OSC源创会 年终盛典 2016

PHP7.1 New Features & Performance

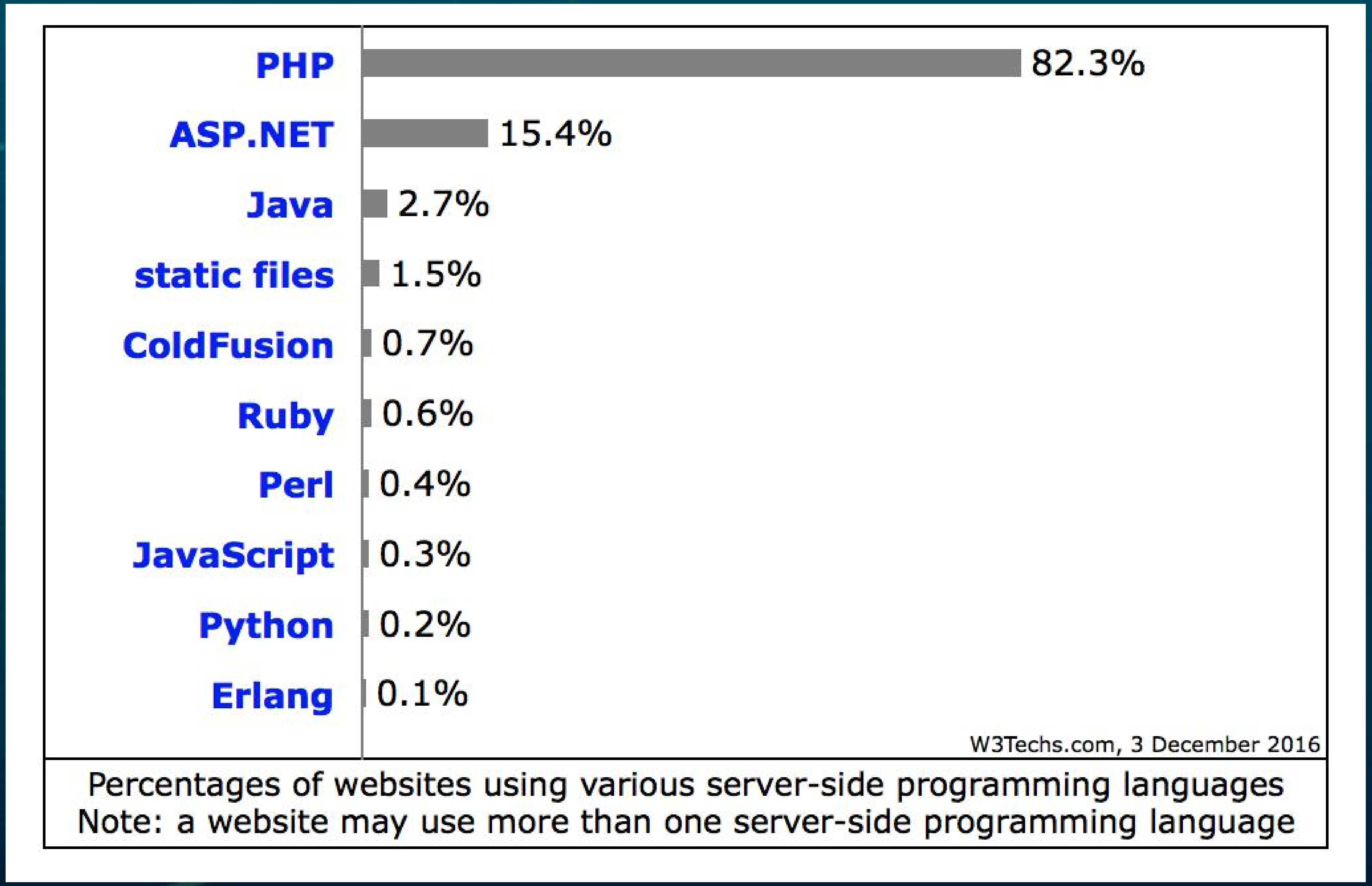
@Laruence

About Me

- Author of Yaf, Yar, Yac, Yaconf, Taint, Lua, etc
- Maintainer of APC, Zend Opcache, Msgpack
- PHP core developer since 2011
- Zend consultant since 2013
- Core author of PHP7
- Chief Architect at Lianjia

PHP

- 20 years history
- Most popular Web service program language
- Over 82% sites are use PHP as server side program language
- Latest version is PHP-7.1



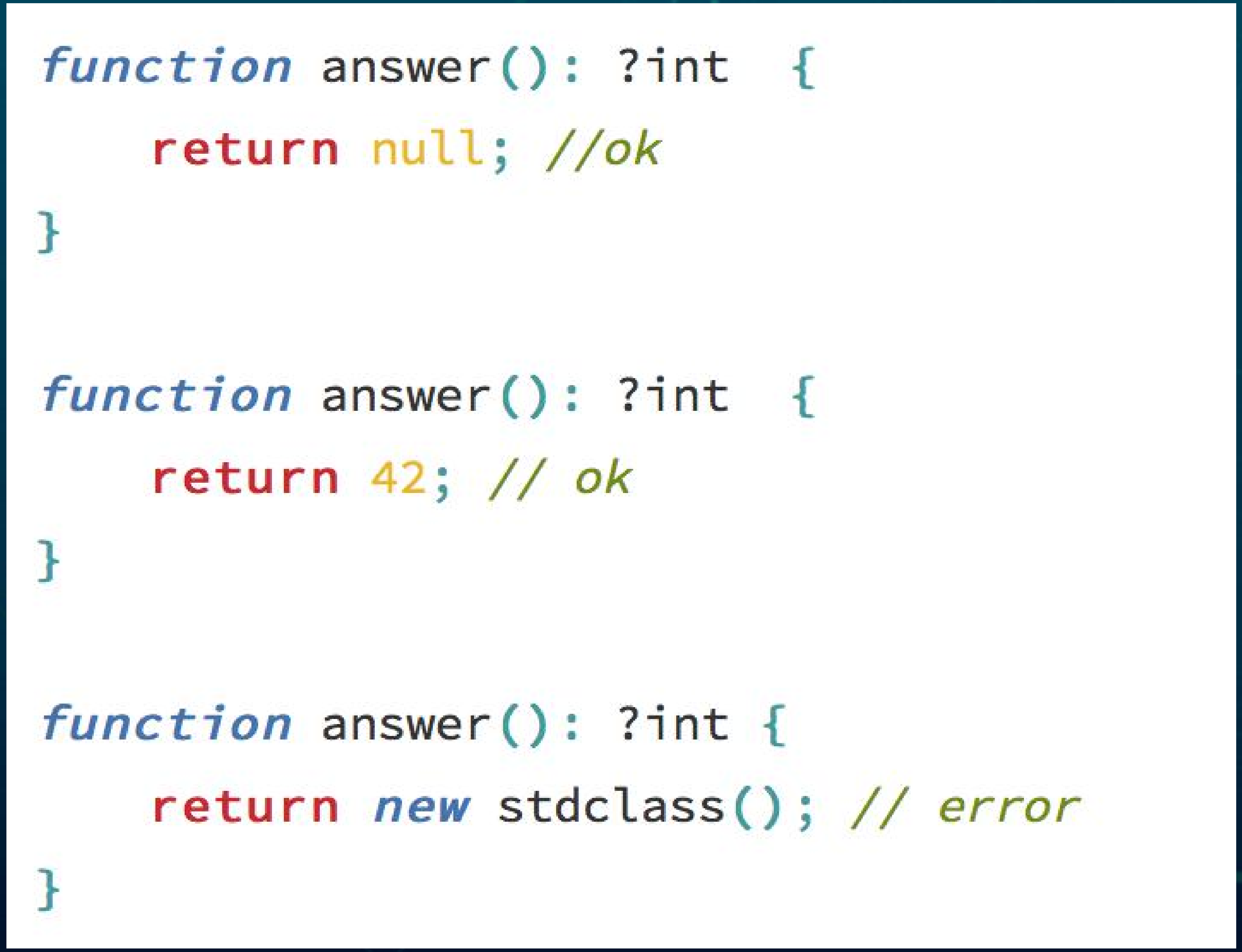
Percentages of websites using various server-side programming languages
Note: a website may use more than one server-side programming language

PHP71

- Nullable types
- Void return type
- Iterable pseudo-type
- Class constant visibility modifiers
- Specify keys in list()
- Square bracket syntax for list()
- Catching multiple exception types
- Missing arguments Excetpion
- Warn about invalid strings in arithmetic
- Generalize support of negative string offsets
- ….

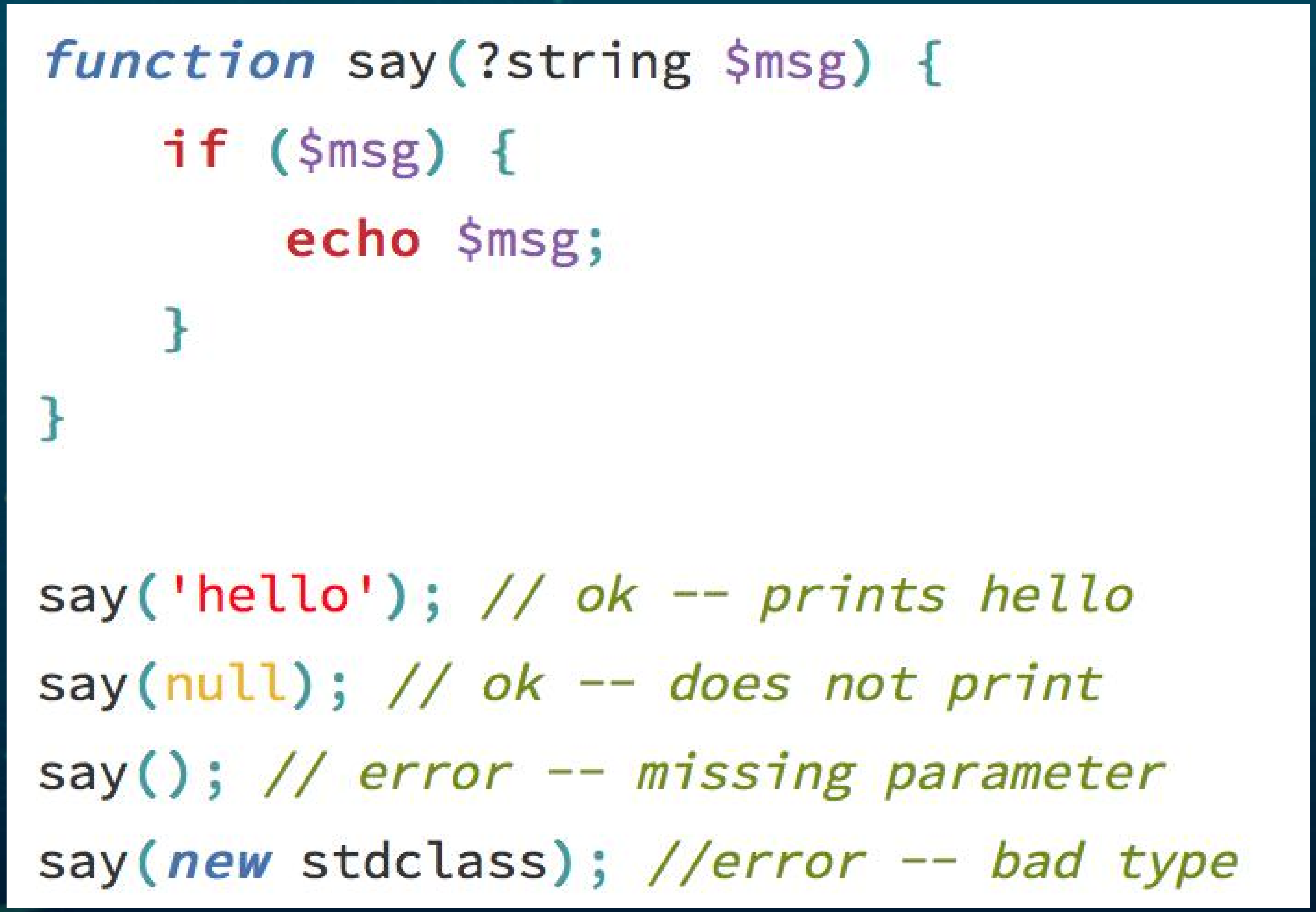
Nullable types

- An enhancement for typehints



```php
function answer(): ?int  {
    return null; //ok
}

function answer(): ?int  {
    return 42; // ok
}

function answer(): ?int {
    return new stdclass(); // error
}
```

return type

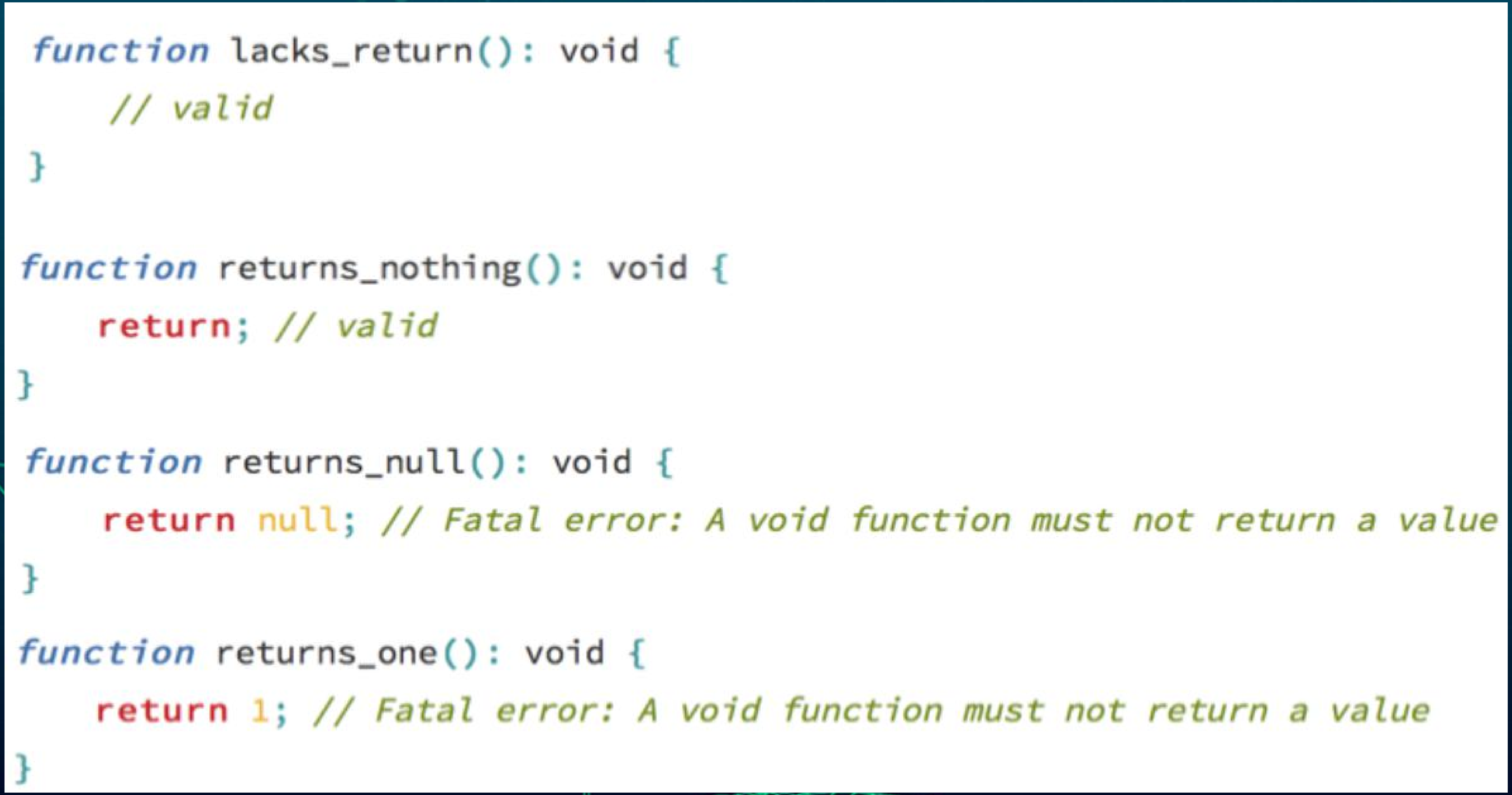


```php
function say(?string $msg) {
    if ($msg) {
        echo $msg;
    }
}

say('hello'); // ok -- prints hello
say(null); // ok -- does not print
say(); // error -- missing parameter
say(new stdclass); //error -- bad type
```

parameters

Void return types

- An enhancement for return type hint



```
function lacks_return(): void {
    // valid
}

function returns_nothing(): void {
    return; // valid
}

function returns_null(): void {
    return null; // Fatal error: A void function must not return a value
}

function returns_one(): void {
    return 1; // Fatal error: A void function must not return a value
}
```

Iterable Pseudo type

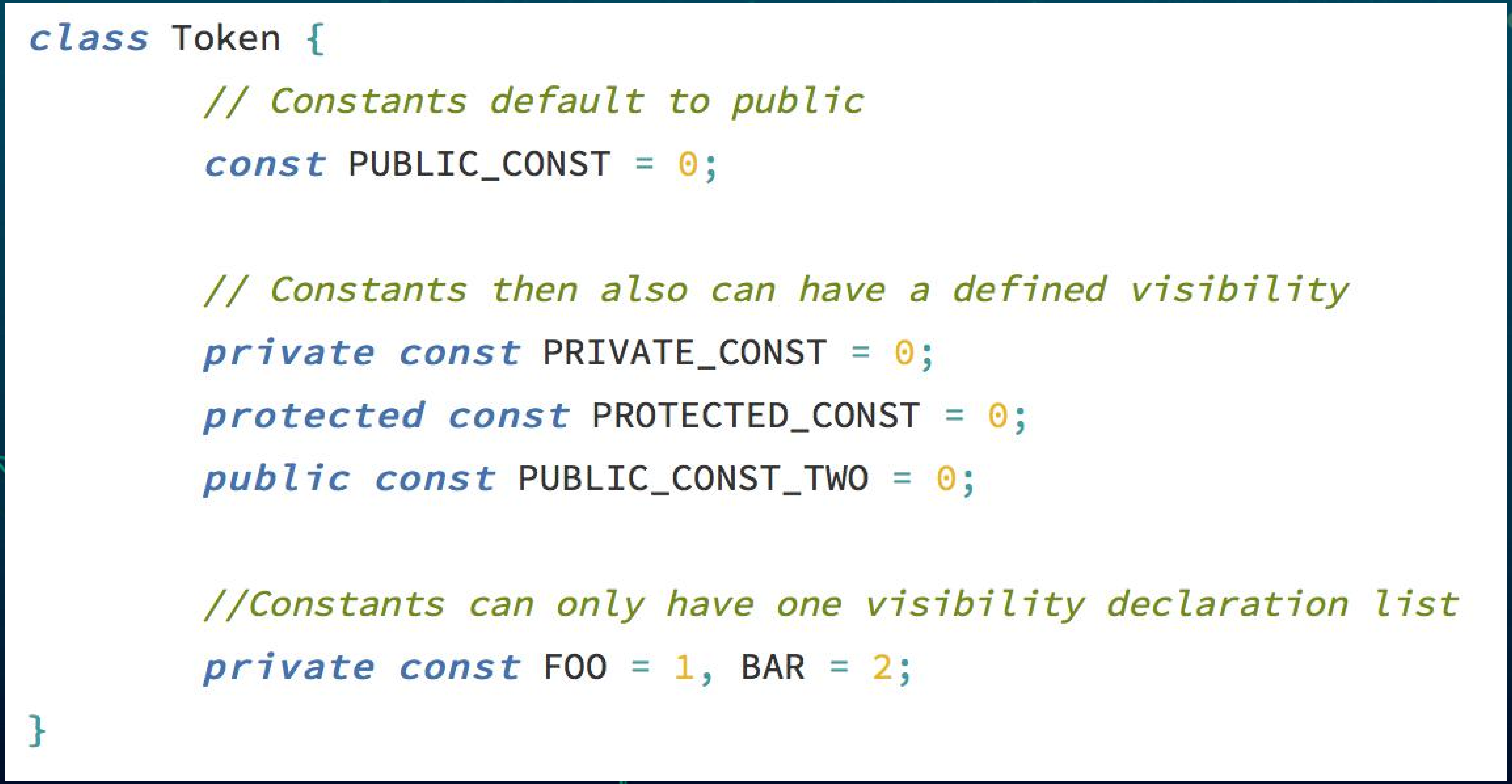
- Iterable accepts any array or object implementing Traversable
- Iterable can also be used in return type

```
function foo(iterable $iterable) {
    foreach ($iterable as $value) {
        // ...
    }
}
```

```
function bar(): iterable {
    return [1, 2, 3];
}
```

Class constant visibility modifiers

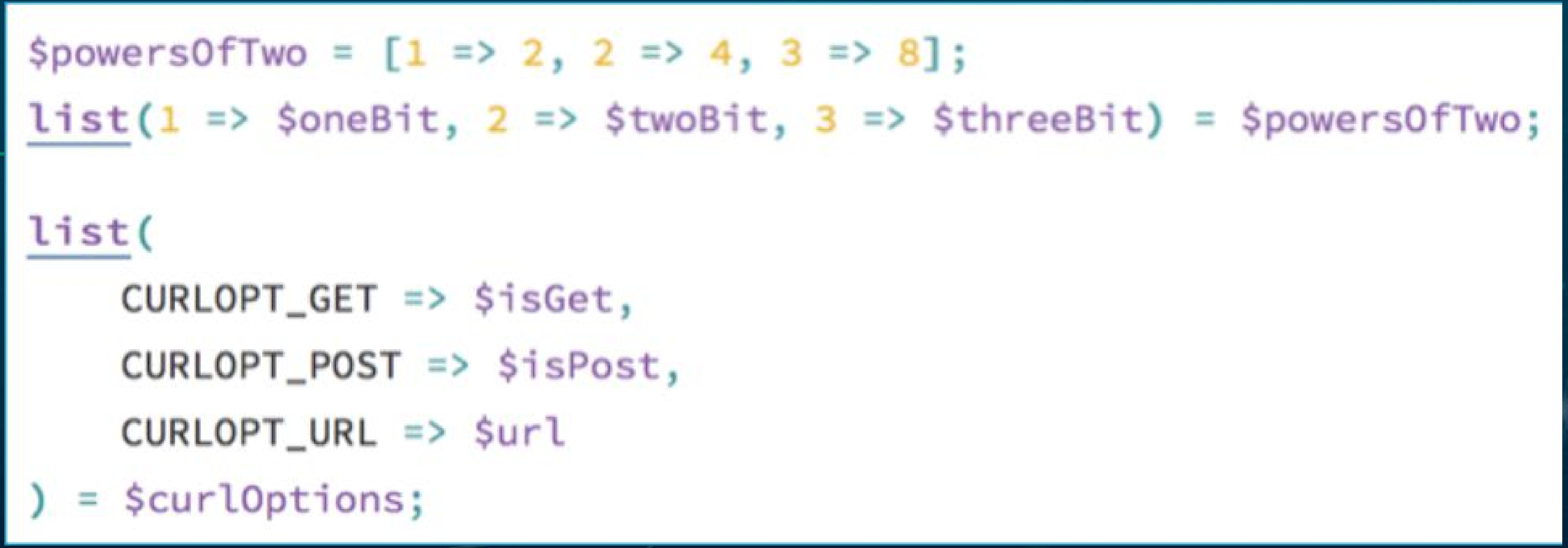
- Support class constant visibility



```php
class Token {
        // Constants default to public
        const PUBLIC_CONST = 0;

        // Constants then also can have a defined visibility
        private const PRIVATE_CONST = 0;
        protected const PROTECTED_CONST = 0;
        public const PUBLIC_CONST_TWO = 0;

        //Constants can only have one visibility declaration list
        private const FOO = 1, BAR = 2;
}
```

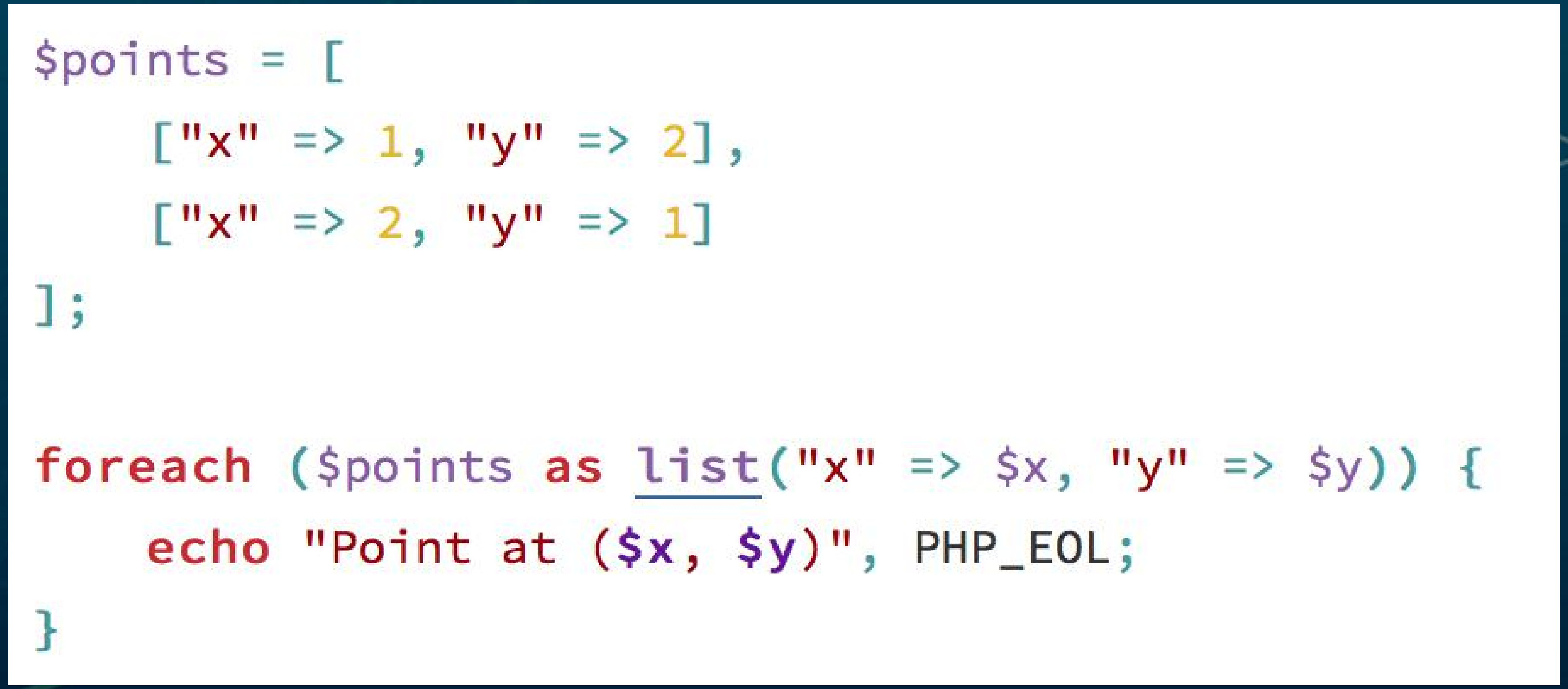
Specify keys in list()

- An enhancement for list()
- Also works in foreach



```php
$powersOfTwo = [1 => 2, 2 => 4, 3 => 8];
list(1 => $oneBit, 2 => $twoBit, 3 => $threeBit) = $powersOfTwo;

list(
    CURLOPT_GET => $isGet,
    CURLOPT_POST => $isPost,
    CURLOPT_URL => $url
) = $curlOptions;
```

specify keys in list

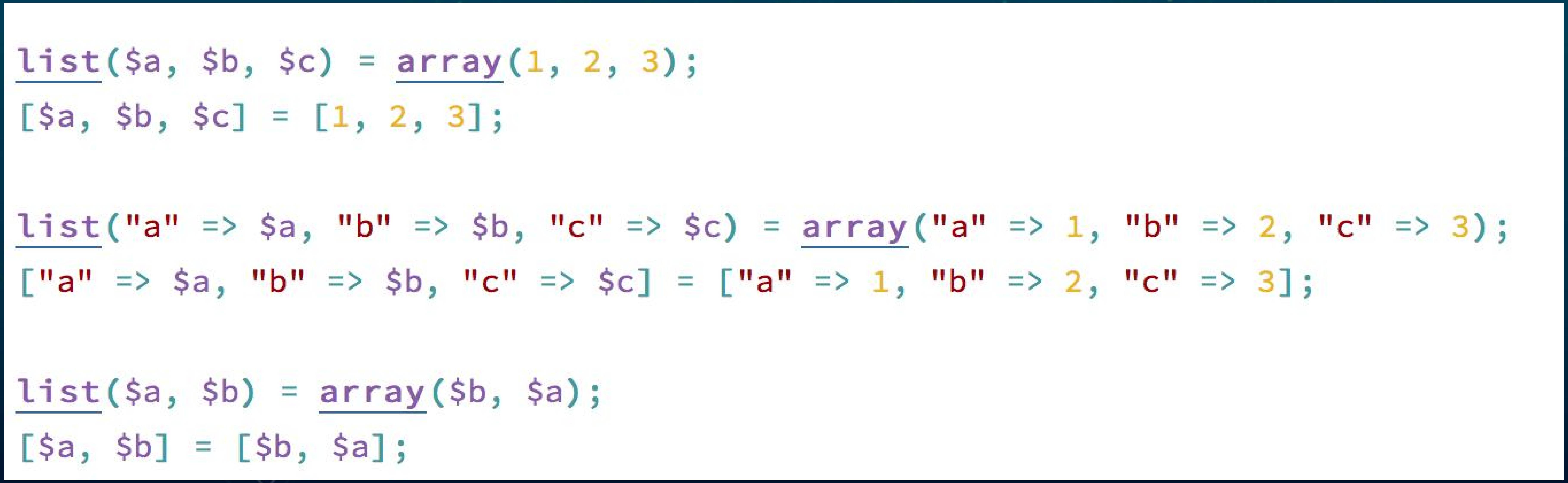


```php
$points = [
    ["x" => 1, "y" => 2],
    ["x" => 2, "y" => 1]
];

foreach ($points as list("x" => $x, "y" => $y)) {
    echo "Point at ($x, $y)", PHP_EOL;
}
```

specify keys in list with foreach

Square backtrace for list()

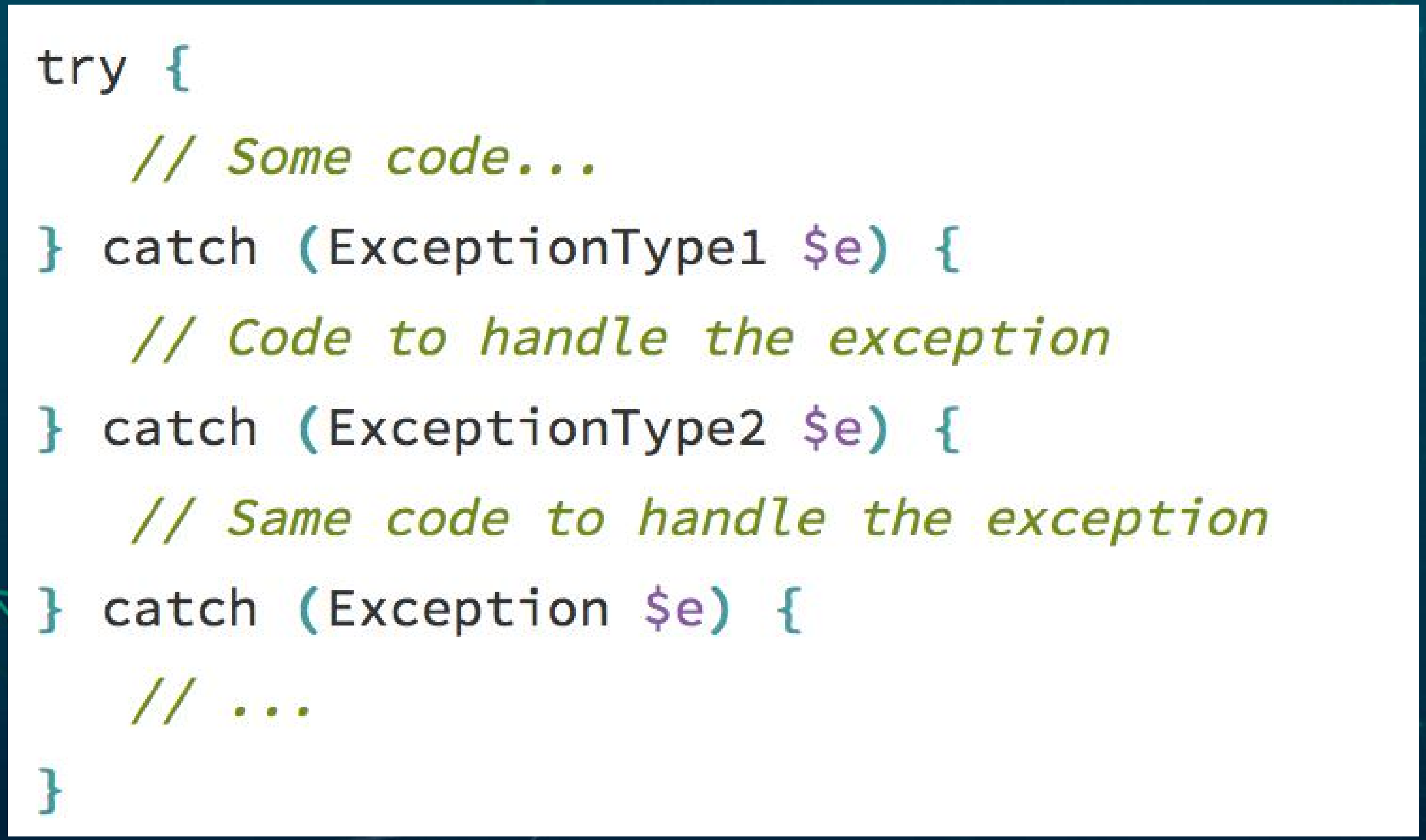
- Continuation syntax for short array syntax introduced in 5.4
- Also works in foreach too



```php
list($a, $b, $c) = array(1, 2, 3);
[$a, $b, $c] = [1, 2, 3];

list("a" => $a, "b" => $b, "c" => $c) = array("a" => 1, "b" => 2, "c" => 3);
["a" => $a, "b" => $b, "c" => $c] = ["a" => 1, "b" => 2, "c" => 3];

list($a, $b) = array($b, $a);
[$a, $b] = [$b, $a];
```

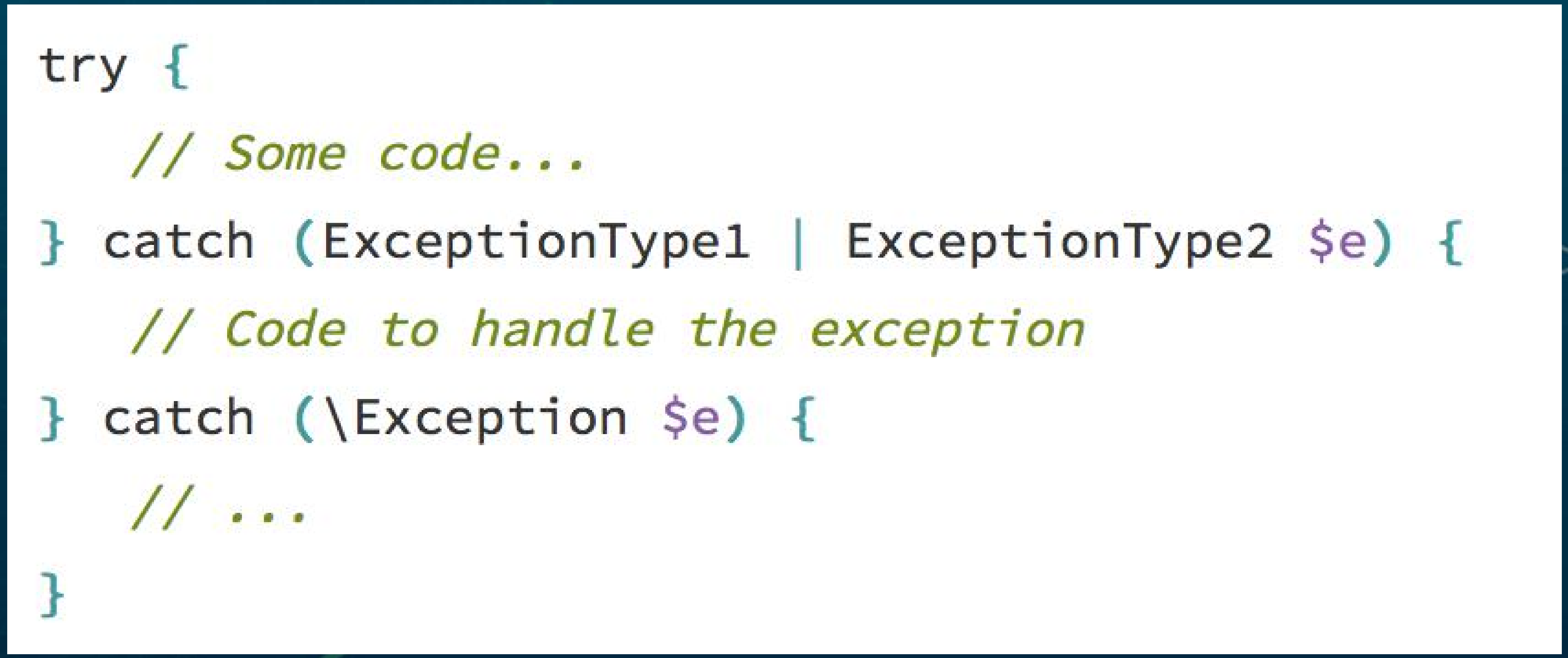
Catching multiply exception types

- Allow catching multiply exception types in single catch



```
try {
    // Some code...
} catch (ExceptionType1 $e) {
    // Code to handle the exception
} catch (ExceptionType2 $e) {
    // Same code to handle the exception
} catch (Exception $e) {
    // ...
}
```

normal way



```
try {
    // Some code...
} catch (ExceptionType1 | ExceptionType2 $e) {
    // Code to handle the exception
} catch (\Exception $e) {
    // ...
}
```

PHP71 way

Missing Arguments Exception

- Disable calling “user” functions with insufficient actual parameters

```php
function foo($a) {
   var_dump($a);   // NULL + Warning: Undefined variable: a
   var_dump($a);   // NULL + Warning: Undefined variable: a
}
foo();            // Warning: Missing argument 1 for foo()
```

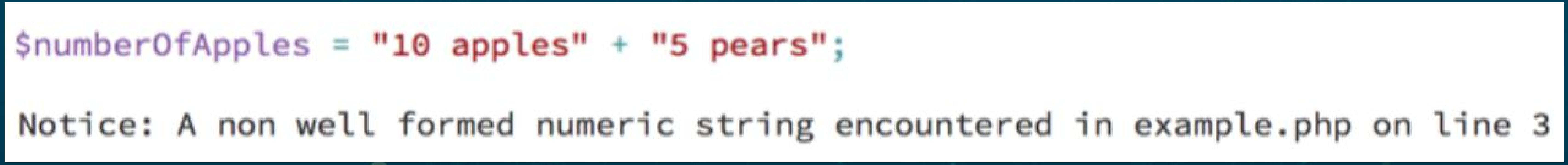
before 71

```php
function foo($a) {
   var_dump($a);   // not executed
   var_dump($a);   // not executed
}
foo();            // throw Error("Too few arguments to function foo(), 0 passed in %s on line
```

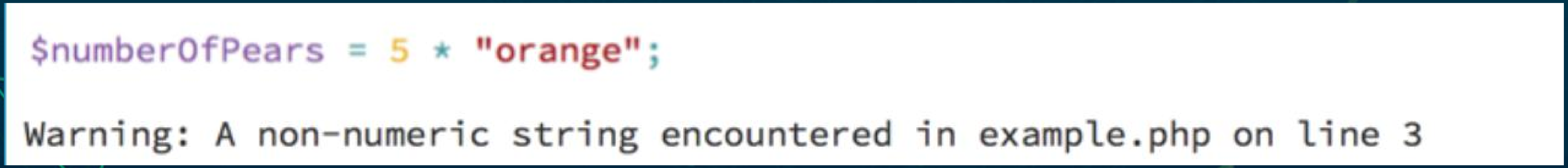
after 71

Warn about invalid strings in arithmetic

- Produce E_NOTICE or E_WARNING when using invalid numeric strings with arithmetic operators



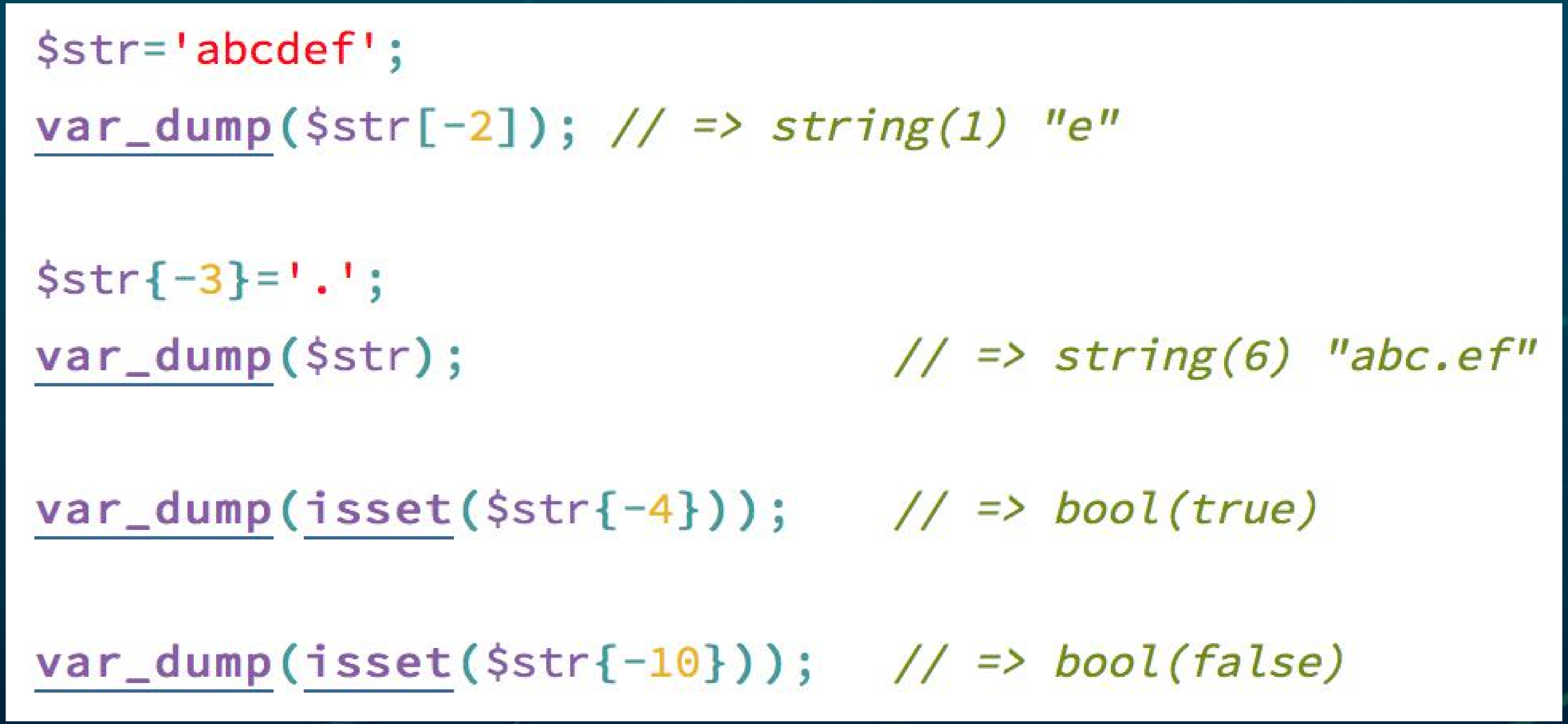
```
$numberOfApples = "10 apples" + "5 pears";

Notice: A non well formed numeric string encountered in example.php on line 3
```



```
$numberOfPears = 5 * "orange";

Warning: A non-numeric string encountered in example.php on line 3
```

Generalize support of negative string offsets

- Support of negative string offsets when it make sense



```php
$str='abcdef';
var_dump($str[-2]); // => string(1) "e"

$str{-3}='.';
var_dump($str);                 // => string(6) "abc.ef"

var_dump(isset($str{-4}));      // => bool(true)

var_dump(isset($str{-10}));     // => bool(false)
```

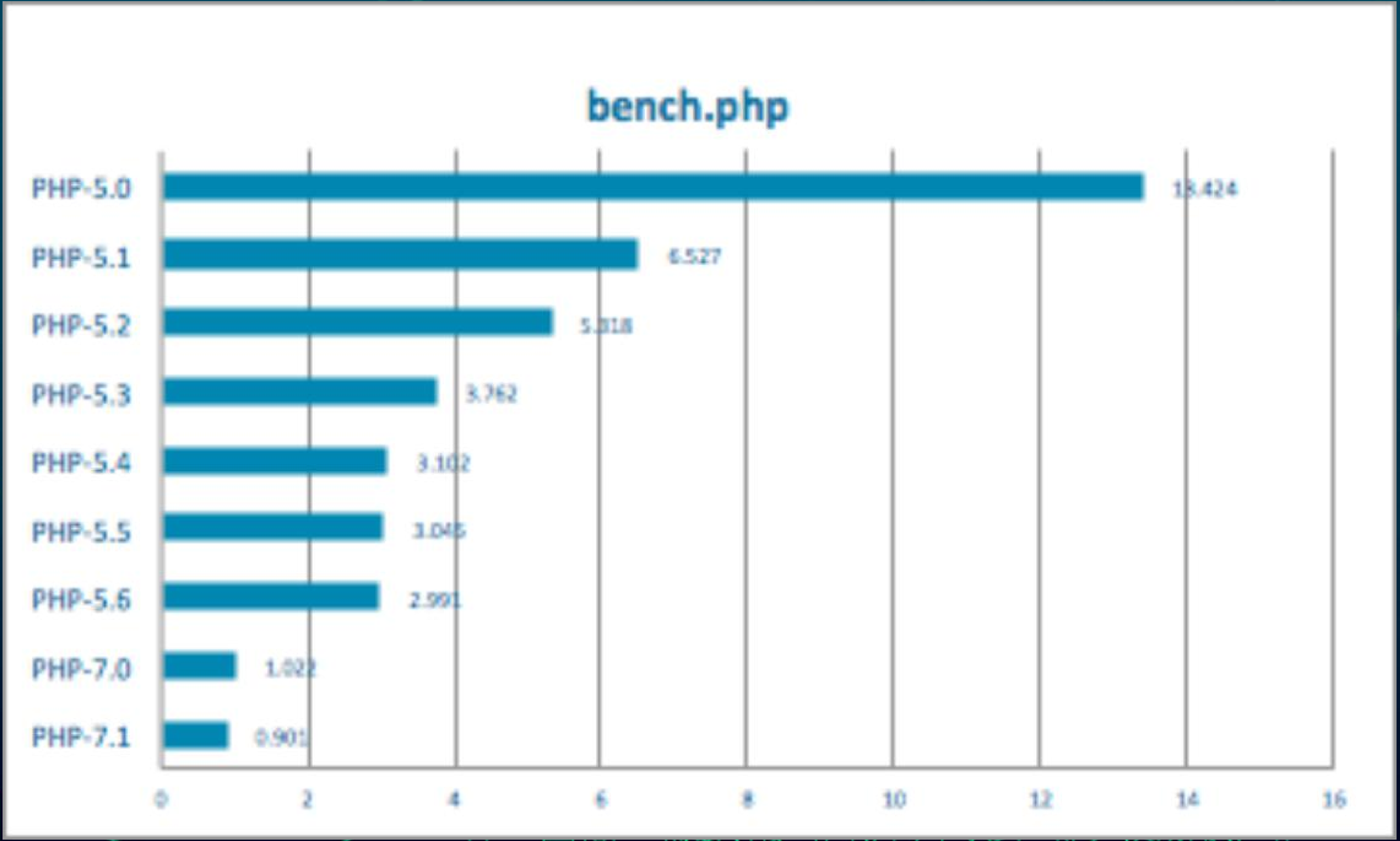
strpos
stripos
substr_count
grapheme_strpos
grapheme_stripos
grapheme_extract
iconv_strpos
file_get_contents
mb_strimwidth
mb_ereg_search_setpos
mb_strpos
mb_stripos

OSC源创会 年终盛典 2016

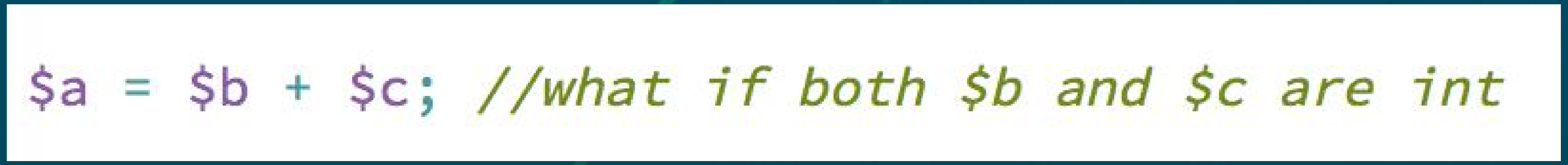
One More thing

PHP71 Performance

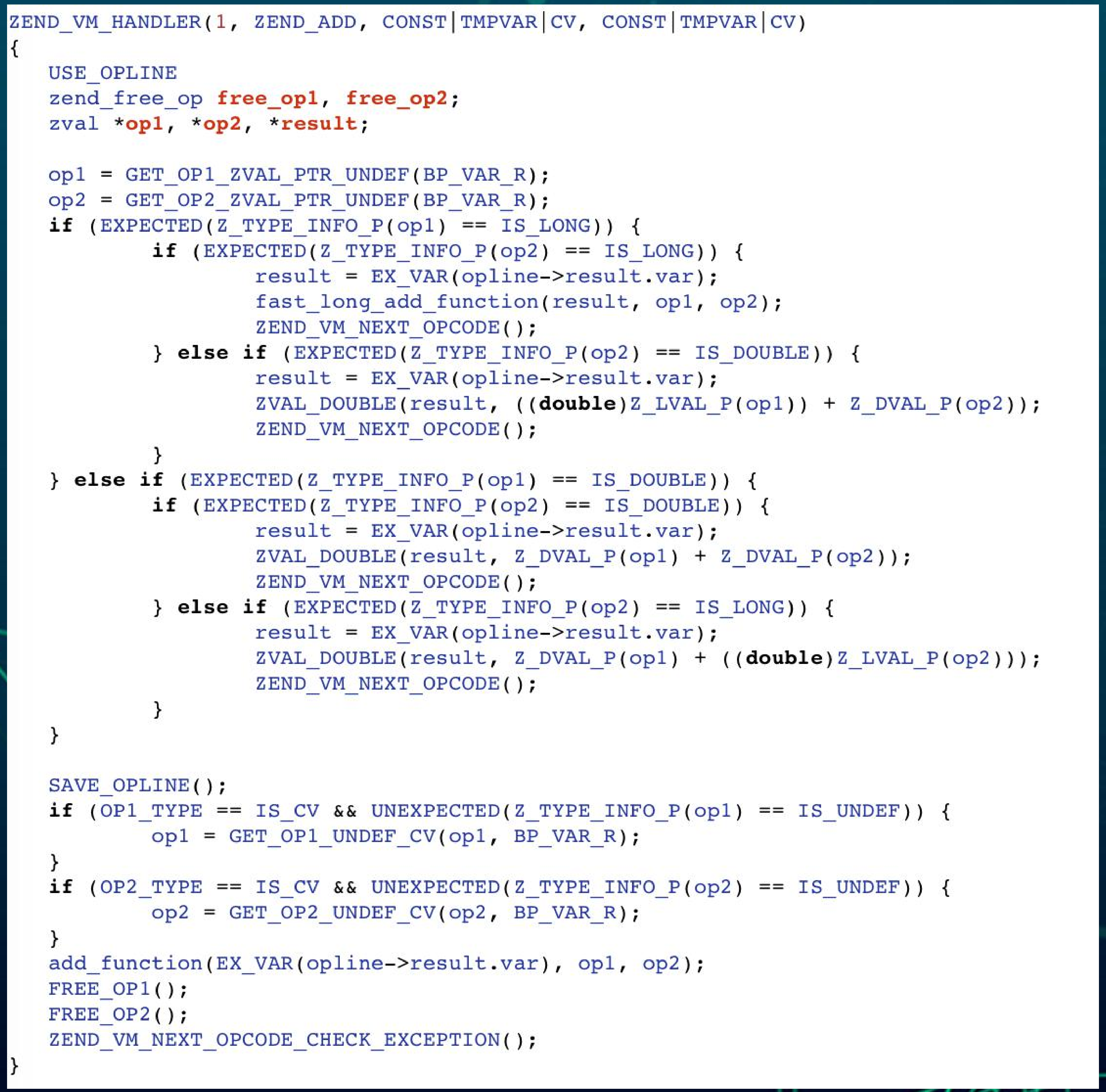
- Over 10% Perfromance improvement



Type specific opcode handlers

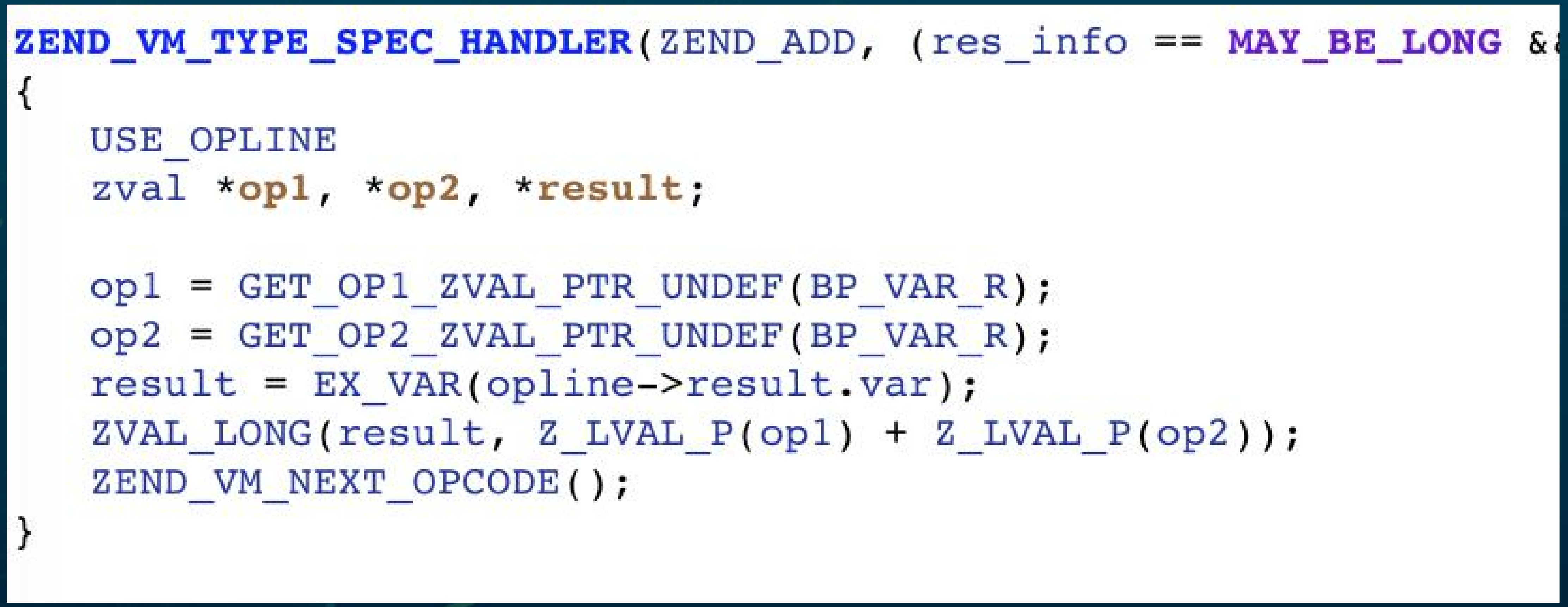


```php
$a = $b + $c; //what if both $b and $c are int
```



```c
ZEND_VM_HANDLER(1, ZEND_ADD, CONST|TMPVAR|CV, CONST|TMPVAR|CV)
{
   USE_OPLINE
   zend_free_op free_op1, free_op2;
   zval *op1, *op2, *result;

   op1 = GET_OP1_ZVAL_PTR_UNDEF(BP_VAR_R);
   op2 = GET_OP2_ZVAL_PTR_UNDEF(BP_VAR_R);
   if (EXPECTED(Z_TYPE_INFO_P(op1) == IS_LONG)) {
           if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_LONG)) {
                   result = EX_VAR(opline->result.var);
                   fast_long_add_function(result, op1, op2);
                   ZEND_VM_NEXT_OPCODE();
           } else if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_DOUBLE)) {
                   result = EX_VAR(opline->result.var);
                   ZVAL_DOUBLE(result, ((double)Z_LVAL_P(op1)) + Z_DVAL_P(op2));
                   ZEND_VM_NEXT_OPCODE();
           }
   } else if (EXPECTED(Z_TYPE_INFO_P(op1) == IS_DOUBLE)) {
           if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_DOUBLE)) {
                   result = EX_VAR(opline->result.var);
                   ZVAL_DOUBLE(result, Z_DVAL_P(op1) + Z_DVAL_P(op2));
                   ZEND_VM_NEXT_OPCODE();
           } else if (EXPECTED(Z_TYPE_INFO_P(op2) == IS_LONG)) {
                   result = EX_VAR(opline->result.var);
                   ZVAL_DOUBLE(result, Z_DVAL_P(op1) + ((double)Z_LVAL_P(op2)));
                   ZEND_VM_NEXT_OPCODE();
           }
   }

   SAVE_OPLINE();
   if (OP1_TYPE == IS_CV && UNEXPECTED(Z_TYPE_INFO_P(op1) == IS_UNDEF)) {
           op1 = GET_OP1_UNDEF_CV(op1, BP_VAR_R);
   }
   if (OP2_TYPE == IS_CV && UNEXPECTED(Z_TYPE_INFO_P(op2) == IS_UNDEF)) {
           op2 = GET_OP2_UNDEF_CV(op2, BP_VAR_R);
   }
   add_function(EX_VAR(opline->result.var), op1, op2);
   FREE_OP1();
   FREE_OP2();
   ZEND_VM_NEXT_OPCODE_CHECK_EXCEPTION();
}
```

ZEND_ADD

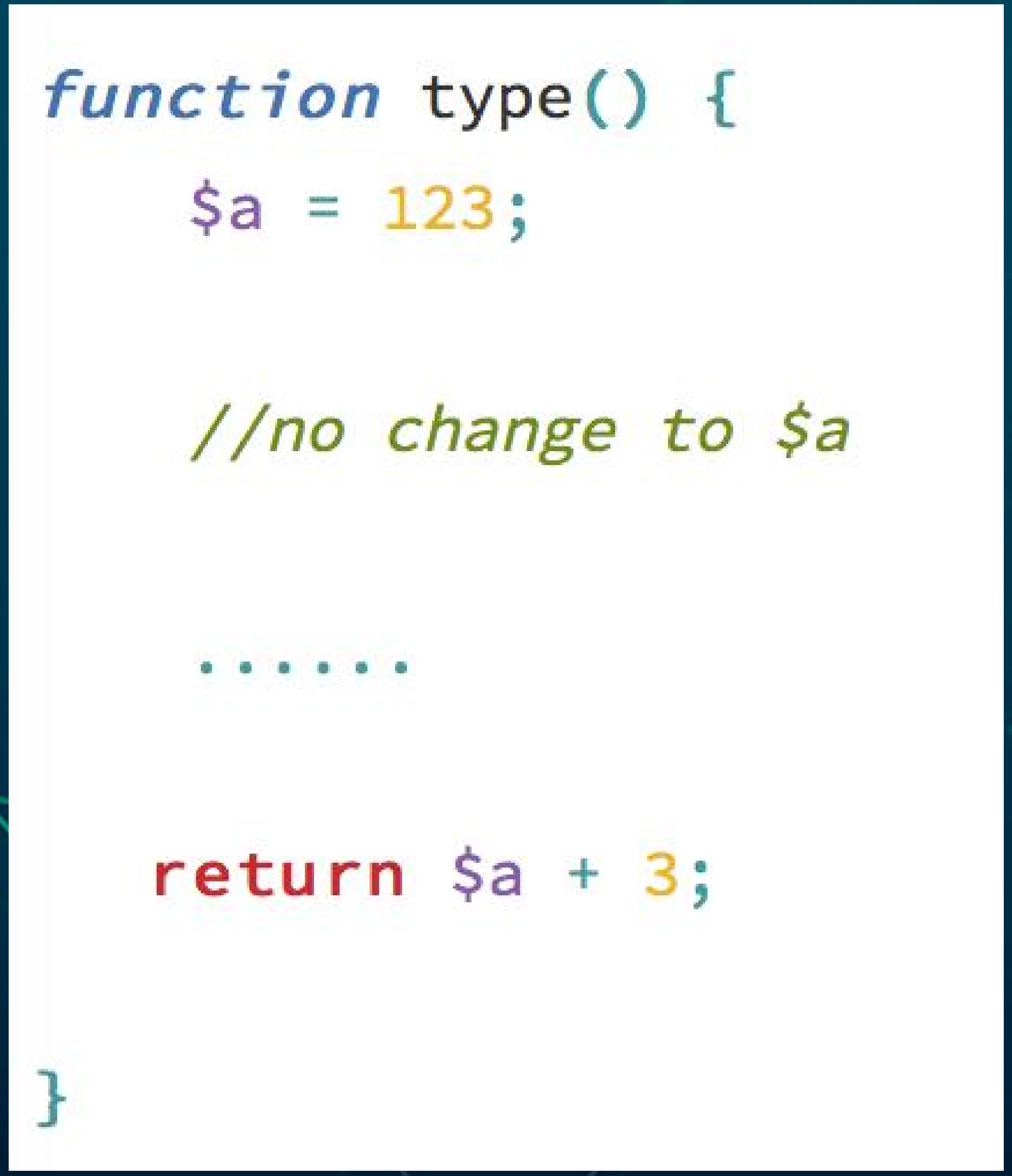


```c
ZEND_VM_TYPE_SPEC_HANDLER(ZEND_ADD, (res_info == MAY_BE_LONG &&
{
   USE_OPLINE
   zval *op1, *op2, *result;

   op1 = GET_OP1_ZVAL_PTR_UNDEF(BP_VAR_R);
   op2 = GET_OP2_ZVAL_PTR_UNDEF(BP_VAR_R);
   result = EX_VAR(opline->result.var);
   ZVAL_LONG(result, Z_LVAL_P(op1) + Z_LVAL_P(op2));
   ZEND_VM_NEXT_OPCODE();
}
```

ZEND_ADD_LONG_LONG

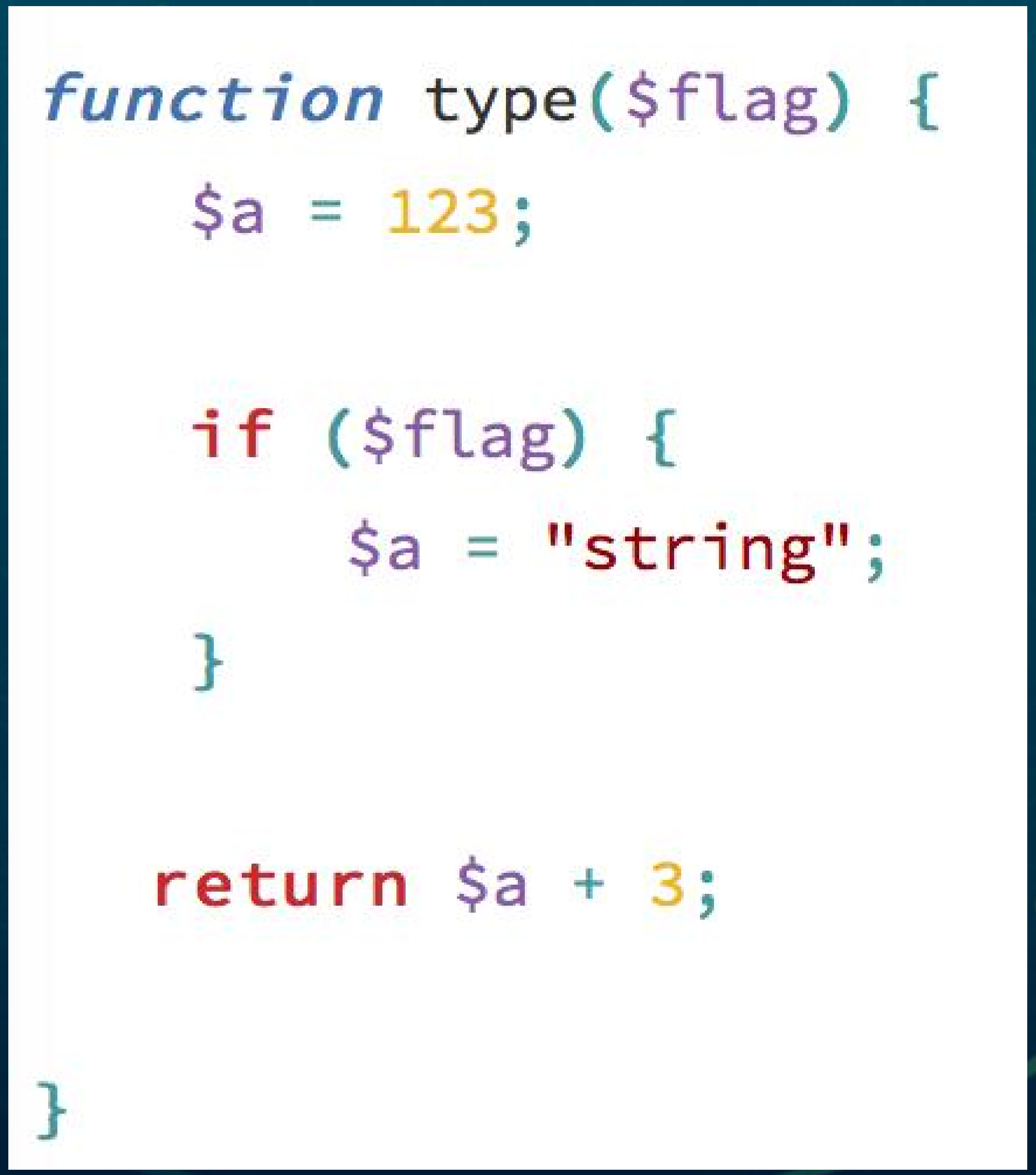
Type Inference system

- Part work of our JIT in 2013



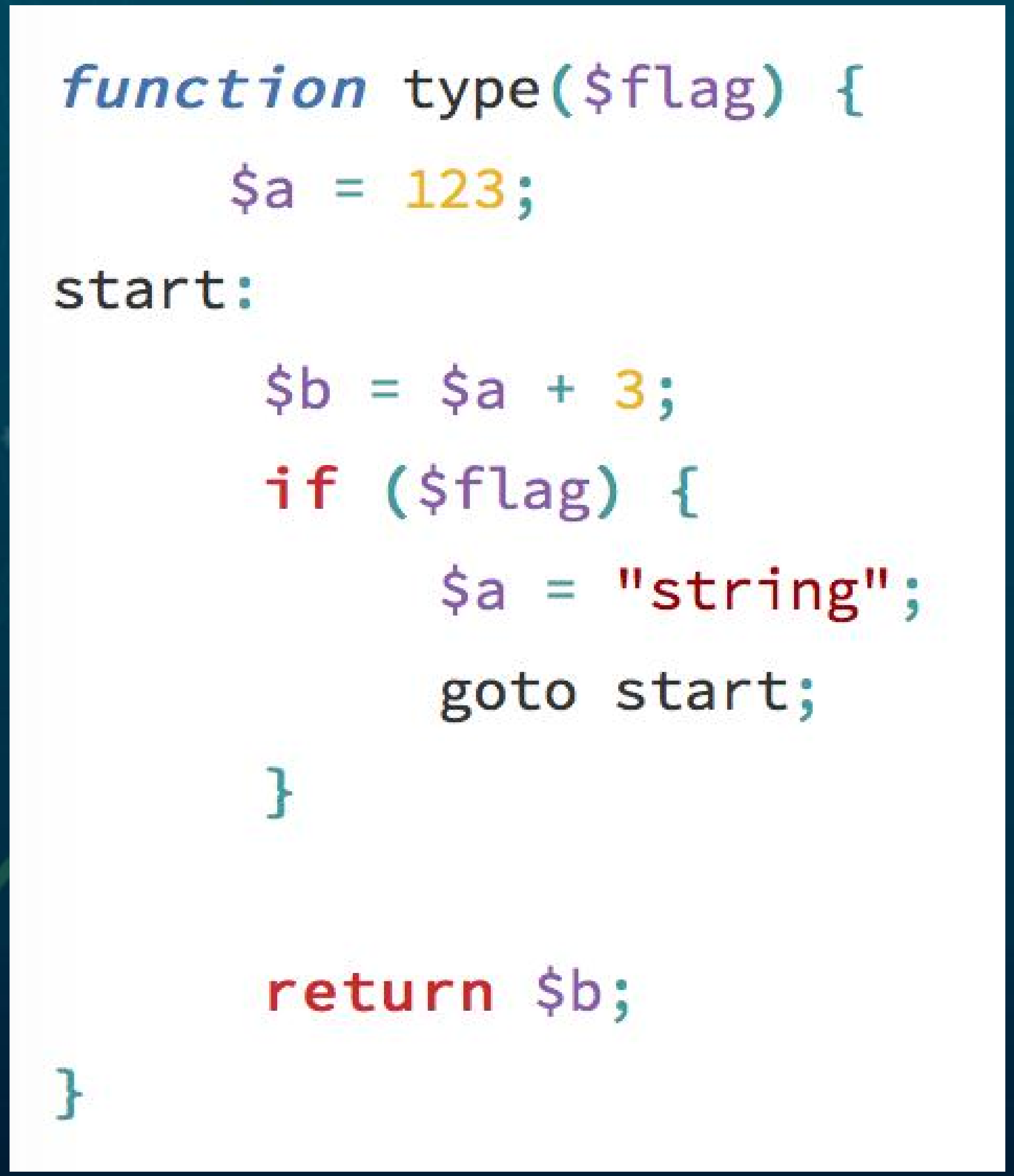
```php
function type() {
    $a = 123;

    //no change to $a

    ......

    return $a + 3;

}
```

normal



```php
function type($flag) {
    $a = 123;

    if ($flag) {
        $a = "string";
    }

    return $a + 3;

}
```

branch



```php
function type($flag) {
    $a = 123;
start:
        $b = $a + 3;
        if ($flag) {
            $a = "string";
            goto start;
        }

        return $b;
}
```

loop

SSA

- Static Single-Assignment form
- Explicit use-def chain

```
function type() {

    $a = 123;
    $b = $a + 3;
    $a = (string)$b;
    $c = $a + 2;

    return c;
}
```

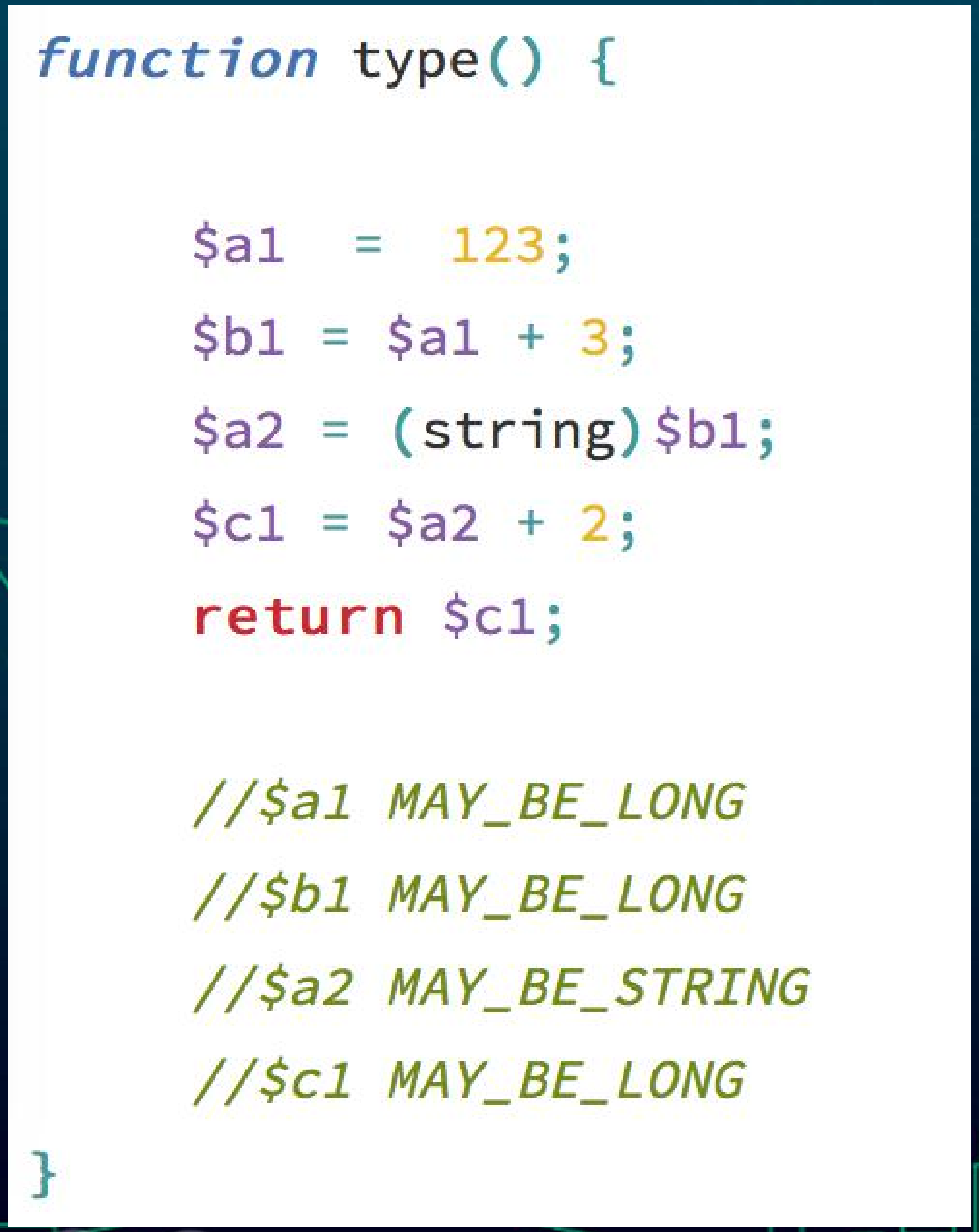
```
function type() {

    $a1  =  123;
    $b1 = $a1 + 3;
    $a2 = (string)$b1;
    $c1 = $a2 + 2;

    return $c1;
}
```

```
function type($flag) {
    $a = 123;

    if ($flag) {
        $a = "string";
    }

    return $a + 3;
}
```

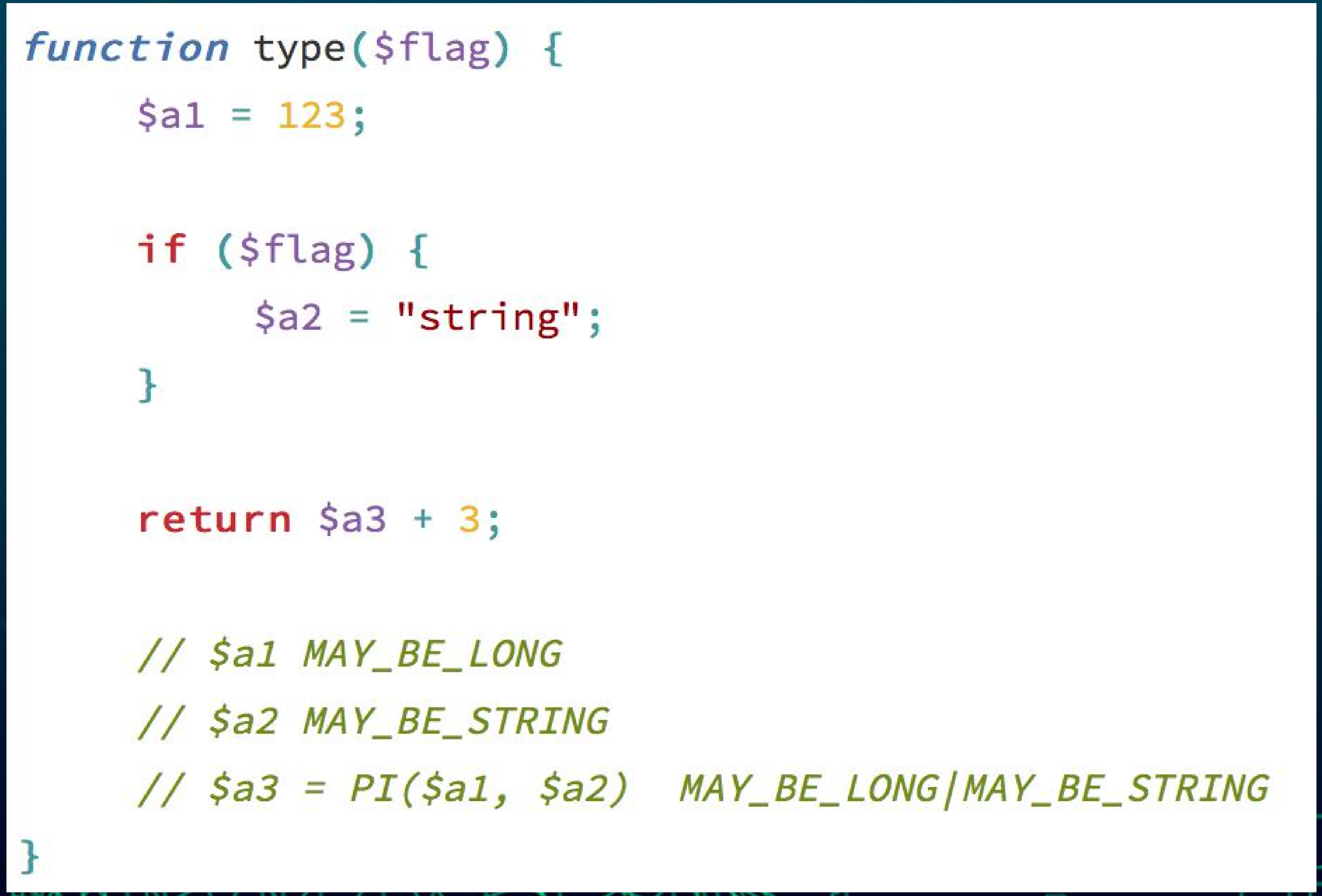
```
function type($flag) {
    $a1 = 123;

    if ($flag) {
        $a2 = "string";
    }

    return $a(?) + 3;
}
```

Type Inference

- What’ s ? ’ s type at ? point
- Computed in compiling time



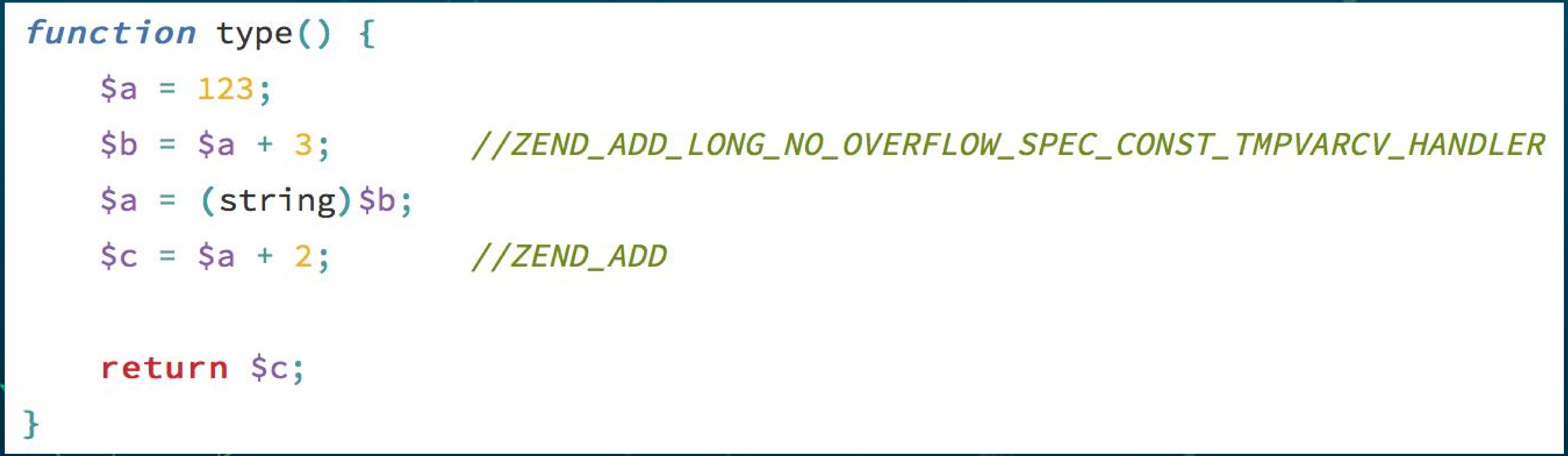
```
function type() {

    $a1  =  123;
    $b1 = $a1 + 3;
    $a2 = (string)$b1;
    $c1 = $a2 + 2;
    return $c1;

    //$a1 MAY_BE_LONG
    //$b1 MAY_BE_LONG
    //$a2 MAY_BE_STRING
    //$c1 MAY_BE_LONG
}
```



```
function type($flag) {
    $a1 = 123;

    if ($flag) {
        $a2 = "string";
    }

    return $a3 + 3;

    // $a1 MAY_BE_LONG
    // $a2 MAY_BE_STRING
    // $a3 = PI($a1, $a2)  MAY_BE_LONG|MAY_BE_STRING
}
```

Type Specific opcode handler

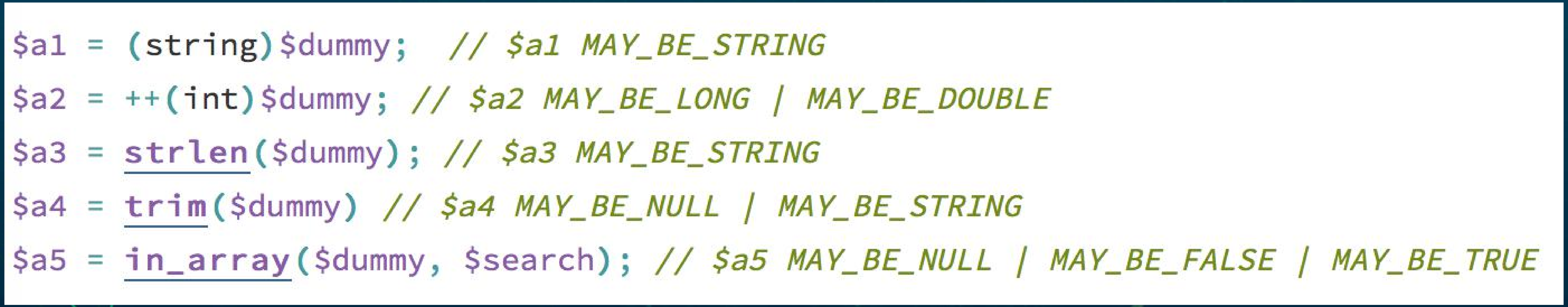
- Use type specific opcode handler if possible



```php
function type() {
    $a = 123;
    $b = $a + 3;        //ZEND_ADD_LONG_NO_OVERFLOW_SPEC_CONST_TMPVARCV_HANDLER
    $a = (string)$b;
    $c = $a + 2;        //ZEND_ADD

    return $c;
}
```

Type Inference system

- A tedious work
- And we can only get ~30% type-infos in WP



```
$a1 = (string)$dummy;   // $a1 MAY_BE_STRING
$a2 = ++(int)$dummy; // $a2 MAY_BE_LONG | MAY_BE_DOUBLE
$a3 = strlen($dummy); // $a3 MAY_BE_STRING
$a4 = trim($dummy) // $a4 MAY_BE_NULL | MAY_BE_STRING
$a5 = in_array($dummy, $search); // $a5 MAY_BE_NULL | MAY_BE_FALSE | MAY_BE_TRUE
```

Type Specific opcode handlers

- ZEND_ADD(SUB|MUL)
- ZEND_PRE_INC(DEC)
- ZEND_POST_INC(DEC)
- ZEND_IS_(NOT_)EQUAL
- ZEND_IS_SMALLER(_OR_EQUAL)
- ZEND_QM_ASSIGN
- ZEND_SEND_VAR(_EX)
- ZEND_FETCH_DIM_R

And dozens of minor improvements

- More packed array constructions
- Constant propagation based on DFA
- Return type checks eliding
- dozens of minor improvements even I can not recall

- Q&A