# 30 Years of PHP: the Brilliant, the Bad, and the Baffling

Gina P. Banyard

The PHP Foundation

2025-06-12

# Happy Birthday PHP



# About me

The PHP Foundation

- BSc in Pure Mathematics from Imperial College London
- Works on PHP since 2019
- Funded by **The PHP Foundation** since 2022
- 26 years old

**Mastodon** `@Girgias@phpc.social`

**Website** `gpb.moe`

**GitHub** `Girgias`

# The Brilliant

# PHP's request model

Fire and forget request model

No need to care about state:

- Closes file handlers, DB and network connections, etc.
- Horizontally scalable

So good that Amazon reinvented PHP with AWS Lambdas

# PHP is pragmatic

No one “true” way of doing something

Takes inspiration and incorporates different paradigms:

- Procedural
- OOP
- Functional

Gradual integration;

- From mildly dynamic HTML pages
- To full on PHP frameworks or CMSs

# Easy to host & deploy

- Super easy to configure and host
- Provides the tools for hosting companies
- Can just FTP files and it works

# Different operators for concatenation & addition



```php
$r = "1" + 1;
```

- JS: `"11"`
- PHP: `2`

# Copy on Write semantics for arrays and strings

Only duplicates data when parameter is written to

- No spooky action at a distance
- No need to pass by reference to prevent copies

# Prefixed variables

Obvious what is a variable

Meta-programming made easy:

```
$class = new $className();
$class->$prop;
$class->$fnName();
```

# Gradual and Runtime type checking

- Allows to write one time scripts without bothering about stuff
- Have an error **at** incorrect call site
  - Rather than at within the function
  - Or worse, deep in a function call stack
- Can enhance it with Static Analysis tools

# Composer/Autoloading

- Fixed the `include`/`require` hell
- Automatic linking of classes without thinking
- Composer is one of the best package managers out there

# The community ❤



# The Bad

# Returning `false` for failure



# Inconsistent naming and parameter order



Organic evolution, so no real consultation

```
function strpos(
  string $haystack, string $needle, int $offset = 0
): int|false;

function in_array(
  mixed $needle, array $haystack, bool $strict = false
): bool;
```

Use name arguments since 8.0.

# PHP 4 to 5 Objects

PHP 4 objects use to be passed by value, like arrays and strings.

To prevent copies needed to pass them by reference:

- PHP used to allow call-site pass by ref for **any** variable:

```php
do_some_stuff_to_my_obj(&$obj);
```

PHP 5 objects became “pass by handler”

- Solves the issue of “reference types”
- But: no more “structs”

# Variable variables

Too much meta programming

```php
$name = "hello";
$hello = "world";
$$name = "Rotterdam";
echo $name . ' ' . $hello;
```

Output:

# Variable variables

Too much meta programming

```php
$name = "hello";
$hello = "world";
$$name = "Rotterdam";
echo $name . ' ' . $hello;
```

Output:

```
hello Rotterdam
```

# Implicit nullable types

```php
interface Fooable {
    function foo(array $f = null);
}
```

`$f` has a type of `array|null`.

Prior to PHP 7.1, the following was allowed (violating LSP)

```php
interface LooseFoo extends Fooable {
    function foo(array $f = []);
}
```

# Ternary op associativity

The ternary operator in PHP used to be left-associative:

```php
return $a == 1 ? 'one'
     : $a == 2 ? 'two'
     : $a == 3 ? 'three'
     : $a == 4 ? 'four'
               : 'other';

return (($a == 1 ? 'one'
     : $a == 2) ? 'two'
     : $a == 3) ? 'three'
     : $a == 4) ? 'four'
               : 'other';
```

# PHP's governance

Free for all prior to RFC process

- Explains some of the weird functions

The RFC process is the **only** process:

- Ill-suited for certain technical issues

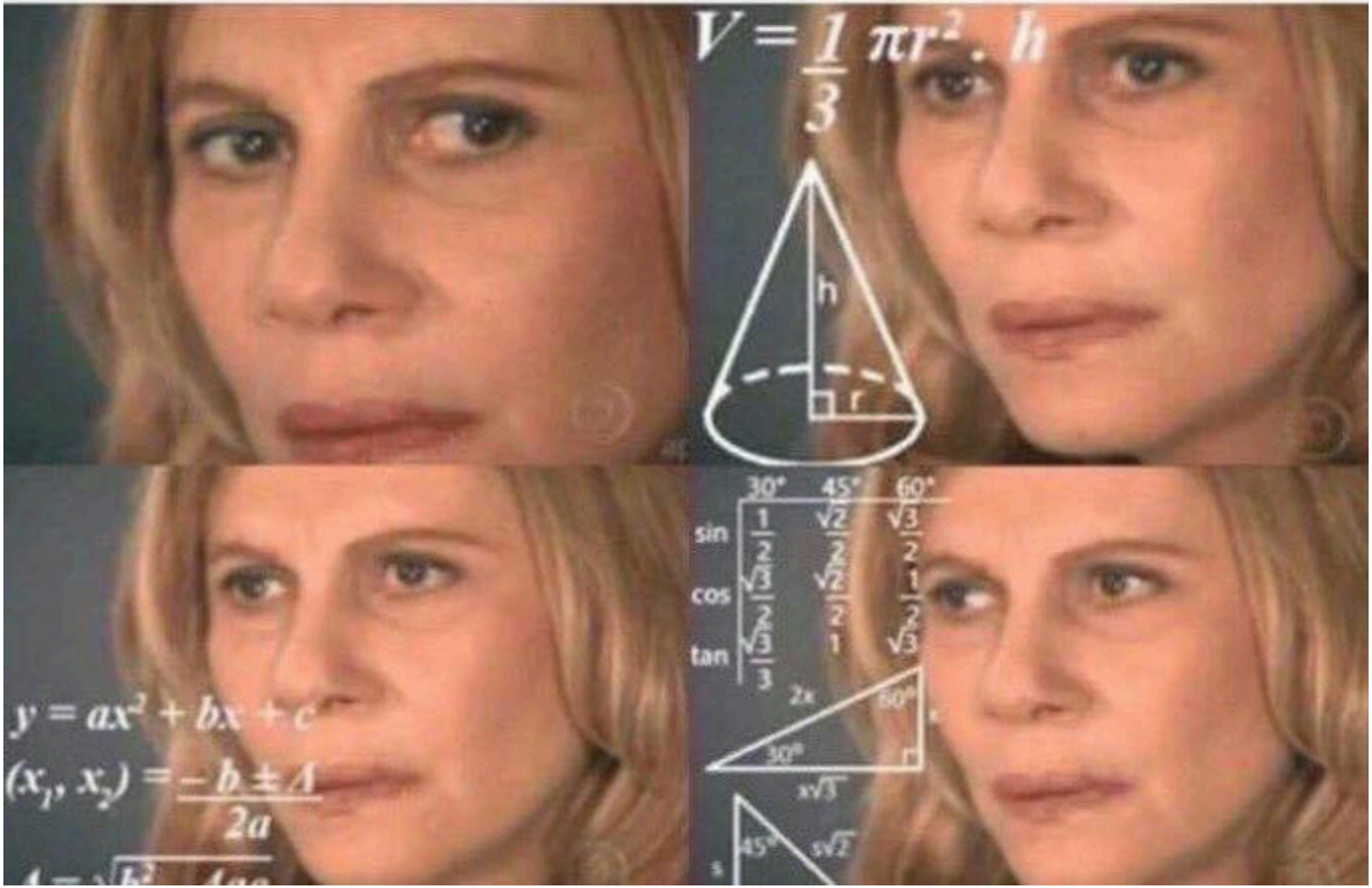
# The Baffling

# PHP 6

Failed native Unicode project

Version number skipped to prevent confusion

Especially as a lot of PHP 6 features were integrated in 5.4


$V = \frac{1}{3}\pi r^2 . h$
h
r
$y = ax^2 + bx + c$
$(x_1, x_2) = \frac{-b \pm \Delta}{2a}$
30° 45° 60°
sin $\frac{1}{2}$ $\frac{\sqrt{2}}{2}$ $\frac{\sqrt{3}}{2}$
cos $\frac{\sqrt{3}}{2}$ $\frac{\sqrt{2}}{2}$ $\frac{1}{2}$
tan $\frac{\sqrt{3}}{3}$ 1 $\sqrt{3}$
2x
60°
30°
x√3
45°
x√2
x

# register_globals

INI setting that registers the content of EGPCS (`$_ENVIRONMENT`, `$_GET`, `$_POST`, `$_COOKIES`, `$_SERVER`) as global variables.

For URL: `http://www.example.com/test.php?page=3`

```
// No need to use $_GET['page'] !
if ($page < 1) {// Error}
// Do something
```

# register_globals

INI setting that registers the content of EGPCS (`$_ENVIRONMENT`, `$_GET`, `$_POST`, `$_COOKIES`, `$_SERVER`) as global variables.

For URL: `http://www.example.com/test.php?page=3`

```
// No need to use $_GET['page'] !
if ($page < 1) {// Error}
// Do something
```

**Massive** security issue:

- No distinction from provenance (e.g. POST or GET)
- User can set *any* global variable just by adding a query parameter e.g. `http://www.example.com/admin.php?is_admin=1`

# Magic quotes

- Convenience feature for inserting HTML form data into DBs
- Automatic escaped ", ', \, "\0" characters from:
  - $_GET
  - $_POST
  - $_COOKIE
  - $_REQUEST

# Magic quotes

- Convenience feature for inserting HTML form data into DBs
- Automatic escaped `"`, `'`, `\`, `"\0"` characters from:
  - `$_GET`
  - `$_POST`
  - `$_COOKIE`
  - `$_REQUEST`
- Massive security risk
- Too broad and impacted data not even intended for DB
- Portability issues as it was controlled via INI setting
- Performance penalty

# mbstring.func_overload

Bit mask value:

- 1: Overload `mail()` into `mb_send_mail()`
- 2: Overload string functions, e.g.:
  - `strlen()` ⇒ `mb_strlen()`
  - `strpos()` ⇒ `mb_strpos()`
  - `substr()` ⇒ `mb_substr()`
- 4: Overload regex functions:
  - `split()` ⇒ `mb_split()`
  - `ereg()` ⇒ `mb_ereg()`
  - `eregi_replace()` ⇒ `mb_eregi_replace()`

# Constant type/value redefinition

```php
class P {
    const FOO = "from P";
}

class C extends P {
    const FOO = 42;
}

$o = new C();
var_dump($o::FOO);
```

# Constant fallback to string

```
start:
try {
  echo Hello . Rotterdam;
} catch (\Error $e) {
  $msg = $e->getMessage();
  if (str_starts_with($msg, "Undefined constant ")) {
    $c = substr($msg, strlen("Undefined constant \""), -1);
    define($c, $c);
    goto start;
  }
}
```

# PERL increment

```php
$s = "Zz";
$s++;
var_dump($s);
```

Output:

# PERL increment

```php
$s = "Zz";
$s++;
var_dump($s);
```

Output:

```
string(3) "AAa"
```

# PERL increment 2

```php
$s = "Zz ";
$s++;
var_dump($s);
```

Output:

# PERL increment 2

```php
$s = "Zz ";
$s++;
var_dump($s);
```

Output:

```
string(3) "Zz "
```

# PERL increment 3

```php
$s = " Zz";
$s++;
var_dump($s);
```

Output:

# PERL increment 3

```php
$s = " Zz";
$s++;
var_dump($s);
```

Output:

```
string(3) " Aa"
```

# PERL increment 4

```php
$s = "4y6";
for ($i = 1; $i < 100; $i++) {
    $s++;
}
var_dump($s);
```

Output:

# PERL increment 4

```
$s = "4y6";
for ($i = 1; $i < 100; $i++) {
    $s++;
}
var_dump($s);
```

Output:

```
float(50)
```

# PERL increment 4: explanation

```php
$s = "5d9";
var_dump(++$s);
var_dump(++$s);
```

Output:

```
string(3) "5e0"
float(6)
```

# disable_classes

```
// php -d disable_classes="Error" test.php

var_dump(strlen([]));
```

Output:

# disable_classes

```
// php -d disable_classes="Error" test.php

var_dump(strlen([]));
```

Output:

```
PHP Notice:  Accessing static property TypeError::$file as
non static in test.php on line 5
PHP Stack trace:
PHP   1. {main}() test.php:0
Segmentation fault (core dumped)
```

# What are your PHP stories?