



PHP 7.0 / 7.1 / 7.2

New Features By Code

Daniel Schröder - Freelancer - WebDevOps
@schroedan

ANYONE NOT USING



PHP7 IN THE ROOM?

TWICE AS FAST AS



PHP5 IT IS

Scalar type and return type definitions, nullables, iterables

```
function sumOfInts(int ...$ints): int
{
    return array_sum($ints);
}
```

```
function testReturn(): ?string
{
    return 'elePHPant';
}
```

```
function doSomething(iterable $array = ['work', 'work', 'work']): void
{
    echo implode(' ', (array)$array);
}
```

```
function createObject(): object
{
    return new \stdClass();
}
```

Null coalescing operator, spaceship operator

```
// Fetches the value of $_GET['user'] and returns 'nobody' if it does not exist.  
$username = $_GET['user'] ?? 'nobody';
```

```
// This is equivalent to:  
$username = isset($_GET['user']) ? $_GET['user'] : 'nobody';
```

```
// Integers
```

```
echo 1 <=> 1; // 0  
echo 1 <=> 2; // -1  
echo 2 <=> 1; // 1
```

```
// Floats
```

```
echo 1.5 <=> 1.5; // 0  
echo 1.5 <=> 2.5; // -1  
echo 2.5 <=> 1.5; // 1
```

```
// Strings
```

```
echo "a" <=> "a"; // 0  
echo "a" <=> "b"; // -1  
echo "b" <=> "a"; // 1
```

Anonymous classes

```
interface Logger {
    public function log(string $msg);
}

class Application {
    private $logger;

    public function setLogger(Logger $logger) {
        $this->logger = $logger;
    }
}

$app = new Application;
$app->setLogger(new class implements Logger {
    public function log(string $msg) {
        echo $msg;
    }
});
```

Constant arrays, filtered unserialize, negative string offsets

```
define('ANIMALS', [
    'dog',
    'cat',
    'bird'
]);

// converts all objects into __PHP_Incomplete_Class object
$data = unserialize($foo, ["allowed_classes" => false]);

// converts all objects into __PHP_Incomplete_Class object except those of MyClass and MyClass2
$data = unserialize($foo, ["allowed_classes" => ["MyClass", "MyClass2"]]);

// default behaviour (same as omitting the second argument) that accepts all classes
$data = unserialize($foo, ["allowed_classes" => true]);

var_dump("abcdef"[-2]);
var_dump(strpos("aabbcc", "b", -3));
```

Grouped use declarations (with trailing commas)

```
// Pre PHP 7 code
use some\namespace\ClassA;
use some\namespace\ClassB;
use some\namespace\ClassC as C;

use function some\namespace\fn_a;
use function some\namespace\fn_b;
use function some\namespace\fn_c;

use const some\namespace\ConstA;
use const some\namespace\ConstB;
use const some\namespace\ConstC;

// PHP 7+ code
use some\namespace\{ClassA, ClassB, ClassC as C};
use function some\namespace\{fn_a, fn_b, fn_c};
use const some\namespace\{
    ConstA,
    ConstB,
    ConstC,
};
```


Symmetric array destructuring

```
$data = [  
  [1, 'Tom'],  
  [2, 'Fred'],  
];  
  
// list() style  
list($id1, $name1) = $data[0];  
  
// [] style  
[$id1, $name1] = $data[0];  
  
// list() style  
foreach ($data as list($id, $name)) {  
  // logic here with $id and $name  
}  
  
// [] style  
foreach ($data as [$id, $name]) {  
  // logic here with $id and $name  
}
```

... with support for keys in list

```
$data = [  
    ["id" => 1, "name" => 'Tom'],  
    ["id" => 2, "name" => 'Fred'],  
];  
  
// list() style  
list("id" => $id1, "name" => $name1) = $data[0];  
  
// [] style  
["id" => $id1, "name" => $name1] = $data[0];  
  
// list() style  
foreach ($data as list("id" => $id, "name" => $name)) {  
    // logic here with $id and $name  
}  
  
// [] style  
foreach ($data as ["id" => $id, "name" => $name]) {  
    // logic here with $id and $name  
}
```

Class constant visibility, multi catch exceptions

```
class ConstDemo
{
    const PUBLIC_CONST_A = 1;
    public const PUBLIC_CONST_B = 2;
    protected const PROTECTED_CONST = 3;
    private const PRIVATE_CONST = 4;
}

try {
    // some code
} catch (FirstException | SecondException $e) {
    // handle first and second exceptions
}
```

Abstract method overriding, parameter type widening

```
abstract class A
{
    abstract function test(string $s);
}
abstract class B extends A
{
    // overridden - still maintaining contravariance for parameters and covariance for return
    abstract function test($s) : int;
}

interface A
{
    public function Test(array $input);
}

class B implements A
{
    public function Test($input){} // type omitted for $input
}
```

References

<http://php.net/manual/en/appendices.php>

<http://php.net/manual/en/migration70.new-features.php>

<http://php.net/manual/en/migration71.new-features.php>

<http://php.net/manual/en/migration72.new-features.php>



NOW, ARE THERE
ANY QUESTIONS?