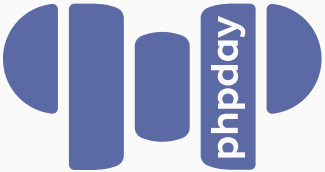
PHP's Type System Dissected

Understanding how PHP's type system works

George Peter Banyard

May 19, 2023

The PHP Foundation



About me

Mastodon: @Girgias@phpc.social

GitHub: Girgias

Site: *https://gpb.moe*

- Studied pure mathematics
- PHP Core dev financed part-time by The PHP Foundation
- Cares about the type system [6] [4] [2] [1]
- Cares about PHP semantics [9] [8] [7] [5] [3]
- Likes anime

Table of Contents

PHP’s Type System

What is a type system?

A type system is a logical system comprising a set of rules that assigns a property called a ***type*** *to every "term".*
A type system dictates the ***operations*** *that can be performed on a term. [14]*

PHP's Type System

The available types are:

- Atomic types
  - Primitive types
  - User defined types
  - Value types
  - The `callable` type
- Composite types
- Type aliases

Primitive types

Universal type	`mixed` (PHP 8.0)
Resource type	
Object type	`object` (PHP 7.2)
Hash table type	`array`
Scalar types	`bool`, `int`, `float`, `string`
Unit type	`null` (PHP 8.0*)
Empty type	`never` (PHP 8.1)

And a special return only type:

`void` (PHP 7.1)

User Defined Types

Also called class-types, they are:

- **Interfaces**
- **Classes**
- **Enumerations** (PHP 8.1)

Relative class types:

- `self`
- `parent`
- `static` (PHP 8.0 as a return type only)

Value types

A value type is a concrete subtype of a type.

- `false` (PHP 8.0*)
- `true` (PHP 8.2)

Warning
It's impossible to define a value type in userland. Create an enumeration instead.

The `callable` type

Type which represents a function:

- A string of characters: *`"strlen"`*
- An object/method pair: *`[$instance, "method"]`*
- An object which implements *`__invoke()`*
- A Closure, obtainable with the syntax: *`strlen(...)`* (as of PHP 8.1)

Warning
It's impossible to define a class property as **`callable`**

Composites types

A composite type is a type combining multiple atomic types.

Intersection type: `A&B` (PHP 8.1)

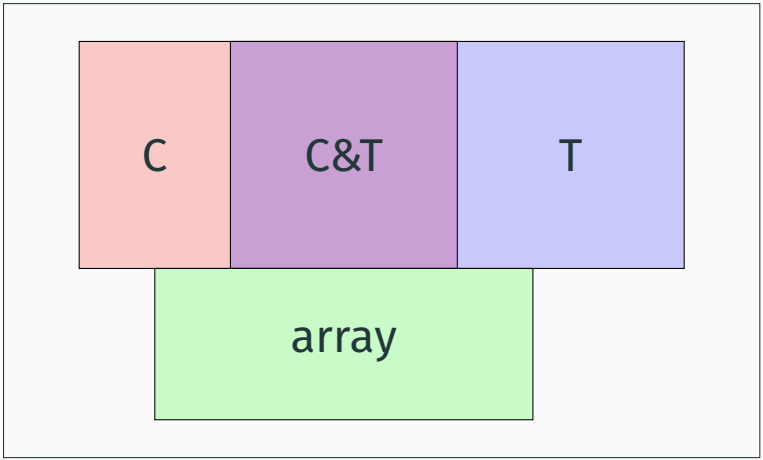
Simple union type: `T|U` (PHP 8.0)

DNF type: `(X&Y)|(V&W)` (PHP 8.2)

Disjunctive normal form
In boolean logic, a Disjunctive Normal Form or DNF is a canonical normal form of a logical formula consisting of a disjunction of conjunctions; it can also be described as an *OR* of *AND*s. [10]

Composites types: why do we care?

Types



Composites types: why do we care?

intersection of types gives us access to every API provided by each type

union of types gives us access only to the common API between each type

Type Alias

As of PHP 8.2, **`iterable`** type alias resolved at compile time.

$$\texttt{iterable} := \texttt{Traversable}|\texttt{array}$$

Before it was a pseudo primitive type.

Warning
It's impossible to define a type alias in userland.

Values are Zvals

```
struct _zval_struct {
    zend_value value;
    uint32_t type_info;
};
```

```
#define IS_NULL      1
#define IS_FALSE     2
#define IS_TRUE      3
#define IS_LONG      4
#define IS_DOUBLE    5
#define IS_STRING    6
#define IS_ARRAY     7
#define IS_OBJECT    8
#define IS_RESOURCE  9
/* ... */
```

zend_type the internal representation of a type

The type of a parameter, return value, or property is represented by a zend_type.

```
typedef struct {
    void *ptr;
    uint32_t type_mask; // Bit-mask of primitive types
} zend_type;
```

ptr is either a class name as a string, or a list of types:

```
typedef struct {
    uint32_t num_types;
    zend_type types[1];
} zend_type_list;
```

Subtyping and Liskov Substitution Principle

What is a subtyping?

In programming language theory, subtyping is a form of type polymorphism in which a subtype is a datatype that is related to another datatype (the supertype) by some notion of substitutability.
If S is a subtype of T, the subtyping relation (written as $S <: T$, $S \sqsubseteq T$, or $S \leq: T$) means that any term of type S can safely be used in any context where a term of type T is expected. [13]

Liskov Substitution Principle

The Liskov Substitution Principle or LSP is a particular definition of a subtyping relation, called strong behavioural subtyping. It was formulated by Barbara Liskov and Jeannette Wing in 1994. [11]

The succinct formulation is:

> *Let* $\phi(\mathsf{x})$ *be a property provable about objects* x *of type* T. *Then* $\phi(\mathsf{y})$ *should be true for objects* y *of type* S *where* S *is a subtype of* T.

Liskov Substitution Principle: Simplified

LSP is a principle about the replacement of a type with another such that the interactions before and after are not affected.

Pre-conditions	cannot be strengthened in the subtype
Post-conditions	cannot be weakened in the subtype
Invariants	must be preserved in the subtype
History rule	constraints must be preserved in the subtype

Liskov Substitution Principle: Visualized

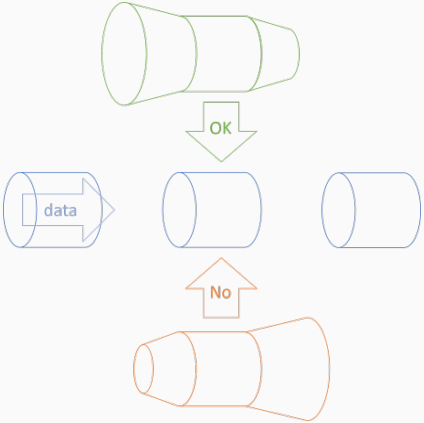


Figure 1: Visualization of Liskov Substitution Principle as a pipe. [12]

Effect of LSP on signatures

Methods cannot add mandatory parameters.

Parameter types of methods must be *contra-variant*, i.e. a supertype.

The return type of methods must be *co-variant*, i.e. a subtype.

Property types must be *co* **and** *contra-variant*.

Covariance of types

In general $S <: T$ if:

- S intersects with a new type U
- S removes a type T_i from a union type $T = T_1|...|T_n$

Covariance of types: Examples

```
class Super1 {
    public function foo(): T|S|U|V {}
}
class Sub1 extends Super1 {
    public function foo(): U|V {}
}
```

Covariance of types: Examples

```
class Super2 {
    public function foo(): A&B {}
}
class Sub2 extends Super2 {
    public function foo(): A&B&C&D {}
}
```

Covariance of types: Examples

```
interface A {}
interface B {}
class X implements A, B {}
class Y implements A, B {}

class Super3 {
    public function foo(): A {}
}
class Sub3 extends Super3 {
    public function foo(): X|Y {}
}
```

Covariance of types: Examples

```
interface A {}
interface B {}
class X implements A, B {}
class Y implements A, B {}

class Super4 {
    public function foo(): A&B {}
}
class Sub4 extends Super4 {
    public function foo(): X|Y {}
}
```

Covariance of types: Examples

```
interface A {}
interface B {}
class X implements A, B {}
class Y implements A, B {}

class Super5 {
    function foo(): (A&B)|D {}
}
class Sub5 extends Super5 {
    function foo(): X|Y|D {}
}
```

Covariance of types: Examples

```
interface A {}
interface B {}
interface C {}

interface X extends A {}

class Super6 {
    public function foo(): A|B {}
}
class Sub6 extends Super6 {
    public function foo(): X&C {}
}
```

U a union type, is $U <: V$?

$\forall$ means "For All", $\exists$ means "There exists".

Let $U = \{U_1, \cdots, U_n\}$ and $V = \{V_1, \cdots, V_m\}$ be a set of types.

$$U_1 \mid \cdots \mid U_n <: V_1 \mid \cdots \mid V_m \impliedby \forall U_i, \exists V_j : U_i <: V_j \tag{1}$$

$$U_1 \mid \cdots \mid U_n <: V_1 \,\&\, \cdots \,\&\, V_m \impliedby \forall U_i, \forall V_j : U_i <: V_j \tag{2}$$

Iterate over the types of U, and verify if U_i is a subtype of V.

PHP implementation of U a union type, is $U <: V$?

```
$u = [$u1, $u2, ..., $uN];
$v = [$v1, $v2, ..., $vM];
$early_status_exit = false;

foreach ($u as $type) {
    if (is_intersection_type($type)) {
        $status = is_intersection_subtype_of_type($type, $v);
    } else {
        $status = is_single_type_subtype_of_type($type, $v);
    }
    if ($status == $early_exit_status) {
        return $status;
    }
}
```

U an intersection type, is $U <: V$?

Let $U = \{U_1, \cdots, U_n\}$ and $V = \{V_1, \cdots, V_m\}$ be a set of types.

$$U_1 \,\&\, \cdots \,\&\, U_n <: V_1 \mid \cdots \mid V_m \impliedby \exists V_j, \exists U_i : U_i <: V_j \tag{3}$$

$$U_1 \,\&\, \cdots \,\&\, U_n <: V_1 \,\&\, \cdots \,\&\, V_m \impliedby \forall V_j, \exists U_i : U_i <: V_j \tag{4}$$

As the order of quantifiers is inverted compared to the union type case, we first need to iterate on V. If V is a union type, it suffices to have one $(i; j)$ pair that satisfies $U_i <: V_j$ for $U <: V$. Otherwise, each V_j needs to be satisfied by at least one U_i for $U <: V$.

PHP implementation of U an intersection type, is $U <: V$?

```
function is_intersection_subtype_of_type($sub, $super) {
    $early_status_exit = !is_intersection_type($super);

    foreach ($super as $single) {
        if (is_intersection_type($single)) {
            $status = is_intersection_subtype_of_type($sub, $single);
        } else {
            $status = is_intersection_subtype_of_class($sub, $single);
        }
        if ($status == $early_exit_status) { return $status; }
    }
    return !$early_status_exit;
}
```

The Future of PHP's type system?

- Function types

```
foo(fn<int,string>:bool $callable) {}
```

The Future of PHP's type system?

- Generic types

```
class Collection<T> {
    private array<T> $stack = [];
    public function add(T $v) {
        $this->stack[] $v;
    }
}
```

The Future of PHP's type system?

- User defined type aliases

```
typedef numeric int|float
type numeric as int|float
```

The Future of PHP's type system?

- *in-out* parameters

```
function foo(inout array $v) { $v = 5; }
$a = [];
foo($a);
// TypeError: inout argument 1 passed to foo() must be of
↪  the type array, int assigned
```

Type coercion/juggling in PHP

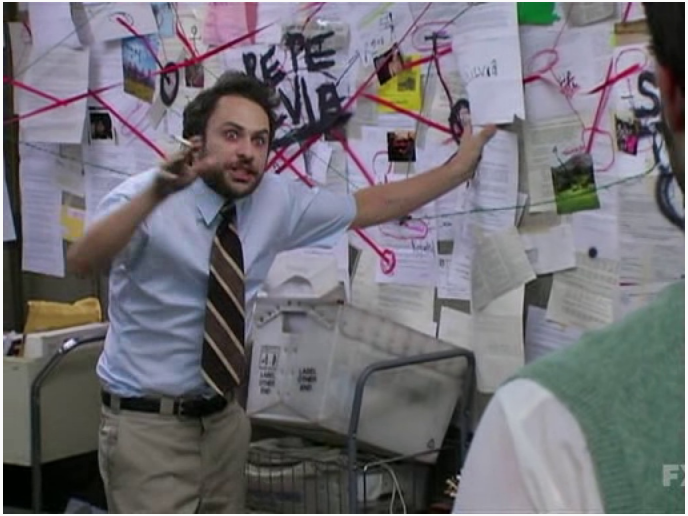


Figure 2: Episode 10 'Sweet Dee Has a Heart Attack', *It's Always Sunny in Philadelphia*, Season 4, [television program] Dir. Matt Shakman. n.k., United States of America, 30/10/2008, Fox TV. 25mins. 00:17:08.

Audience Question

Audience Question

- Who has heard about *`strict_types`*?

Audience Question

- Who knows what *`strict_types`* does?

Type Juggling Contexts in PHP

1. String
2. Integral and String
3. Numeric
4. Logical
5. Comparative
6. Function
7. Increment/Decrement operators
8. Array offsets
9. String offsets
10. `exit` construct

String Type Juggling Context

This is the context when using echo, print, string interpolation, or the string concatenation operator.

```php
var_dump(true . " hello"); // string(7) "1 hello"
$a = [1, 2, 3];
$str = $a . " are great"; // Warning: Array to string conversion
var_dump($str); // string(15) "Array are great"
$o = new stdClass();
echo $o; // TypeError
```

Integral and String Type Juggling Context

This is the context when using a bitwise operators.

```php
var_dump("123" | "abc"); // string(3) "qrs"
var_dump("123" | "1e3"); // string(3) "1w3"
var_dump(36 | "123"); // int(127)
var_dump(36 | "1e3"); // int(1004)
var_dump(36 | true); // int(37)
var_dump(36 & null); // int(0)
var_dump(36 | gmp_init(25));
// object(GMP)#2 (1) { ["num"]=> string(2) "61" }
$o = tidy_parse_string("<p>Hello world</p>");
var_dump(36 | $o); // int(36)
var_dump("abc" | 36);
// TypeError: Unsupported operand types: string | int
```

Numeric Type Juggling Context

This is the context when using an arithmetical operator.

```php
var_dump("123" + "1e3"); // float(1123)
var_dump(36 + "123"); // int(159)
var_dump(36 + "1e3"); // float(1036)
var_dump(36 + true); // int(37)
var_dump(36 + null); // int(36)
var_dump(36 + gmp_init(25));
// object(GMP)#2 (1) { ["num"]=> string(2) "61" }
$o = tidy_parse_string("<p>Hello world</p>");
var_dump(36 + $o); // int(36)
```

Logical Type Juggling Context

This is the context when using conditional statements, the ternary operator, or a logical operator.

Logical Type Juggling Context

This is the context when using conditional statements, the ternary operator, or a logical operator.

```php
$o = gmp_init(10);
if ($o) {
    echo "Hello";
}
// Recoverable fatal error: Object of class GMP could not be
↪  converted to bool in %s on line 3
```

Episode 2 'Nice to Meet You, Euphonium', *Sound! Euphonium*, Season 1, [television program, BluRay]
Dir. Tatsuya Ishihara. Kyoto Animation, Japan, 15/04/2015, Tokyo MX. 23mins 41secs. 00:12:33.

Comparative Context

This is the context when using a comparison operator:

Equal	`==`
Different	`!=` or `<>`
Greater than	`<`
Less than	`>`
Greater or equal than	`<=`
Less or equal than	`>=`
Spaceship	`<=>`

Type OP 1	Type OP 2	Result
`string|null`	`string`	Convert **`null`** to **`""`**, numerical or lexical comparison
`bool|null`	`mixed`	Convert both sides to **`bool`**, *false* < *true*
`object`	`object`	Built-in classes can define their own comparisons, same classes compare properties, otherwise incomparable
`string|` `resource|` `int|float`	`string|` `resource|` `int|float`	Translate strings and resources to numbers, usual math
`array`	`array`	Array with fewer members is smaller, if key from OP 1 is not found in OP 2 then arrays are incomparable, otherwise - compare value by value
`object`	`mixed`	**`object`** is always greater
`array`	`mixed`	**`array`** is always greater

Comparative Context

Implications:

- `[] <=> true: OP1 < OP2`
- `"1" <=> "01": OP1 = OP2`
- `"0e5" <=> "0e9": OP1 = OP2`
- `[15, 20] <=> ["a"=>"a", "b"=>"b"]: OP1 > OP2`
  `["a"=>"a", "b"=>"b"] <=> [15, 20]: OP1 > OP2`
- `new stdClass() <=> tidy_parse_string("<p>Hello world</p>"): OP1 > OP2`
  `tidy_parse_string("<p>Hello world</p>") <=> new stdClass(): OP1 > OP2`

Comparative Context: this is fine

```
new stdClass() <=> gmp_init(250): OP1 > OP2
gmp_init(250) <=> new stdClass(): T
TypeError: Number must be of type GMP|string|int, stdClass
 ↪  given
```

Comparative Context: this is fine

```
3 <=> new stdClass():
PHP Notice:  Object of class stdClass could not be converted
↪  to int
OP1 > OP2
3 <=> tidy_parse_string("<p>Hello world</p>"): OP1 > OP2

5.5 <=> gmp_init(250): OP1 < OP2
"25.9" <=> gmp_init(250): T
ValueError: Number is not an integer string
```

Function Type Juggling Context

This is the context when a value is passed to a typed parameter, property, or returned from a function which declares a return type.

In this context, the value must be a value of the type.

Function Type Juggling Context

This is the context when a value is passed to a typed parameter, property, or returned from a function which declares a return type.

In this context, the value must be a value of the type.

Exceptions:

1. **`int`** to **`float`** promotion
2. The type and the value are scalars, the value gets converted to the appropriate type if compatible
3. The **`string`** type accepts objects that are castable to **`string`**

Note
Internal functions also coerce **`null`** for scalar type declarations, this is deprecated as of PHP 8.1.

`int` to `float` Promotion

```php
function something_with_float(float $f) {
    var_dump($f);
}

something_with_float(15); // float(15)
```

Coercion for Scalar Types

If multiple scalars types are allowed, the order is the following:

1. `int`
2. `float`
3. `string`
4. `bool`

Coercion for Scalar Types: `float` to `int|string`

```php
function thing_with_int_or_string(int|string $v) {
    var_dump($v);
}
thing_with_int_or_string(15.6);
// Deprecated: Implicit conversion from float 15.6 to int
↪  loses precision
// int(15)
```

Coercion for Scalar Types: `string` to `int|float`

Note
If the value is a **`string`** and the declared type has **`int`** and **`float`** then the numeric string semantics decide of the destination type.

```php
function thing_with_int_or_float(int|float $v) {
    var_dump($v);
}

thing_with_int_or_float("15.6"); // float(15.6)
thing_with_int_or_float("25"); // int(25)
```

`string` Castable Objects

```php
class StringableClass {
    public function __toString(): string {
        return "Some string";
    }
}

function foo(string $v) {
    var_dump($v);
}
$o = new StringableClass();
foo($o); // string(11) "Some string"
```

`string` Castable Objects

```php
function foo(string $v) {
    var_dump($v);
}

$o = gmp_init(15);
var_dump($o instanceof Stringable);         // bool(false)
var_dump(method_exists($o, "__toString")); // bool(false)
var_dump((string) $o); // string(2) "15"
foo($o); // string(2) "15"
```

Increment and Decrement operators

```php
$i = 42;
$f = -20.6;

var_dump(++$i); // int(43)
var_dump(++$f); // float(-19.6)
var_dump(--$i); // int(42)
var_dump(--$f); // float(-20.6)
```

Increment and Decrement operators

```php
$i = 42;
$f = -20.6;

var_dump(++$i); // int(43)
var_dump(++$f); // float(-19.6)
var_dump(--$i); // int(42)
var_dump(--$f); // float(-20.6)

$s = "dz";
var_dump(++$s); // string(2) "ea"
```

Increment and Decrement operators type juggling

```php
$array = [];
var_dump(++$array); // TypeError: Cannot increment array
var_dump(--$array); // TypeError: Cannot decrement array

$resource = STDERR;
var_dump(++$resource); // TypeError: Cannot increment resource
var_dump(--$resource); // TypeError: Cannot decrement resource
```

Increment and Decrement operators type juggling

```php
$false = false;
var_dump(++$false); // bool(false)
var_dump(--$false); // bool(false)
$true = true;
var_dump(++$true); // bool(true)
var_dump(--$true); // bool(true)
```

Increment and Decrement operators type juggling

```php
$stringInt = "10";
var_dump(++$stringInt); // int(11)
var_dump(--$stringInt); // int(9)
$stringFloat = "5.7";
var_dump(++$stringFloat); // float(6.7)
var_dump(--$stringFloat); // float(4.7)
```

Increment and Decrement operators with objects

```php
$o = gmp_init(36);
var_dump(++$o); /*
object(GMP)#2 (1) {
  ["num"]=>
  string(2) "37"
} */

$o = tidy_parse_string("<p>Hello world</p>");
var_dump(++$o);
// Fatal error: Uncaught TypeError: Cannot increment tidy
```

Increment and Decrement operators with `null`

Decrement:

```php
$n = null;
--$n;
var_dump($n); // NULL
```

Increment and Decrement operators with `null`

Decrement:

```php
$n = null;
--$n;
var_dump($n); // NULL
```

Increment:

```php
$n = null;
++$n;
var_dump($n); // int(1)
```

Increment and Decrement operators with non-numeric `strings`

Decrement:

```php
$s = "foo";
var_dump(--$s;); // string(3) "foo"
$e = "";
var_dump(--$e); // int(-1)
```

Increment and Decrement operators with non-numeric `strings`

Decrement:

```php
$s = "foo";
var_dump(--$s;); // string(3) "foo"
$e = "";
var_dump(--$e); // int(-1)
```

Increment:

```php
$s = "foo";
var_dump(++$s;); // string(3) "fop"
$e = "";
var_dump(++$e); // string(1) "1"
```

PERL string increment

```
$s = "é";
var_dump(++$s); // string(2) "é"
```

PERL string increment

```
$s = "é";
var_dump(++$s); // string(2) "é"

$s = "Z";
var_dump(++$s); // string(2) "AA"

// Trailing whitespace
$s = "Z ";
var_dump(++$s); // string(2) "Z "

// Leading whitespace
$s = " Z";
var_dump(++$s); // string(2) " A"
```

PERL string increment

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

PERL string increment

```
$s = "4y6";
for ($i = 1; $i < 100; $i++) {
    $s++;
}
var_dump($s); // float(50)
```

Episode 2 'Hesitation Flute', *Sound! Euphonium*, Season 2, [television program, BluRay]
Dir. Taichi Ishidate. Kyoto Animation, Japan, 13/10/2016, Tokyo MX. 23mins 41secs. 00:14:26.

PERL string incremented to float explanation

```
$s = "5d9";
var_dump(++$s); // string(3) "5e0"
var_dump(++$s); // float(6)
```

Array offsets

```php
$a = [];
$a[5] = "Fifth key";
$a["string key"] = "Usual string key";
$a["2"] = "Integer string key";
$a["007"] = "Numeric string key";
$a[""] = "Empty key";

var_dump($a);
```

Array offsets

```
array(5) {
  [5]=>
  string(9) "Fifth key"
  ["string key"]=>
  string(16) "Usual string key"
  [2]=>
  string(18) "Integer string key"
  ["007"]=>
  string(18) "Numeric string key"
  [""]=>
  string(9) "Empty key"
}
```

Array offset type juggling

```php
$a = [];
$a[null] = "null";
$a[false] = false;
$a[true] = true;

var_dump($a);
```

Array offset type juggling

```
array(3) {
  [""]=>
  string(4) "null"
  [0]=>
  bool(false)
  [1]=>
  bool(true)
}
```

Array offset type juggling

```php
$a = [];
$a[15.5] = 15.5;
$a["15.5"] = "15.5";

var_dump($a);
```

Array offset type juggling

```
Deprecated: Implicit conversion from float 15.5 to int loses
↪  precision
array(2) {
  [15]=>
  float(15.5)
  ["15.5"]=>
  string(4) "15.5"
}
```

Array offset type juggling

```php
$a[STDERR] = "Resource";
var_dump($a);
// Warning: Resource ID#3 used as offset, casting to integer
↪  (3)
array(1) {
  [3]=>
  string(8) "Resource"
}

$a[[]] = "Array";
// TypeError: Illegal offset type
```

Array offset type juggling

```php
$o = new stdClass();
$a[$o] = "Object";
// TypeError: Illegal offset type

class Stringy {
    public function __toString() { return "foo"; }
}
$stringable = new Stringy();
$a[$stringable] = "Stringable";
// TypeError: Illegal offset type
```

Unset Array Offsets

```
unset($a[null]);
unset($a[false]);
unset($a[true]);
unset($a[15.5]);
Deprecated: Implicit conversion from float 15.5 to int loses
↪  precision
unset($a[STDERR]);
Warning: Resource ID#3 used as offset, casting to integer
↪  (3)
```

Unset Array Offsets

```php
unset($a[[]]);
// TypeError: Illegal offset type

unset($a[$o]);
// TypeError: Illegal offset type
```

`empty`/`isset` Array Offsets

```php
var_dump(isset($a[null]));   // bool(true)
var_dump(isset($a[false]));  // bool(true)
var_dump(isset($a[true]));   // bool(true)
var_dump(isset($a[15.5]));   // bool(true)
// Deprecated: Implicit conversion from float 15.5 to int
↪  loses precision
var_dump(isset($a[STDERR])); // bool(true)
// Warning: Resource ID#3 used as offset, casting to integer
↪  (3)
```

empty/isset Array Offsets

```php
isset($a[[]]);
// TypeError: Illegal offset type

isset($a[$o]);
// TypeError: Illegal offset type
```

`empty`/`isset` Array Offsets

```php
var_dump($a[null]  ?? 'default');  // string(4) "null"
var_dump($a[false] ?? 'default');  // bool(false)
var_dump($a[true]  ?? 'default');  // bool(true)
var_dump($a[15.5]  ?? 'default');  // float(15.5)
// Deprecated: Implicit conversion from float 15.5 to int
↪  loses precision
var_dump($a[STDERR] ?? 'default'); // string(8) "Resource"
// Warning: Resource ID#3 used as offset, casting to integer
↪  (3)
```

`empty`/`isset` Array Offsets

```php
var_dump($a[[]] ?? 'default');
// TypeError: Illegal offset type

var_dump($a[$o] ?? 'default');
// TypeError: Illegal offset type
```

String offsets

```php
$s = "abcdefghijklmnopqrstuvwxyz";
var_dump($s[6]);     // string(1) "g"
```

String offsets

```php
$s = "abcdefghijklmnopqrstuvwxyz";
var_dump($s[6]);     // string(1) "g"

var_dump($s[null]);  // string(1) "a"
// Warning: String offset cast occurred

var_dump($s[false]); // string(1) "a"
// Warning: String offset cast occurred

var_dump($s[true]);  // string(1) "b"
// Warning: String offset cast occurred
```

String offsets

```php
var_dump($s[8.0]); // string(1) "i"
// Warning: String offset cast occurred

var_dump($s[8.6]); // string(1) "i"
// Warning: String offset cast occurred
```

String offsets

```php
var_dump($s["2"]);    // string(1) "c"
var_dump($s["007"]);  // string(1) "h"
var_dump($s["19.0"]);
// TypeError: Cannot access offset of type string on string
var_dump($s["19.6"]);
// TypeError: Cannot access offset of type string on string
var_dump($s["key"]);
// TypeError: Cannot access offset of type string on string
```

String offsets

```php
var_dump($s[[]]);
// TypeError: Cannot access offset of type array on string
var_dump($s[STDERR]);
// TypeError: Cannot access offset of type resource on
↪  string
$o = new stdClass();
var_dump($s[$o]);
// TypeError: Cannot access offset of type stdClass on
↪  string
```

Unset String Offsets

```php
unset($s[6]);
// Error: Cannot unset string offsets
```

Unset String Offsets

```php
unset($s[6]);
// Error: Cannot unset string offsets

unset($s["a"]["b"]);
// TypeError: Cannot access offset of type string on string
```

`empty`/`isset` String Offsets

```php
var_dump(isset($s[6]));     // bool(true)
var_dump(isset($s[null]));  // bool(true)
var_dump(isset($s[false])); // bool(true)
var_dump(isset($s[true]));  // bool(true)
var_dump(isset($s[8.0]));   // bool(true)
var_dump(isset($s[8.6]));   // bool(true)
// Deprecated: Implicit conversion from float 8.6 to int
↪  loses precision
```

`empty`/`isset` String Offsets

```php
var_dump(isset($s["2"]));    // bool(true)
var_dump(isset($s["007"]));  // bool(true)
var_dump(isset($s["19.0"])); // bool(false)
var_dump(isset($s["19.6"])); // bool(false)
var_dump(isset($s["key"]));  // bool(false)
var_dump(isset($s[[]]));     // bool(false)
var_dump(isset($s[STDERR])); // bool(false)
var_dump(isset($s[$o]));     // bool(false)
```

`empty`/`isset` String Offsets

```php
var_dump($s[6] ?? 'default');     // string(1) "g"
var_dump($s[null] ?? 'default');  // string(1) "a"
var_dump($s[false] ?? 'default'); // string(1) "a"
var_dump($s[true] ?? 'default');  // string(1) "b"
var_dump($s[8.0] ?? 'default');   // string(1) "i"
var_dump($s[8.6] ?? 'default');   // string(1) "i"
```

`empty`/`isset` String Offsets

```php
var_dump($s["2"] ?? 'default');    // string(1) "c"
var_dump($s["007"] ?? 'default');  // string(1) "h"
var_dump($s["19.0"] ?? 'default'); // string(7) "default"
var_dump($s["19.6"] ?? 'default'); // string(7) "default"
var_dump($s["key"] ?? 'default');  // string(7) "default"
```

`empty`/`isset` String Offsets

```php
var_dump($s[STDERR] ?? 'default');
// TypeError: Cannot access offset of type resource on
↪  string
var_dump($s[[]] ?? 'default');
// TypeError: Cannot access offset of type array on string
var_dump($s[$o] ?? 'default');
// TypeError: Cannot access offset of type object on string
```

Episode 13 'Early-Spring Epilogue', *Sound! Euphonium*, Season 2, [television program, BluRay]
Dirs. Tatsuya Ishihara, et al. Kyoto Animation, Japan, 28/12/2016, Tokyo MX. 23mins 41secs. 00:02:56.

exit construct

exit accepts either an **int** or **string**

```
exit(10);
// Process exited with code 10.

exit("Programm exit");
// Programm exit
```

`exit` construct

If the value is not an **`int`** then ***`exit`*** will interpret it as **`string`**

```
exit(null);
//

exit(false);
//

exit(true);
// 1

exit(15.0);
// 15
```

`exit` construct

If the value is not an **`int`** then ***`exit`*** will interpret it as **`string`**

```php
exit([]);
// Warning: Array to string conversion in %s on line %d
// Array

$r = STDERR;
exit($r);
// Resource id #3
```

If the value is not an **int** then *exit* will interpret it as **string**

```php
$o = new stdClass;
try {
    exit($o);
} catch (\Error $e) {
    echo $e::class, PHP_EOL, "I've escaped the exit";
}
// Error
// I've escaped the exit
```

Which type juggling contexts does `strict_types` affect?

1. String
2. Integral and String
3. Numeric
4. Logical
5. Comparative
6. Function
7. Increment/Decrement operators
8. Array offsets
9. String offsets
10. *`exit`* construct

Which type juggling contexts does `strict_types` affect?

1. String
2. Integral and String
3. Numeric
4. Logical
5. Comparative
6. Function
7. Increment/Decrement operators
8. Array offsets
9. String offsets
10. *`exit`* construct

What does `strict_types` do?

Only enabled in PHP scripts that use *`declare(strict_types=1);`*

Disables coercion of scalar types in the following cases only:

- Arguments passed to function calls made in userland
- Return value for user defined functions
- Value assignment to a typed property

strict_types

strict_types does not change the behaviour of:

strict_types

strict_types does not change the behaviour of:

- Binary operations:

```php
declare(strict_types=1);
var_dump(10 + "45"); // int(55)
var_dump(true . " hello"); // string(7) "1 hello"
```

strict_types

strict_types does not change the behaviour of:

- Comparison operations:

```php
declare(strict_types=1);
var_dump("1" == "01"); // bool(true)
var_dump(014 == "14"); // bool(false)
var_dump(14 == "014"); // bool(true)
```

strict_types

strict_types does not change the behaviour of:

- *exit* construct:

```php
declare(strict_types=1);
exit(true);
// 1
```

strict_types

strict_types does not change the behaviour of:

- Scalar type coercion for functions called by the engine:

```php
declare(strict_types=1);
$f = fn (int $i): bool => (bool) ($i % 2);
$a = ['1', '2', 3, 4, '5.0', '6.0'];
var_dump(array_filter($a, $f));
/* array(3) {
  [0]=>
  string(1) "1"
  [2]=>
  int(3)
  [4]=>
  string(3) "5.0"
} */
```

Hot take

The `strict_types` declare was a mistake.

Thank you!

GitHub: Girgias

Site: `https://gpb.moe`

Mastodon: @Girgias@phpc.social

Feedback:

References

[1] George Peter Banyard. *Add true type*. English. Apr. 2022. URL: *https://wiki.php.net/rfc/true-type* (visited on 10/10/2022).

[2] George Peter Banyard. *Allow null and false as stand-alone types*. English. Feb. 2022. URL: *https://wiki.php.net/rfc/null-false-standalone-types* (visited on 10/10/2022).

[3] George Peter Banyard. *Define proper semantics for range() function*. Mar. 2023. URL: https://wiki.php.net/rfc/proper-range-semantics (visited on 05/11/2023).

[4] George Peter Banyard. *Disjunctive Normal Form Types*. English. Apr. 2022. URL: https://wiki.php.net/rfc/dnf_types (visited on 10/10/2022).

[5] George Peter Banyard. *Path to Saner Increment/Decrement operators*. English. Nov. 2022. URL: https://wiki.php.net/rfc/saner-inc-dec-operators (visited on 02/15/2023).

[6] George Peter Banyard. *Pure intersection types*. English. Mar. 2021. URL: https://wiki.php.net/rfc/pure-intersection-types (visited on 10/10/2022).

[7] George Peter Banyard. *Saner array_(sum|product)()*. English. Jan. 2023. URL: https://wiki.php.net/rfc/saner-array-sum-product (visited on 02/15/2023).

[8] George Peter Banyard. *Saner numeric strings*. English. June 2020. URL: https://wiki.php.net/rfc/saner-numeric-strings (visited on 10/10/2022).

[9] George Peter Banyard and Máté Kocsis. *Locale-independent float to string cast*. English. Mar. 2020. URL: https://wiki.php.net/rfc/locale_independent_float_to_string (visited on 02/19/2023).

[10] *Disjunctive normal form*. en. Page Version ID: 1097076215. July 2022. URL: https://en.wikipedia.org/w/index.php?title=Disjunctive_normal_form&oldid=1097076215 (visited on 10/24/2022).

[11] Barbara H. Liskov and Jeannette M. Wing. "A behavioral notion of subtyping". In: *ACM Transactions on Programming Languages and Systems* 16.6 (Nov. 1994), pp. 1811–1841. ISSN: 0164-0925. DOI: 10.1145/197320.197383. URL: https://doi.org/10.1145/197320.197383 (visited on 10/10/2022).

[12] Mark Seemann. *The Liskov Substitution Principle as a profunctor*. English. Blog. Dec. 2021. URL: *https://blog.ploeh.dk/2021/12/06/the-liskov-substitution-principle-as-a-profunctor/* (visited on 10/10/2022).

[13] *Subtyping*. en. Page Version ID: 1127276117. Dec. 2022. URL: *https://en.wikipedia.org/w/index.php?title=Subtyping&oldid=1127276117* (visited on 02/15/2023).

[14] *Type system*. en. Page Version ID: 1114430701. Oct. 2022. URL: *https://en.wikipedia.org/w/index.php?title=Type_system&oldid=1114430701* (visited on 10/10/2022).