PHP7 +PSR2

1. Do Not Use mysql_ Functions

The time has finally come when you won't just be advised to stop using mysql_ functions. PHP 7 will remove them altogether from core which means you'll need to move to the far better mysqli_ functions, or the even more flexible PDO implementation.

2. Do Not Write Wasteful Code

This one may be a no-brainer but it will become increasingly important because the speed increases in PHP 7 may hide some of your issues. Don't be content with your site speed simply because the switch to PHP 7 made it faster.
To understand just how important speed is and what you can do to make things better, take a look at our **Beginners' guide to speed optimization** article.
As developers you should always make sure to load scripts only when they are needed, concatenate them when possible, write efficient database queries, use caching when possible and so on.

3. Do Not Use PHP Close Tags At The End Of A file

It is not required by PHP and by omitting it at the end of a file you are making sure that no trailing whitespace can be added.

4. Do Not Pass By Reference If Not Needed

I personally don't like passing by reference. I understand that in some cases it is useful, but in many others it makes code harder to understand and follow and especially difficult to predict the result.
Apparently people think it makes their code faster though which according to respectable PHP programmers is just not true.
One example of why references are bad is PHP built in shuffle() or sort(). Instead of returning a shuffled or sorted array, they modify the original which is completely illogical to my mind.

5. Do Not Perform Queries In A Loop

Performing database queries in a loop is just wasteful. It puts unnecessary strain on your systems and it is likely you can achieve the same result faster outside the loop. When I bump into a situation where this would be needed I can usually solve the issue with two separate queries I use to build an array of data. I then loop over the array, no need to perform queries in the process.
Due to the way WordPress works there may be some exceptions to this. While get_post_meta() will grab a meta value from the database, you can use it in a loop if you're looping through one specific post's metadata. This is because when you use it for the first time WordPress actually retrieves all metadata and caches it. Subsequent calls use the cached data, not database calls.
The best way to work these things out is to read function documentations and to use something like the Query Monitor.

6. Do Not Use * In SQL Queries

All right, this one is more of a MySQL issue, but we tend to write our SQL code in PHP so I say it's fair game. In any case, don't use wildcards in SQL queries if

you can avoid them, especially if you have a database with a lot of columns. Specify the exact columns you need and only retrieve those. This helps minimize your resource usage, protect your data and make things as clear as possible.
While on the subject of SQL, know your available functions and test for speed as much as possible. When calculating averages, sums or similar numbers use SQL functions instead of PHP functions. If you are unsure of the speed of a query test it and try some other variations – use the best one.

7. Do Not Trust User Input

It is not wise to trust user input. Always filter, sanitize, escape, check and use fallbacks. There are three issues with user data: we developers don't take every possibility into account, it is frequently incorrect and it may be intentionally malicious.
A well thought out system can protect against all of these. Make sure to use built in functions like filter_var() to check for proper values and escaping and other functions when working with databases.
WordPress has a bunch of functions to help you out. Take a look at the Validating, escaping and sanitising user data article for more information.

8. Do Not Try To Be Clever

Your goal should be to write elegant code that expresses your intentions the most clearly. You may be able to shave off an extra 0.01 second off each page load by shortening everything to one letter variables, using multi-level ternary logic and other cleverness but this really is nothing compared to the headaches you will be causing yourself and everyone else around you.
Name your variables appropriately, document your code, choose clarity over brevity. Even better, use standardized object oriented code which more or less documents itself without the need for a lot of inline comments.

9. Do Not Reinvent The Wheel

PHP has been around for a long while now, websites have been made for even longer. Chances are that whatever you need to make, someone has made before. Don't be afraid to lean on others for support, Github is your friend, Composer is your friend, Packagist is your friend.
From loggers to color manipulation tools, from profilers to unit testing frameworks, from Mailchimp APIs to Twitter Bootstrap, everything is available at the push of a button (or typing of a command), use them!

10. Do Not Neglect Other Languages

use null coalesces with encapsulation only:

($var??'')== 'val' not **$var??''== 'val'**

New in PHP 7: null coalesce operator

Not the catchiest name for an operator, but PHP 7 brings in the rather handy null coalesce so I thought I'd share an example.

In PHP 5, we already have a ternary operator, which tests a value, and then returns the second element if that returns true and the third if it doesn't:

echo $count ? $count : 10; // outputs 10

There is also a shorthand for that which allows you to skip the second element if it's the same as the first one:

echo $count ?: 10; // also outputs 10
In PHP 7 we additionally get the ?? operator which rather than indicating extreme confusion which is how I would usually use two question marks together instead allows us to chain together a string of values. Reading from left to right, the first value which exists and is not null is the value that will be returned.
// $a is not set
$b = 16;

echo $a ?? 2; // outputs 2
echo $a ?? $b ?? 7; // outputs 16
This construct is useful for giving priority to one or more values coming perhaps from user input or existing configuration, and safely falling back on a given default if that configuration is missing. It's kind of a small feature but it's one that I know I'll be using as soon as my applications upgrade to PHP 7.

PSR2:

Control structure keywords:

- if
- else
- elseif/else if
- Alternative syntax for control structures
- while
- do-while
- for
- foreach
- break
- continue
- switch
- declare
- return
- require
- include
- require_once

MUST have one space after them; method and function calls MUST NOT.