PHP

From Wikipedia, the free encyclopedia

PHP is a server-side scripting language designed primarily for web development but also used as a general-purpose programming language. Originally created by Rasmus Lerdorf in 1994,[5] the PHP reference implementation is now produced by The PHP Development Team.[6] PHP originally stood for *Personal Home Page*,[5] but it now stands for the recursive acronym *PHP: Hypertext Preprocessor*.[7]

PHP code may be embedded into HTML code, or it can be used in combination with various web template systems, web content management systems and web frameworks. PHP code is usually processed by a PHP interpreter implemented as a module in the web server or as a Common Gateway Interface (CGI) executable. The web server combines the results of the interpreted and executed PHP code, which may be any type of data, including images, with the generated web page. PHP code may also be executed with a command-line interface (CLI) and can be used to implement standalone graphical applications.[8]

The standard PHP interpreter, powered by the Zend Engine, is free software released under the PHP License. PHP has been widely ported and can be deployed on most web servers on almost every operating system and platform, free of charge.[9]

The PHP language evolved without a written formal specification or standard until 2014, leaving the canonical PHP interpreter as a *de facto* standard. Since 2014 work has gone on to create a formal PHP specification.[10]

During the 2010s there have been increased efforts towards standardisation and code sharing in PHP applications by projects such as PHP-FIG (http://www.php-fig.org/) in the form of PSR-initiatives (http://www.php-fig.org/psr/) as well as Composer dependency manager and the Packagist repository (https://packagist.org/).

PHP

Paradigm	Imperative, functional, object-oriented, procedural, reflective
Designed by	Rasmus Lerdorf
Developer	The PHP Development Team, Zend Technologies
First appeared	June 8, 1995[1]
Stable release	7.0.13[2] / November 10, 2016
Preview release	7.1.0 Release Candidate 6[3] / November 10, 2016
Typing discipline	Dynamic, weak, gradual (as of PHP 7.0.0)[4]
Implementation language	C (primarily; some components C++)
OS	Unix-like, Windows
License	PHP License (most of Zend Engine under Zend Engine License & The TSRM License)
Filename extensions	.php, .phtml, .php3, .php4, .php5, .php7, .phps
Website	php.net (https://php.net)

Major implementations

Zend Engine, HHVM, Phalanger, Quercus, Project Zero, Parrot

Influenced by

C, C++, Java, Perl, Tcl[1]

Influenced

Falcon, Hack

PHP Programming at Wikibooks

Contents

- 1 History
  - 1.1 Early history
  - 1.2 PHP 3 and 4
  - 1.3 PHP 5

    - 1.4 PHP 6 and Unicode
    - 1.5 PHP 7
    - 1.6 Release history
- 2 Mascot
- 3 Syntax
    - 3.1 Data types
    - 3.2 Functions
    - 3.3 Object-oriented programming
- 4 Implementations
- 5 Licensing
- 6 Development and community
- 7 Installation and configuration
- 8 Use
- 9 Security
- 10 See also
- 11 References
- 12 Further reading
- 13 External links

History

Early history

PHP development began in 1995 when Rasmus Lerdorf wrote several Common Gateway Interface (CGI) programs in C,[11][12][13] which he used to maintain his personal homepage. He extended them to work with web forms and to communicate with databases, and called this implementation "Personal Home Page/Forms Interpreter" or PHP/FI.

Rasmus Lerdorf, who wrote the original Common Gateway Interface (CGI) component, together with Andi Gutmans and Zeev Suraski, who rewrote the parser that formed PHP 3.

PHP/FI could help to build simple, dynamic web applications. To accelerate bug reporting and to improve the code, Lerdorf initially announced the release of PHP/FI as "Personal Home Page Tools (PHP Tools) version 1.0" on the Usenet discussion group *comp.infosystems.www.authoring.cgi* on June 8, 1995.[14][15] This release already had the basic functionality that PHP has as of 2013. This included Perl-like variables, form handling, and the ability to embed HTML. The syntax resembled that of Perl but was simpler, more limited and less consistent.[6]

Lerdorf did not intend the early PHP to become a new programming language, but it grew organically, with Lerdorf noting in retrospect: "I don’t know how to stop it, there was never any intent to write a programming language […] I have absolutely no idea how to write a programming language, I just kept adding the next logical step on the way."[16] A development team began to form and, after months of work and beta testing, officially released PHP/FI 2 in November 1997.

The fact that PHP lacked an original overall design but instead developed organically has led to inconsistent naming of functions and inconsistent ordering of their parameters.[17] In some cases, the function names were chosen to match the lower-level libraries which PHP was "wrapping",[18] while in some very early versions of PHP the length of the function names was used internally as a hash function, so names were chosen to improve the distribution of hash values.[19]

PHP 3 and 4

Zeev Suraski and Andi Gutmans rewrote the parser in 1997 and formed the base of PHP 3, changing the language's name to the recursive acronym *PHP: Hypertext Preprocessor*.[6][20] Afterwards, public testing of PHP 3 began, and the official launch came in June 1998. Suraski and Gutmans then started a new rewrite of PHP's core, producing the Zend Engine in 1999.[21] They also founded Zend Technologies in Ramat Gan, Israel.[6]

functions.php

```
<?php
if (function_exists('register_sidebar')) {
    register_sidebar(array(
        'name'          => 'Right Sidebar',
        'id'            => 'sidebar-right',
        'description'   => 'This will display on the right side of your website pages.',
        'before_widget' => '<li id="%1$s" class="widget %2$s">',
        'after_widget' => '</li>',
        'before_title' => '<h2 class="widgettitle">',
        'after_title' => '</h2>',
    ));
    /* Add One Call to register_sidebar() for each sidebar you need.
    register_sidebar(array(
        'name'          => 'Left Sidebar',
        'id'            => 'sidebar-left',
        'description'   => 'This will display on the left side of your website pages under the left navigation.',
        'before_widget' => '<li id="%1$s" class="widget %2$s">',
        'after_widget' => '</li>',
        'before_title' => '<h2 class="widgettitle">',
        'after_title' => '</h2>',
    ));
    */
}
/*

add_action('init', 'html_page_permalink', -1);

function html_page_permalink() {
    global $wp_rewrite;
    if ( !strpos($wp_rewrite->get_page_permastruct(), '.shtml')){
        $wp_rewrite->page_structure = $wp_rewrite->page_structure . '.shtml';
    }
}
add_filter('user_trailingslashit', 'no_page_slash',66,2);

function no_page_slash($string, $type){
    global $wp_rewrite;
    if ($wp_rewrite->using_permalinks() && $wp_rewrite->use_trailing_slashes==true && $type == 'page'){
        return untrailingslashit($string);
    }else{
        return $string;
    }
}
*/
```

This is an example of custom php code on a computer screen.

On May 22, 2000, PHP 4, powered by the Zend Engine 1.0, was released.[6] As of August 2008 this branch reached version 4.4.9. PHP 4 is no longer under development nor will any security updates be released.[22][23]

PHP 5

On July 13, 2004, PHP 5 was released, powered by the new Zend Engine II.[6] PHP 5 included new features such as improved support for object-oriented programming, the PHP Data Objects (PDO) extension (which defines a lightweight and consistent interface for accessing databases), and numerous performance enhancements.[24] In 2008 PHP 5 became the only stable version under development. Late static binding had been missing from PHP and was added in version 5.3.[25][26]

Many high-profile open-source projects ceased to support PHP 4 in new code as of February 5, 2008, because of the GoPHP5 initiative,[27] provided by a consortium of PHP developers promoting the transition from PHP 4 to PHP 5.[28][29]

Over time, PHP interpreters became available on most existing 32-bit and 64-bit operating systems, either by building them from the PHP source code, or by using pre-built binaries.[30] For the PHP versions 5.3 and 5.4, the only available Microsoft Windows binary distributions were 32-bit x86 builds,[31][32] requiring Windows 32-bit compatibility mode while using Internet Information Services (IIS) on a 64-bit Windows platform. PHP version 5.5 made the 64-bit x86-64 builds available for Microsoft Windows.[33]

PHP 6 and Unicode

PHP received mixed reviews due to lacking native Unicode support at the core language level.[34][35] In 2005, a project headed by Andrei Zmievski was initiated to bring native Unicode support throughout PHP, by embedding the International Components for Unicode (ICU) library, and representing text strings as UTF-16 internally.[36] Since this would cause major changes both to the internals of the language and to user code, it was planned to release this as version 6.0 of the language, along with other major features then in development.[37]

However, a shortage of developers who understood the necessary changes, and performance problems arising from conversion to and from UTF-16, which is rarely used in a web context, led to delays in the project.[38] As a result, a PHP 5.3 release was created in 2009, with many non-Unicode features back-ported from PHP 6, notably namespaces. In March 2010, the project in its current form was officially abandoned, and a PHP 5.4 release was prepared containing most remaining non-Unicode features from PHP 6, such as traits and closure re-binding.[39] Initial hopes were that a new plan would be formed for Unicode integration, but as of 2014 none have been adopted.

PHP 7

During 2014 and 2015, a new major PHP version was developed, which was numbered PHP 7. The numbering of this version involved some debate.[40] While the PHP 6 Unicode experiment had never been released, several articles and book titles referenced the PHP 6 name, which might have caused confusion if a new release were to reuse the name.[41] After a vote, the name PHP 7 was chosen.[42]

The foundation of PHP 7 is a PHP branch that was originally dubbed *PHP next generation* (*phpng*). It was authored by Dmitry Stogov, Xinchen Hui and Nikita Popov,[43] and aimed to optimize PHP performance by refactoring the Zend Engine while retaining near-complete language compatibility.[44] As of 14 July 2014, WordPress-based benchmarks, which served as the main benchmark suite for the phpng project, showed an almost 100% increase in performance. Changes from phpng are also expected to make it easier to improve performance in the future, as more compact data structures and other changes are seen as better suited for a successful migration to a just-in-time (JIT) compiler.[45] Because of the significant changes, the reworked Zend Engine is called *Zend Engine 3*, succeeding Zend Engine 2 used in PHP 5.[46]

Because of major internal changes in phpng, it must receive a new major version number of PHP, rather than a minor PHP 5 release, according to PHP's release process.[47] Major versions of PHP are allowed to break backward-compatibility of code and therefore PHP 7 presented an opportunity for other improvements beyond phpng that require backward-compatibility breaks. In particular, it involved the following changes:

- Many fatal- or recoverable-level legacy PHP error mechanisms were replaced with modern object-oriented exceptions[48]
- The syntax for variable dereferencing was reworked to be internally more consistent and complete, allowing the use of the operators ->, *[]*, *()*, *{}*, and *::* with arbitrary meaningful left-hand-side expressions[49]
- Support for legacy PHP 4-style constructor methods was deprecated[50]
- The behavior of the *foreach* statement was changed to be more predictable[51]
- Constructors for the few classes built-in to PHP which returned null upon failure were changed to throw an exception instead, for consistency[52]
- Several unmaintained or deprecated server application programming interfaces (SAPIs) and extensions were removed from the PHP core, most notably the legacy *mysql* extension[53]
- The behavior of the *list()* operator was changed to remove support for strings[54]
- Support for legacy ASP-style PHP code delimiters (<% and %>, <*script language=php*> and </*script*>) was removed[55]
- An oversight allowing a switch statement to have multiple *default* clauses was fixed[56]
- Support for hexadecimal number support in some implicit conversions from strings to number types was removed[57]
- The left-shift and right-shift operators were changed to behave more consistently across platforms[58]

- Conversions between integers and floating point numbers were tightened and implemented more consistently across platforms[58][59]

PHP 7 also included new language features. Most notably, it introduces return type declarations for functions,[60] which complement the existing parameter type declarations, and support for the scalar types (integer, float, string, and boolean) in parameter and return type declarations.[61]

Release history

Key

Color	Meaning	Development
Red	Old release	No development
Yellow	Stable release	Security fixes
Green	Stable release	Bug and security fixes
Blue	Future release	New features

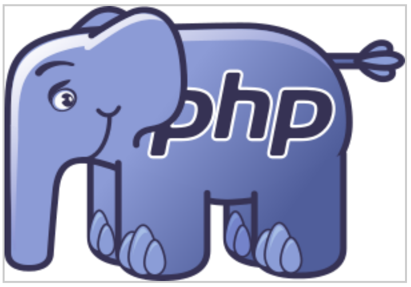
Version	Release date	Supported until[62]	Notes
1.0	8 June 1995		Officially called "Personal Home Page Tools (PHP Tools)". This is the first use of the name "PHP".[6]
2.0	1 November 1997		Officially called "PHP/FI 2.0". This is the first release that could actually be characterised as PHP, being a standalone language with many features that have endured to the present day.
3.0	6 June 1998	20 October 2000[62]	Development moves from one person to multiple developers. Zeev Suraski and Andi Gutmans rewrite the base for this version.[6]
4.0	22 May 2000	23 June 2001[62]	Added more advanced two-stage parse/execute tag-parsing system called the Zend engine.[63]
4.1	10 December 2001	12 March 2002[62]	Introduced "superglobals" (`$_GET`, `$_POST`, `$_SESSION`, etc.)[63]
4.2	22 April 2002	6 September 2002[62]	Disabled `register_globals` by default. Data received over the network is not inserted directly into the global namespace anymore, closing possible security holes in applications.[63]
4.3	27 December 2002	31 March 2005[62]	Introduced the command-line interface (CLI), to supplement the CGI.[63][64]
4.4	11 July 2005	7 August 2008[62]	Fixed a memory corruption bug, which required breaking binary compatibility with extensions compiled against PHP version 4.3.x.[65]
5.0	13 July 2004	5 September 2005[62]	Zend Engine II with a new object model.[66]
5.1	24 November 2005	24 August 2006[62]	Performance improvements with introduction of compiler variables in re-engineered PHP Engine.[66] Added PHP Data Objects (PDO) as a consistent interface for accessing databases.[67]
5.2	2 November 2006	6 January 2011[62]	Enabled the filter extension by default. Native JSON support.[66]
5.3	30 June 2009	14 August 2014[62]	Namespace support; late static bindings, jump label (limited goto), closures, PHP archives (phar), garbage collection for circular references, improved Windows support, sqlite3, mysqlnd as a replacement for libmysql as underlying library for the extensions that work with MySQL, fileinfo as a replacement for mime_magic for better MIME support, the Internationalization extension, and deprecation of ereg extension.
5.4	1 March 2012	3 September 2015[62]	Trait support, short array syntax support. Removed items: `register_globals`, `safe_mode`, `allow_call_time_pass_reference`, `session_register()`, `session_unregister()` and `session_is_registered()`. Built-in web server.[68] Several improvements to existing features, performance and reduced memory requirements.

5.5	20 June 2013	21 July 2016[62]	Support for generators, `finally` blocks for exceptions handling, OpCache (based on Zend Optimizer+) bundled in official distribution.[69]
5.6	28 August 2014	31 December 2018[70]	Constant scalar expressions, variadic functions, argument unpacking, new exponentiation operator, extensions of the `use` statement for functions and constants, new `phpdbg` debugger as a SAPI module, and other smaller improvements.[71]
6.x	Not released	N/A	Abandoned version of PHP that planned to include native Unicode support.[72][73]
7.0	3 December 2015[2]	3 December 2018[70]	Zend Engine 3 (performance improvements[45] and 64-bit integer support on Windows[74]), uniform variable syntax,[49] AST-based compilation process,[75] added `Closure::call()`,[76] bitwise shift consistency across platforms,[77] `??` (null coalesce) operator,[78] Unicode codepoint escape syntax,[79] return type declarations,[60] scalar type (integer, float, string and boolean) declarations,[61] <=> "spaceship" three-way comparison operator,[80] generator delegation,[81] anonymous classes,[82] simpler and more consistently available CSPRNG API,[83] replacement of many remaining internal PHP "errors" with the more modern exceptions,[48] and shorthand syntax for importing multiple items from a namespace.[84]
7.1	November 2016[85]	3 years after release[47]	void return type,[86] class constant visibility modifiers,[87] nullable types[88]

Beginning on June 28, 2011, the PHP Group implemented a timeline for the release of new versions of PHP.[47] Under this system, at least one release should occur every month. Once per year, a minor release should occur which may include new features. Every minor release should at least be supported for two years with security and bug fixes, followed by at least one year of only security fixes, for a total of a three-year release process for every minor release. No new features, unless small and self-contained, are to be introduced into a minor release during the three-year release process..

Mascot

The mascot of the PHP project is the *elePHPant*, a blue elephant with the PHP logo on its side, designed by Vincent Pontier[89] in 1998.[90] The elePHPant is sometimes differently colored when in plush toy form.


The elePHPant, PHP mascot.

Syntax

The following "Hello, World!" program is written in PHP code embedded in an HTML document:

```
<!DOCTYPE html>
<html>
    <head>
        <title>PHP Test</title>
    </head>
    <body>
        <?php echo '<p>Hello World</p>'; ?>
    </body>
</html>
```

However, as no requirement exists for PHP code to be embedded in HTML, the simplest version of *Hello, World!* may be written like this, with the closing tag omitted as preferred in files containing pure PHP code[91]

```
<?='Hello world';
```

The PHP interpreter only executes PHP code within its delimiters. Anything outside its delimiters is not processed by PHP, although non-PHP text is still subject to control structures described in PHP code. The most common delimiters are *<?php* to open and *?>* to close PHP sections. The shortened form *<?* also exists. This short delimiter makes script files less portable, since support for them can be disabled in the local PHP configuration and it is therefore discouraged.[92][93] However, there is no recommendation against the use of the echo short tag *<?=*.[94] Prior to PHP 5.4.0, this short syntax for *echo()* only works with the *short_open_tag* configuration setting enabled, while for PHP 5.4.0 and later it is always available.[92][95][96] The purpose of all these delimiters is to separate PHP code from non-PHP content, such as JavaScript code or HTML markup.[97]

The first form of delimiters, *<?php* and *?>*, in XHTML and other XML documents, creates correctly formed XML processing instructions.[98] This means that the resulting mixture of PHP code and other markup in the server-side file is itself well-formed XML.

Variables are prefixed with a dollar symbol, and a type does not need to be specified in advance. PHP 5 introduced *type hinting* that allows functions to force their parameters to be objects of a specific class, arrays, interfaces or callback functions. However, before PHP 7.0, type hints could not be used with scalar types such as integer or string.[61]

Unlike function and class names, variable names are case sensitive. Both double-quoted ("") and heredoc strings provide the ability to interpolate a variable's value into the string.[99] PHP treats newlines as whitespace in the manner of a free-form language, and statements are terminated by a semicolon.[100] PHP has three types of comment syntax: `/* */` marks block and inline comments; `//` as well as `#` are used for one-line comments.[101] The `echo` statement is one of several facilities PHP provides to output text, *e.g.*, to a web browser.

In terms of keywords and language syntax, PHP is similar to the C style syntax. *if* conditions, *for* and *while* loops, and function returns are similar in syntax to languages such as C, C++, C#, Java and Perl.

The following is an example of PHP for loop:

```php
<?php
for ($x = 0; $x <= 100; $x++) {
    echo "The number is: $x <br>";
}
?>
```

Data types

PHP stores integers in a platform-dependent range, either a 64-bit or 32-bit signed integer equivalent to the C-language long type. Unsigned integers are converted to signed values in certain situations; this behavior is different from that of other programming languages.[102] Integer variables can be assigned using decimal (positive and negative), octal, hexadecimal, and binary notations.

Floating point numbers are also stored in a platform-specific range. They can be specified using floating point notation, or two forms of scientific notation.[103] PHP has a native Boolean type that is similar to the native Boolean types in Java and C++. Using the Boolean type conversion rules, non-zero values are interpreted as true and zero as false, as in Perl and C++.[103]

The null data type represents a variable that has no value; `NULL` is the only allowed value for this data type.[103]

Variables of the "resource" type represent references to resources from external sources. These are typically created by functions from a particular extension, and can only be processed by functions from the same extension; examples include file, image, and database resources.[103]

Arrays can contain elements of any type that PHP can handle, including resources, objects, and other arrays. Order is preserved in lists of values and in hashes with both keys and values, and the two can be intermingled.[103] PHP also supports strings, which can be used with single quotes, double quotes, nowdoc or heredoc syntax.[104]

The Standard PHP Library (SPL) attempts to solve standard problems and implements efficient data access interfaces and classes.[105]

Functions

PHP defines a large array of functions in the core language and many are also available in various extensions; these functions are well documented in the online PHP documentation.[106] However, the built-in library has a wide variety of naming conventions and associated inconsistencies, as described under history above.

Custom functions may be defined by the developer, e.g.:

```
function myAge($birthYear) {                                  // defines a function, this one is named "myAge"
    $yearsOld = date('Y') - $birthYear;                       // calculates the age
    return $yearsOld . ' year' . ($yearsOld != 1 ? 's' : ''); // returns the age in a descriptive form
}

echo 'I am currently ' . myAge(1981) . ' old.';               // outputs the text concatenated
                                                              // with the return value of myAge()
// As the result of this syntax, myAge() is called.
```

In 2016, the output of the above sample program is 'I am currently 35 years old.'

In lieu of function pointers, functions in PHP can be referenced by a string containing their name. In this manner, normal PHP functions can be used, for example, as callbacks or within function tables.[107] User-defined functions may be created at any time without being prototyped.[106][107] Functions may be defined inside code blocks, permitting a run-time decision as to whether or not a function should be defined. There is a `function_exists` function that determines whether a function with a given name has already been defined. Function calls must use parentheses, with the exception of zero-argument class constructor functions called with the PHP operator *new*, in which case parentheses are optional.

Until PHP 5.3, support for anonymous functions and closures did not exist in PHP. While `create_function()` exists since PHP 4.0.1, it is merely a thin wrapper around `eval()` that allows normal PHP functions to be created during program execution.[108] PHP 5.3 added syntax to define an anonymous function or "closure"[109] which can capture variables from the surrounding scope:

```
function getAdder($x) {
    return function($y) use ($x) {
        return $x + $y;
    };
}

$adder = getAdder(8);
echo $adder(2); // prints "10"
```

In the example above, `getAdder()` function creates a closure using passed argument `$x` (the keyword `use` imports a variable from the lexical context), which takes an additional argument `$y`, and returns the created closure to the caller. Such a function is a first-class object, meaning that it can be stored in a variable, passed as a parameter to other functions, etc.[110]

Unusually for a dynamically typed language, PHP supports type declarations on function parameters, which are enforced at runtime. This has been supported for classes and interfaces since PHP 5.0, for arrays since PHP 5.1, for "callables" since PHP 5.4, and scalar (integer, float, string and boolean) types since PHP 7.0.[61] PHP 7.0 also has type declarations for function return types, expressed by placing the type name after the list of parameters, preceded by a colon.[60] For example, the `getAdder` function from the earlier example could be annotated with types like so in PHP 7:

```
function getAdder(int $x): \Closure {
    return function(int $y) use ($x) : int {
        return $x + $y;
    };
}

$adder = getAdder(8);
echo $adder(2);        // prints "10"
echo $adder(null);     // throws an exception because an incorrect type was passed
$adder = getAdder([]); // would also throw an exception
```

By default, scalar type declarations follow weak typing principles. So, for example, if a parameter's type is `int`, PHP would allow not only integers, but also convertible numeric strings, floats or booleans to be passed to that function, and would convert them.[61] However, PHP 7 has a "strict typing" mode which, when used, disallows such conversions for function calls and returns within a file.[61]

Object-oriented programming

Basic object-oriented programming functionality was added in PHP 3 and improved in PHP 4.[6] This allowed for PHP to gain further abstraction, making creative tasks easier for programmers using the language. Object handling was completely rewritten for PHP 5, expanding the feature set and enhancing performance.[111] In previous versions of PHP, objects were handled like value types.[111] The drawback of this method was that code had to make heavy use of PHP's "reference" variables if it wanted to modify an object it was passed rather than creating a copy of it. In the new approach, objects are referenced by handle, and not by value.

PHP 5 introduced private and protected member variables and methods, along with abstract classes, final classes, abstract methods, and final methods. It also introduced a standard way of declaring constructors and destructors, similar to that of other object-oriented languages such as C++, and a standard exception handling model. Furthermore, PHP 5 added interfaces and allowed for multiple interfaces to be implemented. There are special interfaces that allow objects to interact with the runtime system. Objects implementing ArrayAccess can be used with array syntax and objects implementing Iterator or IteratorAggregate can be used with the `foreach` language construct. There is no virtual table feature in the engine, so static variables are bound with a name instead of a reference at compile time.[112]

If the developer creates a copy of an object using the reserved word `clone`, the Zend engine will check whether a `__clone()` method has been defined. If not, it will call a default `__clone()` which will copy the object's properties. If a `__clone()` method is defined, then it will be responsible for setting the necessary properties in the created object. For convenience, the engine will supply a function that imports the properties of the source object, so the programmer can start with a by-value replica of the source object and only override properties that need to be changed.[113]

The following is a basic example of object-oriented programming in PHP:

```php
class Person
{
    public $firstName;
    public $lastName;

    public function __construct($firstName, $lastName = '') { // optional second argument
        $this->firstName = $firstName;
        $this->lastName  = $lastName;
    }

    public function greet() {
        return 'Hello, my name is ' . $this->firstName .
               (($this->lastName != '') ? (' ' . $this->lastName) : '') . '.';
    }

    public static function staticGreet($firstName, $lastName) {
        return 'Hello, my name is ' . $firstName . ' ' . $lastName . '.';
    }
}

$he    = new Person('John', 'Smith');
$she   = new Person('Sally', 'Davis');
$other = new Person('iAmine');

echo $he->greet(); // prints "Hello, my name is John Smith."
echo '<br />';

echo $she->greet(); // prints "Hello, my name is Sally Davis."
echo '<br />';

echo $other->greet(); // prints "Hello, my name is iAmine."
echo '<br />';

echo Person::staticGreet('Jane', 'Doe'); // prints "Hello, my name is Jane Doe."
```

The visibility of PHP properties and methods is defined using the keywords `public`, `private`, and `protected`. The default is public, if only var is used; `var` is a synonym for `public`. Items declared `public` can be accessed everywhere. `protected` limits access to inherited classes (and to the class that defines the item). `private` limits visibility only to the class that defines the item.[114] Objects of the same type have access to each other's private and protected members even though they are not the same instance. PHP's member visibility features have sometimes been described as "highly useful."[115] However, they have also sometimes been described as "at best irrelevant and at worst positively harmful."[116]

Implementations

The original, only complete and most widely used PHP implementation is powered by the Zend Engine and known simply as PHP. To disambiguate it from other implementations, it is sometimes unofficially referred to as "Zend PHP". The Zend Engine compiles PHP source code on-the-fly into an internal format that it can execute, thus it works as an interpreter.[117][118] It is also the "reference implementation" of PHP, as PHP has no formal specification, and so the semantics of Zend PHP define the semantics of PHP itself. Due to the complex and nuanced semantics of PHP, defined by how Zend works, it is difficult for competing implementations to offer complete compatibility.

PHP's single-request-per-script-execution model, and the fact the Zend Engine is an interpreter, leads to inefficiency; as a result, various products have been developed to help improve PHP performance. In order to speed up execution time and not have to compile the PHP source code every time the web page is accessed, PHP scripts can also be deployed in the PHP engine's internal format by using an opcode cache, which works by caching the compiled form of a PHP script (opcodes) in shared memory to avoid the overhead of parsing and compiling the code every time the script runs. An opcode cache, Zend Opcache, is built into PHP since version 5.5.[119] Another example of a widely used opcode cache is the Alternative PHP Cache (APC), which is available as a PECL extension.[120]

While Zend PHP is still the most popular implementation, several other implementations have been developed. Some of these are compilers or support JIT compilation, and hence offer performance benefits over Zend PHP at the expense of lacking full PHP compatibility. Alternative implementations include the following:

- HipHop Virtual Machine (HHVM) – developed at Facebook and available as open source, it converts PHP code into a high-level bytecode (commonly known as an intermediate language), which is then translated into x86-64 machine code dynamically at runtime by a just-in-time (JIT) compiler, resulting in up to 6× performance improvements.[121]
- Parrot – a virtual machine designed to run dynamic languages efficiently; Pipp transforms the PHP source code into the Parrot intermediate representation, which is then translated into the Parrot's bytecode and executed by the virtual machine.
- Phalanger – compiles PHP into Common Intermediate Language (CIL) bytecode
- HipHop – developed at Facebook and available as open source, it transforms the PHP scripts into C++ code and then compiles the resulting code, reducing the server load up to 50%. In early 2013, Facebook

deprecated it in favor of HHVM due to multiple reasons, including deployment difficulties and lack of support for the whole PHP language, including the `create_function()` and `eval()` constructs.[122]

Licensing

PHP is free software released under the PHP License, which stipulates that:[123]

> Products derived from this software may not be called "PHP", nor may "PHP" appear in their name, without prior written permission from group@php.net. You may indicate that your software works in conjunction with PHP by saying "Foo for PHP" instead of calling it "PHP Foo" or "phpfoo".

This restriction on use of "PHP" makes the PHP License incompatible with the General Public License (GPL), while the Zend License is incompatible due to an advertising clause similar to that of the original BSD license.[124]

Development and community

PHP includes various free and open-source libraries in its source distribution, or uses them in resulting PHP binary builds. PHP is fundamentally an Internet-aware system with built-in modules for accessing File Transfer Protocol (FTP) servers and many database servers, including PostgreSQL, MySQL, Microsoft SQL Server and SQLite (which is an embedded database), LDAP servers, and others. Numerous functions familiar to C programmers, such as those in the stdio family, are available in standard PHP builds.[125]

PHP allows developers to write extensions in C to add functionality to the PHP language. PHP extensions can be compiled statically into PHP or loaded dynamically at runtime. Numerous extensions have been written to add support for the Windows API, process management on Unix-like operating systems, multibyte strings (Unicode), cURL, and several popular compression formats. Other PHP features made available through extensions include integration with IRC, dynamic generation of images and Adobe Flash content, *PHP Data Objects* (PDO) as an abstraction layer used for accessing databases,[126][127][128][129][130][131][132] and even speech synthesis. Some of the language's core functions, such as those dealing with strings and arrays, are also implemented as extensions.[133] The PHP Extension Community Library (PECL) project is a repository for extensions to the PHP language.[134]

Some other projects, such as *Zephir*, provide the ability for PHP extensions to be created in a high-level language and compiled into native PHP extensions. Such an approach, instead of writing PHP extensions directly in C, simplifies the development of extensions and reduces the time required for programming and testing.[135]

The PHP Group consists of ten people (as of 2015): Thies C. Arntzen, Stig Bakken, Shane Caraveo, Andi Gutmans, Rasmus Lerdorf, Sam Ruby, Sascha Schumann, Zeev Suraski, Jim Winstead, Andrei Zmievski.[136]

Zend Technologies provides a PHP Certification based on PHP 5.5[137] exam for programmers to become certified PHP developers.

Installation and configuration

There are two primary ways for adding support for PHP to a web server – as a native web server module, or as a CGI executable. PHP has a direct module interface called Server Application Programming Interface (SAPI), which is supported by many web servers including Apache HTTP Server, Microsoft IIS, Netscape (now defunct) and iPlanet. Some other web servers, such as OmniHTTPd, support the Internet Server Application Programming Interface (ISAPI), which is a Microsoft's web server module interface. If PHP has no module support for a web server, it can always be used as a Common Gateway Interface (CGI) or FastCGI processor; in that case, the web server is configured to use PHP's CGI executable to process all requests to PHP files.[138]

PHP-FPM (FastCGI Process Manager) is an alternative FastCGI implementation for PHP, bundled with the official PHP distribution since version 5.3.3.[139] When compared to the older FastCGI implementation, it contains some additional features, mostly useful for heavily loaded web servers.[140]

When using PHP for command-line scripting, a PHP command-line interface (CLI) executable is needed. PHP supports a CLI SAPI as of PHP 4.3.0.[141] The main focus of this SAPI is developing shell applications using PHP. There are quite a few differences between the CLI SAPI and other SAPIs, although they do share many of the same behaviors.[142]

PHP has a direct module interface called SAPI for different web servers;[143] in case of PHP 5 and Apache 2.0 on Windows, it is provided in form of a DLL file called `php5apache2.dll`,[144] which is a module that, among other functions, provides an interface between PHP and the web server, implemented in a form that the server understands. This form is what is known as a SAPI.

There are different kinds of SAPIs for various web server extensions. For example, in addition to those listed above, other SAPIs for the PHP language include the Common Gateway Interface (CGI) and command-line interface (CLI).[143][145]

PHP can also be used for writing desktop graphical user interface (GUI) applications, by using the PHP-GTK extension. PHP-GTK is not included in the official PHP distribution,[138] and as an extension it can be used only with PHP versions 5.1.0 and newer. The most common way of installing PHP-GTK is compiling it from the source code.[146]

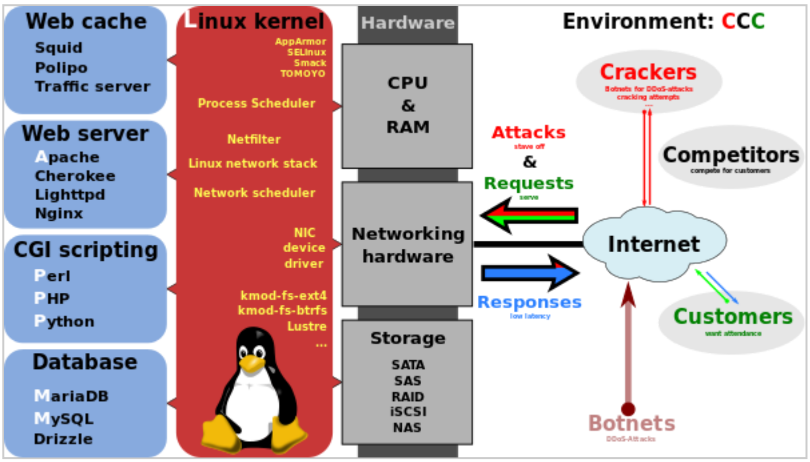
When PHP is installed and used in cloud environments, software development kits (SDKs) are provided for using cloud-specific features. For example:

- Amazon Web Services provides the AWS SDK for PHP[147]
- Windows Azure can be used with the Windows Azure SDK for PHP.[148]

Numerous configuration options are supported, affecting both core PHP features and extensions.[149][150] Configuration file `php.ini` is searched for in different locations, depending on the way PHP is used.[151] The configuration file is split into various sections,[152] while some of the configuration options can be also set within the web server configuration.[153]

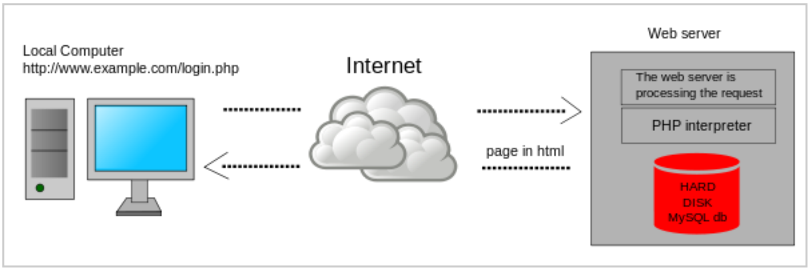
Use

PHP is a general-purpose scripting language that is especially suited to server-side web development, in which case PHP generally runs on a web server. Any PHP code in a requested file is executed by the PHP runtime, usually to create dynamic web page content or dynamic images used on websites or elsewhere.[154] It can also be used for command-line scripting and client-side graphical user interface (GUI) applications. PHP can be deployed on most web servers, many operating systems and platforms, and can be used with many relational database management systems (RDBMS). Most web hosting providers support PHP for use by their clients. It is available free of charge, and the PHP Group provides the complete source code for users to build, customize and extend for their own use.[9]



A broad overview of the LAMP software bundle, displayed here together with Squid.

PHP acts primarily as a filter,[155] taking input from a file or stream containing text and/or PHP instructions and outputting another stream of data. Most commonly the output will be HTML, although it could be JSON, XML or binary data such as image or audio formats. Since PHP 4, the PHP parser compiles input to produce bytecode for processing by the Zend Engine, giving improved performance over its interpreter predecessor.[156]



Dynamic web page: example of server-side scripting (PHP and MySQL).

Originally designed to create dynamic web pages, PHP now focuses mainly on server-side scripting,[157] and it is similar to other server-side scripting languages that provide dynamic content from a web server to a client, such as Microsoft's ASP.NET, Sun Microsystems' JavaServer Pages,[158] and `mod_perl`. PHP has also attracted the development of many software frameworks that provide building blocks and a design structure to promote rapid application development (RAD). Some of these include PRADO, CakePHP, Symfony, CodeIgniter, Laravel, Yii Framework, Phalcon and Zend Framework, offering features similar to other web frameworks.

The LAMP architecture has become popular in the web industry as a way of deploying web applications.[159] PHP is commonly used as the *P* in this bundle alongside Linux, Apache and MySQL, although the *P* may also refer to Python, Perl, or some mix of the three. Similar packages, WAMP and MAMP, are also available for Windows and OS X, with the first letter standing for the respective operating system. Although both PHP and Apache are provided as part of the Mac OS X base install, users of these packages seek a simpler installation mechanism that can be more easily kept up to date.

As of April 2007, over 20 million Internet domains had web services hosted on servers with PHP installed and `mod_php` was recorded as the most popular Apache HTTP Server module.[160] As of October 2010, PHP was used as the server-side programming language on 75% of all websites whose server-side programming language was known[161] (as of February 2014, the percentage had reached 82%[162]), and PHP was the most-used open source software within enterprises.[163] Web content management systems written in PHP include MediaWiki,[164] Joomla,[165] eZ Publish, eZ Platform, SilverStripe,[166] WordPress,[167] Drupal,[168] Moodle,[169] the user-facing portion of Facebook,[170] and Digg.[171]

For specific and more advanced usage scenarios, PHP offers a well defined and documented way for writing custom extensions in C or C++.[172][173][174][175][176][177][178] Besides extending the language itself in form of additional libraries, extensions are providing a way for improving execution speed where it is critical and there is room for improvements by using a true compiled language.[179][180] PHP also offers well defined ways for embedding itself into other software projects. That way PHP can be easily used as an internal scripting language for another project, also providing tight interfacing with the project's specific internal data structures.[181]

PHP received mixed reviews due to lacking support for multithreading at the core language level,[182] though using threads is made possible by the "pthreads" PECL extension.[183][184]

As of January 2013, PHP was used in more than 240 million websites (39% of those sampled) and was installed on 2.1 million web servers.[185]

Security

In 2013, 9% of all vulnerabilities listed by the National Vulnerability Database were linked to PHP;[186] historically, about 30% of all vulnerabilities listed since 1996 in this database are linked to PHP. Technical security flaws of the language itself or of its core libraries are not frequent; (these numbered 22 in 2009, which was about 1% of the total, although PHP applies to about 20% of programs listed.)[187] Recognizing that programmers make mistakes, some languages include taint checking to automatically detect the lack of input validation which induces many issues. Such a feature is being developed for PHP,[188] but its inclusion into a release has been rejected several times in the past.[189][190]

There are advanced protection patches, such as Suhosin and Hardening-Patch, that are especially designed for web hosting environments,[191] primarily due to these environments being seen as places where carelessly written code may run. However, many security features, such as function whitelists, have proven more powerful in application-specific environments. Due to PHP's extensive capabilities and code size, criticism of PHP within security communities can be deflected somewhat by the use of Suhosin. This may cease to be the case if PHP7 moves to using a JIT engine, thereby preventing exploitation mitigation technologies such as W^X from being effective.

There are certain language features and configuration parameters (primarily the default values for such runtime settings) that make PHP applications prone to security issues. Among these, `magic_quotes_gpc` and `register_globals`[192] configuration directives are the best known; the latter made any URL parameters become PHP variables, opening a path for serious security vulnerabilities by allowing an attacker to set the value of any uninitialized global variable and interfere with the execution of a PHP script. Support for "magic quotes" and "register globals" has been deprecated as of PHP 5.3.0, and removed as of PHP 5.4.0.[193]

Another example for the runtime settings vulnerability comes from failing to disable PHP execution (via `engine` configuration directive)[194] for the directory where uploaded images are stored; leaving the default settings can result in execution of malicious PHP code embedded within the uploaded images.[195][196][197] Also, leaving enabled the dynamic loading of PHP extensions (via `enable_dl` configuration directive)[198] in a shared web hosting environment can lead to security issues.[199][200]

Also, implied type conversions that result in incompatible values being treated as identical against the programmer's intent can lead to security issues. For example, the result of the comparison *0e1234 == 0* comparison is *true* because the first compared value is treated as scientific notation having the value (0×10^{1234}), i.e. *zero*. This feature resulted in authentication vulnerabilities in Simple Machines Forum,[201] Typo3[202] and phpBB[203] when MD5 password hashes were compared. Instead, either the function strcmp or the identity operator (===) should be used; *0e1234 === 0* results in *false*.[204]

In a 2013 analysis of over 170,000 website defacements, published by Zone-H, the most frequently (53%) used technique was exploitation of file inclusion vulnerability, mostly related to insecure usage of the PHP functions *include*, *require*, and *allow_url_fopen*.[205][206]

See also

- PEAR (PHP Extension and Application Repository)
- PHP Extension Community Library (PECL)
- PHP accelerator
- List of PHP accelerators
- List of AMP packages
- List of PHP editors
- PHP-GTK
- Template processor
- XAMPP (Free and open source cross-platform web server solution stack package)
- Zend Server
- Hack (programming language)
- Comparison of programming languages
- Comparison of web frameworks

References

1. Lerdorf, Rasmus (2007-04-26). "PHP on Hormones – history of PHP presentation by Rasmus Lerdorf given at the MySQL Conference in Santa Clara, California". The Conversations Network. Retrieved 2009-12-11.
2. "News Archive – 2016: PHP 7.0.13 Released". *php.net*. 2016-11-10. Retrieved 2016-11-10.
3. "News Archive – 2016: PHP 7.1.0 Release Candidate 6 Released". *php.net*. 2016-11-10. Retrieved 2016-11-10.
4. "Type hinting". PHP.net. Retrieved 5 September 2016.
5. "History of PHP". *php.net*.
6. "History of PHP and related projects". The PHP Group. Retrieved 2008-02-25.
7. PHP Manual: Preface (http://php.net/manual/en/preface.php), www.php.net
8. "Introduction: What can PHP do?". *PHP Manual*. Retrieved 2009-03-05.
9. "Embedding PHP in HTML". O'Reilly. 2001-05-03. Retrieved 2008-02-25.
10. Jackson, Joab (2014-07-31). "PHP gets a formal specification, at last". *ITworld*. IDG.
11. Lerdorf, Rasmus (2012-07-20). "I wonder why people keep writing that PHP was ever written in Perl. It never was. #php". Twitter. Retrieved 2014-09-04.
12. Lerdorf, Rasmus (2007-04-26). "PHP on Hormones" (mp3). The Conversations Network. Retrieved 2009-06-22.
13. Lerdorf, Rasmus (2007). "Slide 3". *slides for 'PHP on Hormones' talk*. The PHP Group. Retrieved 2009-06-22.
14. Lerdorf, Rasmus (June 8, 1995). "Announce: Personal Home Page Tools (PHP Tools)". Retrieved 7 June 2011.
15. Lerdorf, Rasmus (1995-06-08). "Announce: Personal Home Page Tools (PHP Tools)". Newsgroup: comp.infosystems.www.authoring.cgi. Retrieved 2006-09-17.
16. "Rasmus Lerdorf, Senior Technical Yahoo: PHP, Behind the Mic". 2003-11-19. Archived from the original on 2013-07-28., cited at Felipe Ribeiro (Aug 7, 2012). "This Is Not Another "PHP Sucks" Article".
17. "Problems with PHP". Retrieved 20 December 2010.
18. "php.internals: Re: Function name consistency". *news.php.net*. 2013-12-28. Retrieved 2014-02-09.
19. Rasmus Lerdorf (Dec 16, 2013). "Re: Flexible function naming". Newsgroup: php.internals. Retrieved December 26, 2013.
20. "PHP - Acronym Meaning Vote". *PHP.net*. Archived from the original on August 15, 2000.
21. "Zend Engine version 2.0: Feature Overview and Design". Zend Technologies Ltd. Retrieved 2006-09-17.

22. "php.net 2007 news archive". The PHP Group. 2007-07-13. Retrieved 2008-02-22.
23. Kerner, Sean Michael (2008-02-01). "PHP 4 is Dead—Long Live PHP 5". InternetNews. Retrieved 2008-03-16.
24. Trachtenberg, Adam (2004-07-15). "Why PHP 5 Rocks!". O'Reilly. Retrieved 2008-02-22.
25. "Late Static Binding in PHP". Digital Sandwich. 2006-02-23. Retrieved 2008-03-25.
26. "Static Keyword". The PHP Group. Retrieved 2008-03-25.
27. "GoPHP5". Archived from the original on 2011-07-17.
28. GoPHP5. "PHP projects join forces to Go PHP 5". *GoPHP5 Press Release*. Archived from the original (PDF) on 2009-12-10. Retrieved 2008-02-23.
29. "GoPHP5". GoPHP5. Archived from the original on 2011-04-27. Retrieved 2008-02-22.
30. "PHP Installation and Configuration". *www.php.net*. Retrieved 2013-10-29.
31. "PHP for Windows: Binaries and sources releases (5.3)". *php.net*. Retrieved 2013-10-29.
32. "PHP for Windows: Binaries and sources releases (5.4)". *php.net*. Retrieved 2013-10-29.
33. "PHP for Windows: Binaries and sources releases (5.5)". *php.net*. Retrieved 2013-10-29.
34. "Types: Strings (PHP Manual)". *PHP.net*. Retrieved 2013-09-22.
35. "Details of the String Type (PHP Manual)". *PHP.net*. Retrieved 2013-09-22.
36. Andrei Zmievski (2005-08-10). "PHP Unicode support design document (mailing list post)". Retrieved 2014-02-09.
37. "PHP 5.5 or 6.0". Retrieved 2014-02-09.
38. Andrei Zmievski. "The Good, the Bad, and the Ugly: What Happened to Unicode and PHP 6". Retrieved 2014-02-09.
39. Rasmus Lerdorf (2010-03-11). "PHP 6 (mailing list post)". Retrieved 2014-02-07.
40. https://philsturgeon.uk/php/2014/07/23/neverending-muppet-debate-of-php-6-v-php-7/
41. "RFC: Name of Next Release of PHP". *php.net*. 2014-07-07. Retrieved 2014-07-15.
42. "Re: [PHP-DEV] [VOTE][RFC] Name of Next Release of PHP (again)". 2014-07-30. Retrieved 2014-07-30.
43. http://news.php.net/php.internals/73888
44. "PHP: rfc:phpng". *php.net*. Retrieved 16 December 2014.
45. "PHP: phpng". *php.net*. Retrieved 2014-07-15.
46. "Merge branch 'ZendEngine3' ". *github.com*. 2014-12-05. Retrieved 2014-12-05.
47. "PHP: Release Process". 2011-06-20. Retrieved 2013-10-06.
48. "PHP RFC: Exceptions in the engine (for PHP 7)". *php.net*. Retrieved 2015-05-21.
49. "PHP RFC: Uniform Variable Syntax". *php.net*. 2014-05-31. Retrieved 2014-07-30.
50. "Online PHP shell | 150+ versions + stats + vld + hhvm".
51. "PHP RFC: Fix "foreach" behavior". *php.net*. Retrieved 2015-05-21.
52. "PHP RFC: Constructor behaviour of internal classes". *php.net*. Retrieved 2015-05-21.
53. "PHP RFC: Removal of dead or not yet PHP7 ported SAPIs and extensions". *php.net*. Retrieved 2015-05-21.
54. "PHP RFC: Fix list() behavior inconsistency". *php.net*. Retrieved 2015-05-21.
55. "PHP RFC: Remove alternative PHP tags". *php.net*. Retrieved 2015-05-21.
56. "PHP RFC: Make defining multiple default cases in a switch a syntax error". *php.net*. Retrieved 2015-05-21.
57. "PHP RFC: Remove hex support in numeric strings". *php.net*. Retrieved 2015-05-21.
58. "PHP RFC: Integer Semantics". *php.net*. Retrieved 2015-05-21.
59. "PHP RFC: ZPP Failure on Overflow". *php.net*. Retrieved 2015-05-21.
60. "RFC: Return Types". *php.net*. 2015-01-27. Retrieved 2015-01-28.
61. "RFC: Scalar Type Declarations". *php.net*. 2015-03-16. Retrieved 2015-03-17.
62. "Unsupported Branches". *php.net*. Retrieved 2015-11-14.
63. "PHP: PHP 4 ChangeLog". The PHP Group. 2008-01-03. Retrieved 2008-02-22.
64. "Using PHP from the command line". *PHP Manual*. The PHP Group. Retrieved 2009-09-11.
65. "PHP 4.4.0 Release Announcement". *PHP Mannual*. The PHP Group. Retrieved 2013-11-24.
66. "PHP: PHP 5 ChangeLog". The PHP Group. 2007-11-08. Retrieved 2008-02-22.
67. "PHP manual: PDO". The PHP Group. 2011-11-15. Retrieved 2011-11-15.
68. "Built-in web server". Retrieved March 26, 2012.
69. "PHP 5.5.0 changes". *php.net*. Retrieved 2015-03-03.
70. "Supported Versions". *php.net*. Retrieved 2015-12-02.
71. "Migrating from PHP 5.5.x to PHP 5.6.x". *php.net*. Retrieved 2014-03-24.
72. "Resetting PHP 6". "There have been books on the shelves purporting to cover PHP 6 since at least 2008. But, in March 2010, the PHP 6 release is not out - in fact, it is not even close to out. Recent events suggest that PHP 6 will not be released before 2011 - if, indeed, it is released at all."
73. "PHP 7 moves full speed ahead". "Recent versions of PHP have been part of the 5.x release series, but there will be no PHP 6. “We’re going to skip [version] 6 because years ago, we had plans for a 6 but those plans were very different from what we’re doing now,” Gutmans said. Going right to version 7 avoids confusion."

74. "PHP: rfc:size_t_and_int64_next". *php.net*. Retrieved 16 December 2014.
75. "PHP: rfc:abstract_syntax_tree". *php.net*. Retrieved 16 December 2014.
76. "PHP: rfc:closure_apply". *php.net*. Retrieved 16 December 2014.
77. "PHP: rfc:integer_semantics". *php.net*. Retrieved 16 December 2014.
78. "PHP: rfc:isset_ternary". *php.net*. Retrieved 16 December 2014.
79. "RFC: Unicode Codepoint Escape Syntax". 2014-11-24. Retrieved 2014-12-19.
80. "Combined Comparison (Spaceship) Operator". *php.net*. Retrieved 2015-05-21.
81. "PHP RFC: Generator Delegation". *php.net*. Retrieved 2015-05-21.
82. "PHP RFC: Anonymous Classes". *php.net*. Retrieved 2015-05-21.
83. "PHP RFC: Easy User-land CSPRNG". *php.net*. Retrieved 2015-05-21.
84. "PHP RFC: Group Use Declarations". *php.net*. Retrieved 2015-05-21.
85. "Preparation Tasks". Retrieved 5 July 2016.
86. "PHP: rfc:void_return_type". *php.net*. 2015-11-09. Retrieved 2015-11-14.
87. "PHP: rfc:class_constant_visibility". *php.net*. 2015-10-27. Retrieved 2015-12-08.
88. "PHP RFC: Nullable Types". *php.net*. 2014-04-10. Retrieved 2016-06-14.
89. "PHP: ElePHPant". 4 Oct 2014. Retrieved 4 Oct 2014.
90. https://wwphp-fb.github.io/faq/community/elephpant/
91. "tags - Manual". *php.net*. Retrieved 2014-02-17.
92. "PHP: rfc:shortags". *php.net*. 2008-04-03. Retrieved 2014-05-08.
93. "PHP: Basic syntax". The PHP Group. Retrieved 2008-02-22.
94. "Basic Coding Standard". PHP Framework Interoperability Group. Retrieved 2016-01-03.
95. "echo - Manual". *php.net*. Retrieved 2014-02-17.
96. "Description of core php.ini directives - Manual". *php.net*. 2002-03-17. Retrieved 2014-02-17.
97. "Your first PHP-enabled page". The PHP Group. Retrieved 2008-02-25.
98. Bray, Tim; et al. (26 November 2008). "Processing Instructions". *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. W3C. Retrieved 2009-06-18.
99. "Variables". The PHP Group. Retrieved 2008-03-16.
100. "Instruction separation". The PHP Group. Retrieved 2008-03-16.
101. "Comments". The PHP Group. Retrieved 2008-03-16.
102. "Integers in PHP, running with scissors, and portability". MySQL Performance Blog. March 27, 2007. Retrieved 2007-03-28.
103. "Types". The PHP Group. Retrieved 2008-03-16.
104. "Strings". The PHP Group. Retrieved 2008-03-21.
105. "SPL – StandardPHPLibrary". *PHP.net*. March 16, 2009. Retrieved 2009-03-16.
106. "User-defined functions (PHP manual)". *php.net*. 2014-07-04. Retrieved 2014-07-07.
107. "Variable functions (PHP manual)". *php.net*. 2014-07-04. Retrieved 2014-07-07.
108. "create_function() (PHP manual)". *php.net*. 2014-07-04. Retrieved 2014-07-07.
109. "Anonymous functions (PHP manual)". *php.net*. 2014-07-04. Retrieved 2014-07-07.
110. Christian Seiler; Dmitry Stogov (2008-07-01). "Request for Comments: Lambda functions and closures". *php.net*. Retrieved 2014-07-07.
111. "PHP 5 Object References". *mjtsai.com*. Retrieved 2008-03-16.
112. "Classes and Objects (PHP 5)". The PHP Group. Retrieved 2008-03-16.
113. "Object cloning". The PHP Group. Retrieved 2008-03-16.
114. "Visibility (PHP Manual)". *theserverpages.com*. 2005-05-19. Retrieved 2010-08-26.
115. Gervasio, Alejandro. "More on Private Methods with PHP 5 Member Visibility". *devshed.com*. Retrieved 24 November 2010.
116. "Visibility in PHP: Public, Private and Protected". *aperiplus.sourceforge.net*. Retrieved 2010-08-26.
117. "How do computer languages work?". Retrieved 2009-11-04.
118. (Gilmore 2006, p. 43)
119. "[VOTE] Integrating Zend Optimizer+ into the PHP distribution". *news.php.net*. Retrieved 2013-03-08.
120. "Alternative PHP Cache". *PHP.net*. Retrieved 2013-09-21.
121. "We are the 98.5% (and the 16%) « HipHop Virtual Machine". *hhvm.com*. December 2013. Retrieved 2014-02-23.
122. "Announcement on GitHub removing HPHPc support". Retrieved 2013-05-24.
123. "The PHP License, version 3.01". Retrieved 2010-05-20.
124. "GPL-Incompatible, Free Software Licenses". *Various Licenses and Comments about Them*. Free Software Foundation. Retrieved 2011-01-03.
125. "PHP: Function and Method listing - Manual". The PHP Group. Retrieved 2015-01-14.

126. "Introduction - Manual". *php.net*. 2013-06-07. Retrieved 2013-06-13.
127. Darryl Patterson (5 August 2004). "Simplify Business Logic with PHP DataObjects - O'Reilly Media". *ibm.com*. Retrieved 16 December 2014.
128. "IBM - United States". *IBM - United States*. Retrieved 16 December 2014.
129. "Five common PHP database problems". *ibm.com*. 2006-08-01. Retrieved 2013-06-13.
130. "IBM Redbooks - Developing PHP Applications for IBM Data Servers". *redbooks.ibm.com*. Retrieved 16 December 2014.
131. php|architect (http://www.phparch.com/issue.php?mid=65)
132. Krill, Paul (19 October 2005). "PHP catching on at enterprises, vying with Java". InfoWorld. Archived from the original on 13 July 2014.
133. "Cross Reference: /PHP_5_4/ext/standard/". *php.net*. Retrieved 16 December 2014.
134. "Developing Custom PHP Extensions". *devnewz.com*. 2002-09-09. Archived from the original on 2008-02-18. Retrieved 2008-02-25.
135. "Why Zephir?". *zephir-lang.com*. 2015-10-20. Retrieved 2015-12-14.
136. "PHP Credits". *php.net*. Retrieved 2015-07-29.
137. http://www.zend.com/en/services/certification/php-5-certification
138. "General Installation Considerations". *php.net*. Retrieved 2013-09-22.
139. "News Archive: PHP 5.3.3 Released!". *php.net*. 2010-07-22.
140. "FastCGI Process Manager (FPM)". *php.net*. Retrieved 2013-09-22.
141. "Command line usage: Introduction". *php.net*. Retrieved 2013-09-22.
142. "Command line usage: Differences to other SAPIs". *php.net*. Retrieved 2013-09-22.
143. "General Installation Considerations". *php.net*. Retrieved 2013-09-22.
144. "PHP: Apache 2.x on Microsoft Windows". *php.net*. Retrieved 2013-09-22.
145. "Command line usage: Introduction". *php.net*. Retrieved 2013-09-22.
146. "Installing PHP-GTK 2". *php.net*. Retrieved 2013-09-22.
147. "AWS SDK for PHP". *aws.amazon.com*. Retrieved 2014-03-06.
148. "Windows Azure SDK for PHP - Interoperability Bridges and Labs Center". *interoperabilitybridges.com*. Retrieved 2014-03-06.
149. "Runtime configuration: Table of contents". *php.net*. Retrieved 2013-09-22.
150. "php.ini directives: List of php.ini directives". *php.net*. Retrieved 2013-09-22.
151. "Runtime configuration: The configuration file". PHP.net. Retrieved 2013-09-22.
152. "php.ini directives: List of php.ini sections". PHP.net. Retrieved 2013-09-22.
153. "Runtime configuration: Where a configuration setting may be set". PHP.net. Retrieved 2013-09-22.
154. "PHP Manual Image Processing and GD;". php.net. Retrieved 2011-04-09.
155. Archived (https://web.archive.org/web/20080611231433/http://gtk.php.net/manual1/it/html/intro.whatis.php.whatdoes.html) June 11, 2008, at the Wayback Machine.
156. "PHP and MySQL". University of Alabama. Archived from the original on 2008-02-28. Retrieved 2008-02-25.
157. "PHP Server-Side Scripting Language". Indiana University. 2007-04-04. Retrieved 2008-02-25.
158. "JavaServer Pages Technology — JavaServer Pages Comparing Methods for Server-Side Dynamic Content White Paper". Sun Microsystems. Retrieved 2008-02-25.
159. "Five simple ways to tune your LAMP application".
160. "PHP: PHP Usage Stats". SecuritySpace. 2007-04-01. Retrieved 2008-02-24.
161. "Usage of server-side programming languages for websites". W3Techs. 2010-10-29. Retrieved 2010-10-29.
162. "Usage of server-side programming languages for websites". W3Techs. Retrieved 2014-03-19.
163. "PHP and Perl crashing the enterprise party".
164. "Manual:Installation requirements#PHP". MediaWiki. 2010-01-25. Retrieved 2010-02-26. "PHP is the programming language in which MediaWiki is written [...]"
165. What is Joomla? (http://www.joomla.org/about-joomla.html)
166. "Server requirements of SilverStripe". Retrieved 13 October 2014. "SilverStripe requires PHP 5.3.2+"
167. "About WordPress". Retrieved 2010-02-26. "WordPress was [...] built on PHP"
168. "PHP and Drupal". Drupal.org. Retrieved 2010-06-13.
169. "About". Moodle.org. Retrieved 2009-12-20.
170. "PHP and Facebook | Facebook". Blog.facebook.com. Retrieved 2009-07-29.
171. "PHP and Digg". O'Reilly. Retrieved 2010-06-13.
172. "PHP at the core: Extension structure". *PHP.net*. Retrieved 2013-09-22.
173. "PHP at the core: The "counter" Extension – A Continuing Example". *PHP.net*. Retrieved 2013-09-22.
174. "Extension Writing Part I: Introduction to PHP and Zend". Zend Technologies. 2005-03-01. Retrieved 2013-09-22.

175. "Extension Writing Part II: Parameters, Arrays, and ZVALs". Zend Technologies. 2005-06-06. Retrieved 2013-09-22.
176. "Extension Writing Part II: Parameters, Arrays, and ZVALs (continued)". Zend Technologies. 2005-06-06. Retrieved 2013-09-22.
177. "Extension Writing Part III: Resources". Zend Technologies. 2006-05-12. Retrieved 2013-09-22.
178. "Wrapping C++ Classes in a PHP Extension". Zend Technologies. 2009-04-22. Retrieved 2013-09-22.
179. "Extending PHP with C++?". Stack Overflow. Retrieved 2013-09-22.
180. "How can I use C++ code to interact with PHP?". Stack Overflow. Retrieved 2013-09-22.
181. Golemon, Sara (2006). *Extending and Embedding PHP*. ISBN 978-0-672-32704-9.
182. "Bug Request #46919: Multi threading". *PHP.net*. Retrieved 2013-09-22.
183. "pthreads: Introduction (PHP Manual)". *PHP.net*. Retrieved 2013-09-22.
184. "PECL :: Package :: pthreads". *pecl.php.net*. Retrieved 2014-02-09.
185. Ide, Andy (2013-01-31). "PHP just grows & grows". Retrieved 2013-04-01.
186. "National Vulnerability Database (NVD) Search Vulnerabilities". Retrieved 2014-03-19.
187. "PHP-related vulnerabilities on the National Vulnerability Database". 2012-07-05. Retrieved 2013-04-01.
188. "PHP Taint Mode RFC".
189. "Developer Meeting Notes, Nov. 2005".
190. "Taint mode decision, November 2007".
191. "Hardened-PHP Project". 2008-08-15.
192. "Security: Using Register Globals". *PHP Manual*. PHP.net. Retrieved 2013-09-22.
193. "Magic Quotes". *PHP Manual*. PHP.net. Retrieved 2014-01-17.
194. " 'engine' configuration directive". *PHP: Runtime Configuration*. PHP.net. Retrieved 2014-02-13.
195. "PHP Security Exploit With GIF Images". 2007-06-22. Retrieved 2013-09-22.
196. "PHP security exploit with GIF images". PHP Classes blog. 2007-06-20. Retrieved 2013-09-22.
197. "Passing Malicious PHP Through getimagesize()". 2007-06-04. Retrieved 2013-09-22.
198. " 'enable_dl' configuration directive". *PHP: Runtime Configuration*. PHP.net. Retrieved 2014-02-13.
199. "PHP function reference: dl()". PHP.net. Retrieved 2013-09-22.
200. "My host won't fix their Trojan". WebHosting Talk. Retrieved 2013-09-22.
201. Raz0r. "Simple Machines Forum <= 2.0.3 Admin Password Reset".
202. Nibble Security. "TYPO3-SA-2010-020, TYPO3-SA-2010-022 EXPLAINED".
203. Ahack.ru. "Криптостойкость и небезопасное сравнение".
204. "Comparison operators". PHP.net.
205. Pawel Krawczyk (2013). "Most common attacks on web applications". IPSec.pl. Retrieved 2015-04-15.
206. Pawel Krawczyk (2013). "So what are the "most critical" application flaws? On new OWASP Top 10". IPSec.pl. Retrieved 2015-04-15.

Further reading

- Paul Ford (June 11, 2015). "What is Code?". *Bloomberg Businessweek*. "What's the Absolute Minimum I Must Know About PHP?"

External links

- Official website (http://www.php.net)
- PHP (https://www.dmoz.org/Computers/Programming/Languages/PHP) at DMOZ
- PHP Reference Manual (http://www.php.net/manual)
- PHP source code repository (https://github.com/php/php-src) on GitHub
- PHP PHP and Symfony: Structure, Stability and Flexibility (https://www.symfony.fi/entry/php-and-symfony-structure-stability-and-flexibility)

Categories: Class-based programming languages | Cross-platform software | Dynamically typed programming languages | Filename extensions | Free compilers and interpreters

| High-level programming languages | Internet terminology | Object-oriented programming languages
| PHP software | PHP | Procedural programming languages | Programming languages created in 1995
| Scripting languages | Software using the PHP license | Text-oriented programming languages

---