

PHP8 Variable Types

| | |
|----------------------|---|
| Class/interface name | An instance of the class or interface |
| self | The same class the type is used in |
| parent | The parent of the class the type is used in |
| callable | Something callable |
| bool | A boolean |
| int | An integer |
| float | A float |
| string | A string |
| array | An array |
| iterable | Array or traversable |
| object | An object |
| mixed | Any value |

Return Types

| | |
|--------|-----------------------------------|
| void | Function will not return anything |
| static | Function may return static |

New mixed type

Mixed means any one of:

| | | |
|--------|------|----------|
| array | bool | callable |
| string | int | resource |
| float | null | object |

Note that because `mixed` already contains `null`, it is not nullable itself.

Return static

```
class Foo
{
    public function test():
    static
    {
        return new
    static();
    }
}
```

Static is now a valid return type.

The nullsafe operator (copy)

```
$foo = $bar->getBaz()?->asFormatted();
```

Quickly check for nulls in method returns.

Union types

```
public function foo(Foo|Bar
$input): int|float;
```

You can now specify more than one type in a type hint:

- The `void` type cannot be part of a union.
- Unions can be nullable by including `null` as a type or preceding with a question mark.

Named arguments

```
function foo(string $a, int $b,
?string $c = null)
{
    ...
}
foo(
    c: 'value c',
    a: 'value a'
);
```

Pass values to functions and methods by name, regardless of the order in which they are defined.



By **Dave Child** (DaveChild)
cheatography.com/davechild/
aloneonahill.com

Published 31st March, 2021.
 Last updated 31st March, 2021.
 Page 1 of 3.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Attributes

```
use
App\Attributes\ExampleAttribute;
#[ExampleAttribute]
class Foo
{
    #[ExampleAttribute]
    public const FOO =
'foo';

    #[ExampleAttribute]
    public $x;

    #[ExampleAttribute]
    public function foo(#[ExampleAttribute] $bar) { }
```

Structured metadata

Match expression

```
$result = match($input) {
    0 => "hello",
    '1', '2', '3' => "world",
};
```

Switch statements made simple. Note, the comparisons made by `match` use strict types.

Constructor property promotion

```
class Money
{
    public function __construct(
        public Currency
    $currency,
        public int
    $amount,
    ) {}
}
```

In the above example `Currency` and `int` are both automatically made public properties of the class and assigned values on construction.

Weak maps

```
class Foo
{
    private WeakMap $cache;

    public function getSomethingWithCaching(object $obj): object
    {
        return $this->cache[$obj] ??=
        $this->computeSomethingExpensive($obj);
    }
}
```

A weak map allows for storage of key-value pairs, but the PHP garbage collection will drop a weak map element where the only reference to it is the weak map itself.

Throw expression

```
$triggerError = fn () => throw
new MyError();
$foo = $bar['offset'] ?? throw
new CustomException('offset');
```

Throw is now an expression instead of a statement, making it more flexible.

Non-capturing catches

```
try {
    // Something goes wrong
} catch (MySpecialException)
{
    Log::error("Something went wrong");
}
```

When you catch exceptions and throwables, you no longer need a variable.

::class on objects

```
$foo = new Foo();
var_dump($foo::class);
```

Works the same as `get_class()`.



By **Dave Child** (DaveChild)
cheatography.com/davechild/
altheonahill.com

Published 31st March, 2021.
 Last updated 31st March, 2021.
 Page 2 of 3.

Sponsored by **Readable.com**
 Measure your website readability!
<https://readable.com>

Trailing comma in parameter lists

```
public function(  
    string $parameterA,  
    int $parameterB,  
    Foo $objectfoo,  
) {  
    // ...  
}
```

You can now end a parameter list with a comma without an error.

Create DateTime objects from interfaces

```
DateTime::createFromInterface(DateTimeInterface  
$other);  
DateTimeImmutable::createFromInterface(  
DateTimeInterface $other);
```

New Stringable interface

```
class Foo  
{  
    public function __toString(): string  
    {  
        return 'foo';  
    }  
}
```

New Stringable interface (cont)

```
> }  
}  
function bar(string|Stringable $stringable) {  
    .../  
    bar(new Foo());  
    bar('abc');  
}
```

New functions

```
str_contains('haystack', 'needle')  
str_starts_with('haystack', 'needle')
```

```
str_ends_with('haystack', 'needle')
```

```
fdiv()
```

```
get_debug_type()
```

```
get_resource_id()
```

Breaking Changes

Internal errors and engine warnings have been made consistent and/or reclassified.

The @ operator no longer silences fatal errors.

The default error reporting level is now E_ALL.

The default PDO error mode is no longer silent.

Concatenation precedence has changed - addition now precedes concatenation.

Namespaces can now include reserved names.

ReflectionFunction::__isCallable() has been deprecated.

ReflectionParameter::getClass() has been deprecated.

There are more breaking and engine changes, this list is limited to those you are most likely to encounter. For a full rundown, check out Brent's article linked below or without an error.

With Thanks

This is a condensed version of the excellent PHP8 cheat sheet you can find by Brent at [stitcher](#).

Fetch the numeric ID of a resource



By **Dave Child** (DaveChild)
cheatography.com/davechild/
[aloneonahill.com](https://davechild.aloneonahill.com)

Published 31st March, 2021.
Last updated 31st March, 2021.
Page 3 of 3.

Sponsored by **Readable.com**
Measure your website readability!
<https://readable.com>