Engineer

PHP

the language most of the old web runs on

What it is

PHP is a scripting language that executes on the web server: a request arrives, the PHP code runs, and what reaches the browser is the HTML it produced, never the code itself. The letters now stand for “PHP: Hypertext Preprocessor”, a recursive acronym, though they began as “Personal Home Page”. A PHP file can be ordinary HTML with code placed between `<?php` and `?>` tags, or it can be pure code that a framework routes every request through. It also runs scripts from the command line, but its design leans towards producing web pages.

Its history starts in 1994, when Rasmus Lerdorf wrote a set of CGI binaries in C to count visits to his online résumé and called them Personal Home Page Tools. He released the source in June 1995, and the tools grew into PHP/FI. In 1997 Andi Gutmans and Zeev Suraski began rewriting the parser, and the result was released as PHP 3.0 in June 1998 under the recursive name; at its peak it was installed on roughly 10% of the web servers on the internet. Their next engine, the Zend Engine, named after the two of them, carried PHP 4.0 in May 2000, and PHP 5 followed in July 2004.

On a server today PHP is commonly run by PHP-FPM, the FastCGI Process Manager that ships with it. A web server in front passes each PHP request to a pool of worker processes; each pool can run as its own user and group, with its own environment and php.ini values, and its process manager keeps a fixed number of workers, starts them on demand, or varies the count between limits. The language moves in release branches, and the last four arrived a year apart. Each branch gets two years of active support, with bug and security fixes, then two more years of critical security fixes only, and then reaches end of life. PHP 8.2, released on 8 December 2022, receives security fixes until 31 December 2026; PHP 8.5, released on 20 November 2025, until 31 December 2029.

PHP converts between types when an operation calls for it, and the loose comparison operator `==` is where that shows most. If both sides are numeric strings, or one is a number and the other a numeric string, the comparison is numeric. Before PHP 8.0.0 any string compared with a number was converted to a number first, so `0 == "a"` was true. The strict operator `===` compares the type as well as the value and converts nothing, and the manual's own advice is to prefer it in most cases.

```
<?php
var_dump(0 == "a");      // PHP 7: true, PHP 8: false
var_dump("1" == "01");   // true in both: two numeric strings
var_dump("10" == "1e1"); // true in both: 10 equals 1e1
var_dump(100 == "1e2");  // true in both
var_dump(0 === "a");     // false in both: === never converts
```

Use the concept in a decision

Put "What is PHP?" to work in a real decision. We can help relate the concept to the system you are building or buying. The part users never see: where the data lives, what the rules are, and how other systems are allowed to talk to it.

Explore our services Get in touch →

Test the assumptions behind the term

Use the explanation and linked work to frame questions for your project. We can examine the assumptions with you before a tool is chosen. Designed and ran 6 ETL/ELT pipelines — 6 ETL pipelines across three database engines

Designed and ran 6 ETL/ELT pipelines Get in touch →

Why it matters

PHP matters mostly because of what is already written in it. WordPress and Joomla, the content management systems that a great many organisations use to run sites they edit themselves, are PHP applications: WordPress recommends PHP 8.3 or greater, and Joomla 6 requires at least PHP 8.3.0. WordPress still runs on PHP 7.4, though its requirements page warns that such versions have reached end of life. A business can depend on PHP for years without anyone in it having written a line of the language, which is why the version underneath gets forgotten. A site keeps serving pages after its branch reaches end of life; it simply stops receiving fixes, and nothing on the screen says so.

Upgrades also change behaviour, quietly. PHP 8.0 changed how strings compare with numbers, so a condition that was true on PHP 7 can be false on PHP 8 without any error. PHP 8.4 raised the default bcrypt cost of `password_hash()` from

10 to 12, which makes every new password hash more expensive to compute on the same hardware. Both are documented, and each reaches production only through an upgrade that someone has to plan, test and own.

Our recommendation is to treat the PHP version as a dependency with an expiry date written next to it. An existing PHP application that works is worth keeping: put it on a supported branch, schedule the next upgrade well before security support ends, and give it its own FPM pool and its own system user. For a new build, choose PHP when the job is a site the client will edit through WordPress or Joomla, or when the team that will run it after handover knows PHP well, and choose by that team's skills otherwise. The language is rarely the risk; an unowned version is.

What it looks like in practice

The web tier of our tech stack puts PHP in a specific place. Pages that can be files are built statically, avoiding request-time application execution for those pages, and this site is built by Hugo; Django and PHP are used where there is an application to serve. The same page states how the choice is made: for the problem, for what the client's team can operate after we leave, and for how the thing fails rather than how it demonstrates. That rule covers PHP directly: when the right answer is the stack a client already has, the page commits to saying so.

From 2015 to 2022, the founder worked as GIS software development team lead, taking more than thirty GIS projects through development, deployment and support once they were live. PHP was one of four languages in their application layer, next to JavaScript, Python and SQL; below it all the spatial data sat in PostgreSQL with PostGIS, and in front were Mapbox GL and Leaflet maps, some of them calling the ArcGIS or HERE APIs. Whatever the language, the responsibility ran past shipping and problems were not handed off: you design differently when you know you are the one getting the call if it breaks.

Where it goes wrong

- **A loose comparison guards a secret.** A token or a hash checked with `==` is compared by PHP's numeric rules whenever both sides look like numbers, so two different strings such as `"10"` and `"1e1"` count as equal. Use `password_verify()` for passwords and `hash_equals()` for other secrets: its running time leaks nothing about the contents of the secret, and it expects the user's string as its second argument. Use `===` everywhere else.

- **SQL is assembled from request data.** PDO prepared statements quote their parameters, and the manual says an application that uses them exclusively is protected from SQL injection, with one stated exception: any part of the query still built from unescaped input. A placeholder stands for a whole value, so anything that is not a value, such as the column a list is sorted by, still ends up in the string. Check those parts against a fixed list in code.
- **Password hashes are stored too narrowly.** `PASSWORD_DEFAULT` is bcrypt today, and the manual warns that it is designed to change as stronger algorithms are added. It recommends a column that can expand beyond 60 bytes, with 255 as a good choice. Bcrypt also truncates the password at 72 bytes. Store the whole hash, and call `password_needs_rehash()` after a successful login so older hashes are upgraded as people sign in.
- **The branch passes its end-of-life date.** The application keeps working, so nobody schedules the upgrade, and the eventual jump spans several branches at once. Record the security-support date of the branch in use, upgrade one branch at a time, and run the test suite against the next branch before the current one expires.
- **The FPM pool is shared and loose.** One pool serving several applications as one user lets a flaw in one reach the files of all. `security.limit_extensions` defaults to `.php .phar`, and the manual advises limiting it to `.php` so a misconfigured web server cannot hand other files to PHP. Give each application its own pool and user, and set `pm.max_children` to what the server's memory can hold.

Further reading

- PHP Manual: What is PHP? — the project's own short definition, with the first example of PHP embedded in HTML.
- PHP: Supported Versions — the support policy and the active and security-only dates of every current branch.
- PHP Manual: Comparison Operators — loose and strict comparison, the PHP 7 and PHP 8 outputs side by side, and the comparison table by type.
- PHP Manual: password_hash — algorithms, cost, column width, the 72-byte limit and the change of default cost in 8.4.
- PHP Manual: Prepared statements and stored procedures — how PDO binds parameters and where SQL injection is still possible.
- PHP Manual: FPM configuration — pool directives, the process manager modes, `clear_env` and `security.limit_extensions`.



Glossary

Choose for your actual constraints

Describe the workflow, the existing interfaces and the users who need the change. We can define a testable delivery and a handover that lets your team maintain it.

See what it costs Get in touch →

Make the terminology practical

Ask how “What is PHP?” applies to your organisation. A concrete example from your current system is enough to start the discussion.

Explore our services

welcome@engineer.company +45 50 10 61 84 Copenhagen, Denmark

Take this page with you

Contact card

Carefully Crafted With Love by D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – present · Engineer ApS · All rights reserved.

Printed from https://engineer.company/glossary/php/