SEBASTIAN FELDMANN

A BRIEF HISTORY OF PHP

FROM VERSION 4.0 TO THE UPCOMING 8.3

Hello I’m Sebastian

CHECK24

~ 30 years

phpbu

@movetodevnull

A long, time ago in a galaxy not far away….


php4

Things we didn’t have


debug

WordPress

PHPUnit

Objects

```
function load_data($foo) {
    $foo->bar = 2;
}

$foo = new Foo();
$foo->bar = 1;

load_data($foo);

echo $foo->bar;
```

Objects

```
function load_data($foo) {
    $foo->bar = 2;
}

$foo = new Foo();
$foo->bar = 1;

load_data($foo);

echo $foo->bar;


1
```

Objects

```
function load_data($foo) {
    $foo->bar = 2;
}

$foo = new Foo();
$foo->bar = 1;

load_data(&$foo);

echo $foo->bar;


2
```

Objects

```
function load_data($foo) {
    $foo->bar = 2;
}

$foo = new Foo();
$foo->bar = 1;

load_data(&$foo);

echo $foo->bar;


2
```


php5


CakePHP


ZEND
FRAMEWORK

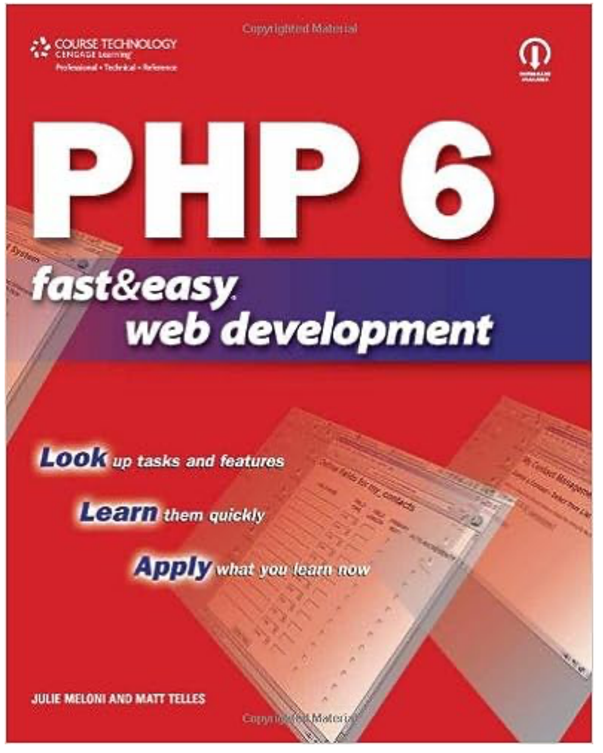

Symfony


php
php
Brasil
php
php
php
php
php
Symfony
ConFoo.CA
php
afup
php

doctrine

PHP-FIG


php6


COURSE TECHNOLOGY
CENGAGE Learning
Professional • Technical • Reference
PHP 6
fast&easy
web development
Look up tasks and features
Learn them quickly
Apply what you learn now
JULIE MELONI AND MATT TELLES


COMPOSER

guzzle

Laravel


Happy 20th
Birthday


5
5


php

PHP-FIG

PSR-0 – Autoloading (deprecated)
PSR-1 – Code Standard
PSR-3 – Logger Interface
PSR-4 – Autoloading
PSR-7 – HTTP Message
PSR-6 – Caching Interface
PSR-16 – Simple Cache

PSR-13 – Link Interface
PSR-11 – Container Interface
PSR-15 – HTTP Request Handler
PSR-17 – Http Factories
PSR-18 – HTTP Client
PSR-12 – Coding Standard
PSR-20 – Clock


php 8

PHP 8.3

Stuff that still „works“

Stuff that still works

```
$foo  = 'Hello World';
$bar  = 'foo';

echo $$bar;
```

Stuff that still works

```
$foo  = 'Hello World';
$bar  = 'foo';

echo $$bar;

Hallo Welt
```

Stuff that still works

```
$foo  = 'Hello World';
$bar  = 'foo';
$buzz = 'bar';

echo $$$buzz;
```

Stuff that still works

```
$foo  = 'Hello World';
$bar  = 'foo';
$buzz = 'bar';

echo $$$buzz;

Hallo Welt
```

Stuff that still works

```
$array = [1, 2, 3];
echo implode(',', $array);

foreach ($array as &$value) {}    // by reference
echo implode(',', $array);

foreach ($array as $value) {}     // by value (i.e., copy)
echo implode(',', $array);
```

Stuff that still works

```
$array = [1, 2, 3];
echo implode(',', $array);

foreach ($array as &$value) {}    // by reference
echo implode(',', $array);

foreach ($array as $value) {}     // by value (i.e., copy)
echo implode(',', $array);
```

```
1,2,3
1,2,3
1,2,2
```

Stuff that still works

```
$array = [1, 2, 3];
echo implode(',', $array);

foreach ($array as &$value) {}    // by reference
echo implode(',', $array);

foreach ($array as $value) {}     // by value (i.e., copy)
echo implode(',', $array);
```

```
1,2,3
1,2,3
1,2,2
```

Stuff that still works

```
$data = [];

$data[] = 'foo';
$data[] = 'bar';
$data[] =  &$baz;

$baz = 'baz';

echo implode(',', $data);
```

Stuff that still works

```
$data = [];

$data[] = 'foo';
$data[] = 'bar';
$data[] =  &$baz;

$baz = 'baz';

echo implode(',', $data);


foo,bar,baz
```

Stuff that still works but… types to the rescue

```php
<?php declare(strict_types=1);
```

The new stuff

Typed class constants

```
class Foo
{
    const string BAR = 'baz';
}
```

Dynamic class constants fetch

```
class Foo
{
    const BAR = 'bar';
}

$name = 'BAR';
```

Dynamic class constants fetch

```
class Foo
{
    const BAR = 'bar';
}

$name = 'BAR';

// instead of this:
constant(Foo::class . '::' . $name);
```

Dynamic class constants fetch

```php
class Foo
{
    const BAR = 'bar';
}

$name = 'BAR';

// instead of this:
constant(Foo::class . '::' . $name);

// you can now do this:
Foo::{$name};
```

Dynamic Enum fetch

```
enum MyEnum: int
{
    case BAR = 42;
}

$name = 'BAR';
```

Dynamic Enum fetch

```php
enum MyEnum: int
{
    case BAR = 42;
}

$name = 'BAR';

// replace this
constant("MyEnum::$enumName")->value;
```

Dynamic Enum fetch

```php
enum MyEnum: int
{
    case BAR = 42;
}

$name = 'BAR';

// replace this
constant("MyEnum::$enumName")->value;

// with this
MyEnum::{$name}->value;
```

#[Override] attribute

```
abstract class Parent
{
    public function methodWithDefaultImplementation(): int
    {
        return 1;
    }
}

final class Child extends Parent
{
    #[Override]
    public function methodWithDefaultImplementation(): int
    {
        return 2;
    }
}
```

#[Override] attribute

```
abstract class Parent
{
    public function methodWithDefaultImplementation(): int
    {
        return 1;
    }
}

final class Child extends Parent
{
    #[Override]
    public function methodWithDefaultImplementation(): int
    {
        return 2;
    }
}
```

#[Override] attribute

```
abstract class Parent
{
    public function methodWithNewImplementation(): int
    {
        return 1;
    }
}

final class Child extends Parent
{
    #[Override]
    public function methodWithDefaultImplementation(): int
    {
        return 2;
    }
}
```

#[Override] attribute

```
abstract class Parent
{
    public function methodWithDefaultImplementation(): int
    {
        return 1;
    }
}

final class Child extends Parent
{
    #[Override]
    public function methodWithDefaultImplementation(): int
    {
        return 2;
    }
}
```

Anonymous `readonly` classes

```
$class = new readonly class
{
    public function __construct(
        public string $foo = 'bar',
    ) {}
};
```

Readonly amendments

```
readonly class Post
{
    public function __construct(
        public DateTime $createdAt,
    ) {}
}
```

Readonly amendments

```php
readonly class Post
{
    public function __construct(
        public DateTime $createdAt,
    ) {}

    public function __clone()
    {
        $this->createdAt = new DateTime();
        // this is allowed,
        // even though `createdAt` is a readonly property.
    }
}
```

New `json_validate()` function

```
$isValid = is_array(json_decode($json, true));
```

New `json_validate()` function

```php
$isValid = is_array(json_decode($json, true));

// php 8.3
$isValid = json_validate($json);
```

New `json_validate()` function

```
json_validate(string $json, int $depth = 512, int $flags = 0): bool
```

Randomizer additions

```
// random string
Randomizer::getBytesFromString(string $string, int $length): string

// return random float between min and max
Randomizer::getFloat(
    float $min,
    float $max,
    IntervalBoundary $boundary = IntervalBoundary::ClosedOpen
): float

// will return float between 0 and 1 excluding 1
Randomizer::getFloat(0, 1, IntervalBoundary::ClosedOpen);
```

Randomizer additions

```
// random string
Randomizer::getBytesFromString(string $string, int $length): string

// return random float between min and max
Randomizer::getFloat(
    float $min,
    float $max,
    IntervalBoundary $boundary = IntervalBoundary::ClosedOpen
): float

// will return float between 0 and 1 excluding 1
Randomizer::getFloat(0, 1, IntervalBoundary::ClosedOpen);
// shorter version
Randomizer::nextFloat(): float
```

More improved error handling

```
unserialize()

range()

// way more ValueError, E_WARNING and E_NOTICE
```

Magic method closures and named arguments

```
class Test {
    public function __call($name, $args)
    {
        var_dump($name, $args);
    }

    public static function __callStatic($name, $args) {
        var_dump($name, $args);
    }
}

$test = new Test();

$closure = $test->magic(...);

$closure(a: 'hello', b: 'world');
```

New `mb_str_pad` function

```
function mb_str_pad(
    string $string,
    int $length,
    string $pad_string = " ",
    int $pad_type = STR_PAD_RIGHT,
    ?string $encoding = null,
): string {}
```

Negative indices in `arrays`

```php
$array = [];

$array[-5] = 'a';
$array[]   = 'b';

var_export($array);
```

Negative indices in `arrays`

```php
$array = [];

$array[-5] = 'a';
$array[]   = 'b';

var_export($array);
```

```
array (
  -5 => 'a',
   0 => 'b',
)
```

Negative indices in `arrays`

```
$array = [];

$array[-5] = 'a';
$array[]   = 'b';

var_export($array);


array (
  -5 => 'a',
  -4 => 'b',
)
```

Specialized Date/Time Exceptions

```
DateMalformedIntervalStringException

DateInvalidOperationException

DateRangeError
```

Specialized Date/Time Exceptions

- The `Epoch doesn't fit in a PHP integer` now returns a new `DateRangeError` instead of a generic `ValueError`, which it does not subclass. This is only an issue for 32-bit platforms.

- The `Only non-special relative time specifications are supported for subtraction` warning with `DateTime::sub()` and `date_sub()` becomes a new `DateInvalidOperationException`.

- The `Unknown or bad format (%s) at position %d (%c): %s` and `String '%s' contains non-relative elements` warnings that are created while parsing wrong/broken `DateInterval` strings will now throw a new `DateMalformedIntervalStringException` when used with the OO interface, instead of showing a warning and returning false.

Invariant constant visibility

```
interface I {
    public const FOO = 'foo';
}

class C implements I {
    private const FOO = 'foo';
}
```

Invariant constant visibility

```
interface I {
    public const FOO = 'foo';
}

class C implements I {
    private const FOO = 'foo';
}


Fatal error: Access level to C::FOO must be public (as in interface I)
```

when?

November 23, 2023

Test the Release Candidates

@movetodevnull