyphens to create tables.

| Header 1 | Header 2 |

| -------- | -------- |

| Cell 1 | Cell 2 |

| Cell 3 | Cell 4 |

## 

## Importance of Coding Standards

1. Consistency: Standards ensure that all developers working on the codebase maintain the same style.
2. Readability: Important characteristic of the code base and makes it easier for developers to join-in on the code.
3. Maintainability: It is easier to update, debug and extend the codebase.
4. Quality: Adhering to best practices helps in producing higher quality code with fewer bugs and errors.

## PHP Standards

-Opening and Closing Tags:

When embedding multi-line PHP snippets, the opening and closing tags must be on separate lines by themselves.

Ex:

function foo() {

**?>**

<div>

**<?php**

echo esc\_html(

bar(

$baz,

$bat

)

);

**?>**

</div>

**<?php**

}

-Never Use Shorthand PHP Start Tags:

Shorthand tags are considered bad practice because they are not XML compliant and are less portable and are not guaranteed to work for every setup

Ex (Correct) :

**<?php** ... **?>**

**<?php** echo esc\_html( $var ); **?>**

Ex (Incorrect) :

**<? ... ?>**

**<?= esc\_html( $var ) ?>**

-Single and Double Quotes:

If you are not evaluating any expression in the string, use single quotes You should almost never have to escape quotes in a string, because you can just alternate your quoting style, like so:

echo **'<a href="/static/link" class="button button-primary">Link name</a>';**

echo **"<a href='{$escaped\_link}'>text with a ' single quote</a>";**

Text that goes into HTML or XML attributes should be escaped so that single or double quotes do not end the attribute value and invalidate the HTML, causing a security issue.

-Naming Conventions:

Use lowercase in variable and function names, separate words using underscores.

function some\_name( $some\_variable ) {}

For Classes, Traits, Interfaces and Enums, use capitalized words separated by underscores. Acronyms should all be uppercase.

class Walker\_Category extends Walker {}

class WP\_HTTP {}

interface Mailer\_Interface {}

trait Forbid\_Dynamic\_Properties {}

enum Post\_Status {}

Constants are uppercase with underscore word separation.

define( 'DOING\_AJAX', true );

Files are named with lowercase letters and words are separated by hyphens.

my-plugin-name.php

Class file names should be based on the class name with class- prepended and the underscores in the class name replaced with hyphens, for example, WP\_Error becomes:

class-wp-error.php

-Whitespace:

Put spaces after commas, and on both sides of arithmetic, logical, comparison and assignment operators, as well as on both sides of opening and closing parentheses of control structure blocks.

SOME\_CONST === 23;

foo() && bar();

! $foo;

array( 1, 2, 3 );

$baz . '-5';

$term .= 'X';

if ( $object instanceof Post\_Type\_Interface ) {}

$result = 2 \*\* 3; // 8.

foreach ( $foo as $bar ) { …

In a switch block, there must be no space between the case condition and the colon.

switch ( $foo ) {

case 'bar': // Correct.

case 'ba' : // Incorrect.

}

-Formatting:

Braces for all blocks shall be used as shown

if ( condition ) {

action1();

action2();

} elseif ( condition2 && condition3 ) {

action3();

action4();

} else {

defaultaction();

}

When splitting a function call into multiple lines, each parameter must be on a separate line and must occupy no more than a single line.

Multi-line parameter values must be assigned to a variable first then the variable is passed through a function call.

$bar = array(

'use\_this' => true,

'meta\_key' => 'field\_name',

);

$baz = sprintf(

/\* translators: %s: Friend's name \*/

\_\_( 'Hello, %s!', 'yourtextdomain' ),

$friend\_name

);

$a = foo(

$bar,

$baz,

/\* translators: %s: cat \*/

sprintf( \_\_( 'The best pet is a %s.' ), 'cat' )

);

## JavaScript Standards

-Naming Conventions:

Using full words and camel casing. Variable names should be descriptive but not excessively so.

Acronyms must be written with all of their composing letters capitalized, abbreviations on the other hand respect the camel case rule.

For Classes, pascal casing is used.

// "Id" is an abbreviation of "Identifier":

const userId = 1;

// "DOM" is an acronym of "Document Object Model":

const currentDOMDocument = window.document;

// Acronyms and abbreviations at the start of a variable name are consistent

// with camelcase rules covering the first letter of a variable or class.

const domDocument = window.document;

class DOMDocument {}

class IdCollection {}

Constants, which are never to be mutates or reassigned are an exception to the camel case rule and use SCREAMING\_SNAKE\_CASE convention.

-Comments:

Comments must come before the code to which they refer and should always be preceded by blank lines. There must also a space between the comment token ( // ) and the text. Capitalize the first letter of the comment.

someStatement();

// Explanation of something complex on the next line

$( 'p' ).doSomething();

// This is a comment that is long enough to warrant being stretched

// over the span of multiple lines.

JSDoc comments use the /\*\* multi-line comment opening in reference to the JS documentation standards.

-Strings:

Use single quotes for string literals:

let myStr = 'strings should be contained in single quotes';

When the string itself contains single quotes, they should be escaped with a backslash:

// Escape single quotes within strings:

'Note the backslash before the \'single quotes\'';

-Arrays:

Creation of arrays should be done using the shorthand [ ] rather than using the Array() constructor notation.

let myArray = [];

You may of course, initialize the array during construction:

let myArray = [ 1, 'Apple', 2, 'Bananas' ];

-Objects:

Creation of objects is preferably done using the { } notation because it is the most performant, it is also the easiest to read.

let myObj = {};

-Semicolons:

Always use them. Never rely on Automatic Semicolon Insertion (ASI).

-Chained Function Calls:

If the chain of method calls is too long, split the calls into separate lines for each call. If the method changes the context, an extra level of indentation must be used.

elements

.addClass( 'foo' )

.children()

.html( 'hello' )

.end()

.appendTo( 'body' );

-Iteration:

It is recommended to store the loop’s max value as opposed to recalculating it every iteration:

// Good & Efficient

var i, max;

// getItemCount() gets called once

for ( i = 0, max = getItemCount(); i < max; i++ ) {

// Do stuff

}

// Bad & Potentially Inefficient:

// getItemCount() gets called every time

for ( i = 0; i < getItemCount(); i++ ) {

// Do stuff

}

# Conclusion

Each language has different coding standards and best practices. It is important to learn them especially when collaborating on a large codebase to minimize bugs and maintain clean, readable code.

# References

1. WordPress.org. (n.d.). *WordPress coding standards: JavaScript*. Retrieved June 27, 2024, from<https://developer.wordpress.org/coding-standards/wordpress-coding-standards/javascript/>
2. WordPress.org. (n.d.). *WordPress coding standards: PHP*. Retrieved June 27, 2024, from<https://developer.wordpress.org/coding-standards/wordpress-coding-standards/php/>