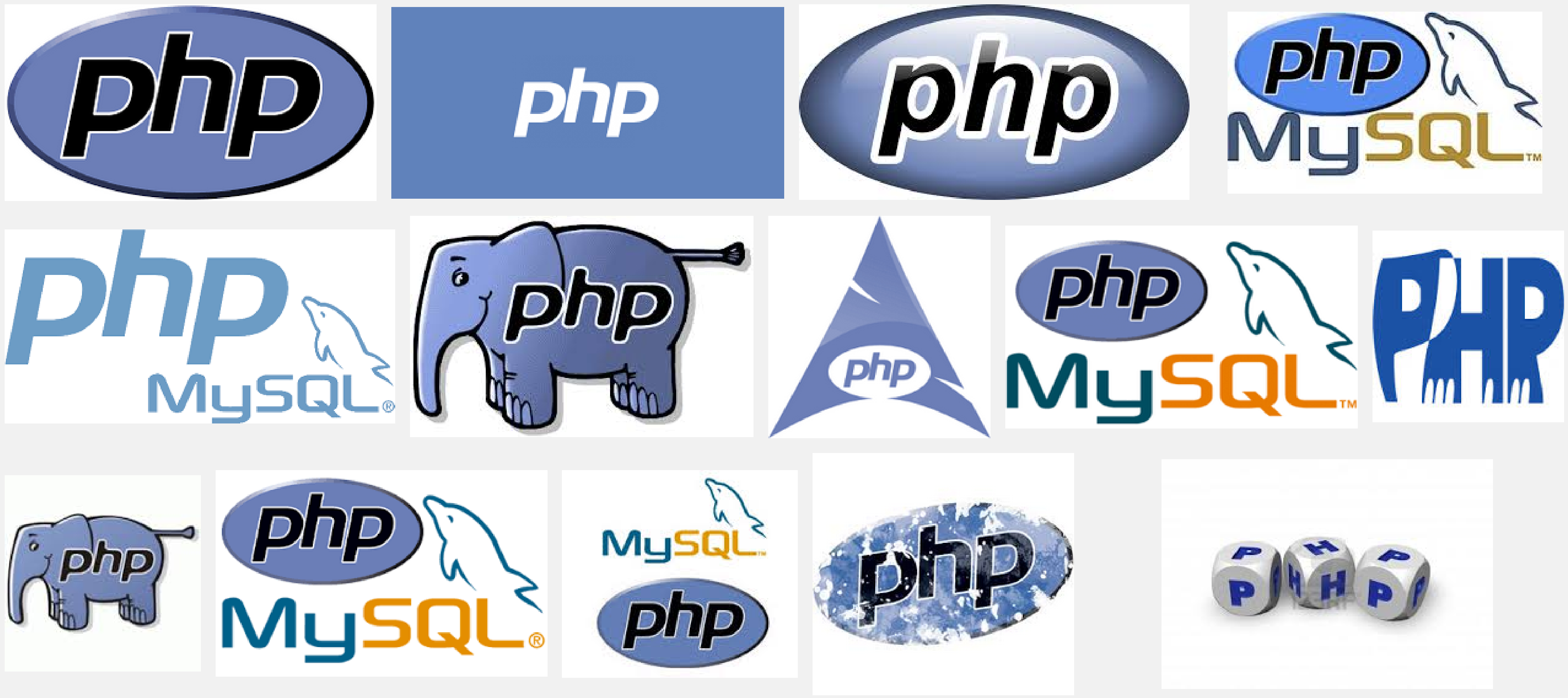

php
php
php
php
MySQL
php
MySQL
php
php
php
MySQL
PHR
php
php
MySQL
MySQL
php
php
PHP

50 shades of PHP

By Max Gopey

50 shades of PHP

Chapter 1

PHP: The Protoduction Way?

Protoduction — A prototype that ends up in production.

Rasmus Lerdorf

Announce: Personal Home Page Tools (PHP Tools)

These tools are a set of small tight cgi binaries written in C.

They perform a number of functions including:

- Logging accesses to your pages in your own private log files
- Real-time viewing of log information
- Providing a nice interface to this log information
- Displaying last access information right on your pages
- Full daily and total access counters

...

The tools also allow you to implement a guestbook or any other form that needs to write information and display it to users later in about 2 minutes.

...

June 1995	PHP Tools 1.0 (Personal Home Page Tools)
September 1995	FI (Forms Interpreter)
October 1995	Personal Home Page Construction Kit
April 1996	PHP/FI
June 1996	PHP/FI 2.0 (beta)
November 1997	PHP/FI 2.0

```
<!-- ===== PHP/FI Code Example ===== -->
<!--include /text/header.html-->

<!--getenv HTTP_USER_AGENT-->
<!--ifsubstr $exec_result Mozilla-->
  Hey, you are using Netscape!<p>
<!--endif-->

<!--sql database select * from table where user='$username'-->
<!--ifless $numentries 1-->
  Sorry, that record does not exist<p>
<!--endif exit-->
  Welcome <!--$user-->!<p>
  You have <!--$index:0--> credits left in your account.<p>

<!--include /text/footer.html-->
```

Zeev Suraski

Andi Gutmans

June 1995	PHP Tools 1.0 (Personal Home Page Tools)
September 1995	FI (Forms Interpreter)
October 1995	Personal Home Page Construction Kit
April 1996	PHP/FI
June 1996	PHP/FI 2.0 (beta)
November 1997	PHP/FI 2.0
June 1998	PHP 3.0 (PHP: Hypertext Preprocessor)

- **PHP — PHP: Hypertext Preprocessor**
- Ace — Ace Code Editor
- cURL — Curl URL Request Library
- GNU — GNU's Not Unix
- Nano — Nano's ANOther editor
- PIP — PIP Installs Packages
- YAML — YAML Ain't Markup Language

I don’t know how to stop it, there was never any intent to write a programming language […] I have absolutely no idea how to write a programming language, I just kept adding the next logical step on the way.

— Rasmus Lerdorf

```
List:       php-general
Subject:    Re: [PHP3] how to escape <INPUT TYPE=TEXT VALUES... (slightly offtopic)
From:       Rasmus Lerdorf <rasmus () lerdorf ! on ! ca>
Date:       1998-09-15 16:07:47


> i do not know how to do escaping so i can enter (INCLUDING the "
> characters)
>
> "it's time" ---- and get ----- "it's time"
> or
> 'i love "dogs"' and get 'i love "dogs"'
>
> i can either try ..VALUE='<?php echo $hallo; ?>'
> or VALUE="<?php echo $hallo; ?>"
>
> but anyway that's not as flexible as i want to be....

You should always have quotes around your value element unless you
are absolutely sure it is only a single word.  And you should probably
have a look at the url_encode() function.

-Rasmus
```

June 1995	PHP Tools 1.0 (Personal Home Page Tools)
September 1995	FI (Forms Interpreter)
October 1995	Personal Home Page Construction Kit
April 1996	PHP/FI
June 1996	PHP/FI 2.0 (beta)
November 1997	PHP/FI 2.0
June 1998	PHP 3.0 (PHP: Hypertext Preprocessor)
May 2000	PHP 4.0
July 2004	PHP 5

Conclusions

1. Solve real tasks — this may result in something useful.
2. Don't be afraid to re-implement.
3. Modularity rules.
4. Share — open-source is a power.

Chapter 2

PHP: Dynamic typing

Why do we love it so much?

A language is dynamically typed if the type is associated with run-time values, and not named variables/fields/etc.

— Paul Biggar

```php
$foo = 5; // int
$foo = "hello"; // string
$foo = [1.2, 3.89]; // array of floats
```

```
# Python
>>> foo = "10"
>>> print foo - 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -:
                                    'str' and 'int'
```

```
// PHP
php > $foo = ”10”;
php > print $foo - 5;
5
```

Conclusion

When developing an application aim to help your end user.

Try to solve side-problems for them and provide them with ability to focus on really important things.

Chapter 3

PHP and Unicode

Do we have at least a chance?

```
php > echo strlen("Hello");
5
php > echo strlen("Привет");
12
```

Andrei Zmievski

STRINGS

- String literals are Unicode
- String offsets work on code points

```
$str = "大学";   // 2 code points
echo $str[1];   // result is 学
$str[0] = 'サ'; // full string is now サ学
```

IDENTIFIERS

- Unicode identifiers are allowed

```
class コンポーネント {

   function ᓱᓴᓐ ᐊᒡᓗᒃᑲᖅ()    { ... }
   function சிவாஜி கணேசன்()  { ... }
   function འབྲུག་ཡུལ།()        { ... }
}

$プロバイダ = array();
$プロバイダ['רַעְיולוּחַ שָׁנָה'] = new コンポーネント;
```

FUNCTIONS

- Functions understand Unicode text and apply appropriate rules
- i.e. case manipulation

```
$str = strtoupper("fußball");  // result is FUSSBALL

$str = strtolower("ΣΕΛΛΑΣ");   // result is σελλάς
```

TRANSLITERATION

```
$names = "
  김, 국삼
  김, 명희
  たけだ, まさゆき
  おおはら, まなぶ
  Горбачев, Михаил
  Козырев, Андрей
  Καφετζόπουλος, Θεόφιλος
  Θεοδωράτου, Ελένη
";
$r = strtotitle(str_transliterate($names, "Any", "Latin"));
```

```
Gim, Gugsam
Gim, Myeonghyi
Takeda, Masayuki
Oohara, Manabu
Gorbačev, Mihail
Kozyrev, Andrej
Kaphetzópoulos, Theóphilos
Theodōrátou, Eléné
```

UTF-8

vs

UTF-16

vs

UTF-32

UTF-8

vs

UTF-16

vs

UTF-32

Intl

- **Collator:** provides string comparison capability with support for appropriate locale-sensitive sort orderings.
- **Number Formatter:** allows to display number according to the localized format or given pattern or set of rules, and to parse strings into numbers.
- **Message Formatter:** allows to create messages incorporating data (such as numbers or dates) formatted according to given pattern and locale rules, and parse messages extracting data from them.
- **Normalizer:** provides a function to transform text into one of the Unicode normalization forms, and provides a routine to test if a given string is already normalized.
- **Locale:** provides interaction with locale identifiers in the form of functions to get subtags from locale identifier; parse, compose, match(lookup and filter) locale identifiers.

Multibyte String

mbstring provides multibyte specific string functions that help you deal with multibyte encodings in PHP. In addition to that, *mbstring* handles character encoding conversion between the possible encoding pairs. *mbstring* is designed to handle Unicode-based encodings such as UTF-8 and UCS-2 and many single-byte encodings for convenience.

Conclusions

(All taken from Andrei Zmievski's speech)

- People matter.
- Rewriting large existing code base is hard.
- Making people do tedious stuff is hard.
- Waiting for results of long iterations is hard.
- Stay committed.

Chapter 4

PHP Nowadays

The quick overview of features released with PHP 5.1–5.6.

Version 5.1.0 (24 Nov 2005)

- Added support for class constants and static members for internal classes.
- Added array type hinting.
- Added a lot of new functions for arrays, streams processing, PostgreSQL v3, etc.
- Many performance improvements

Version 5.2.0 (02 Nov 2006)

- Added Zip Archive extension.
- Added JSON and Filter extensions.
- Added support for constructors in interfaces.
- Many-many new improvements

Version 5.3.0 (30 Jun 2009)

- Added lambda functions and closures.
- Added "jump label" operator (limited "goto"). 😈
- Added NOWDOC & HEREDOC syntax.
- Added "?:" operator.
- Added support for namespaces.
- Added support for Late Static Binding.
- Added __DIR__ constant.

Version 5.4.0 (01 Mar 2012)

- Added short array syntax support ([1,2,3])
- Added support for Class::{expr}() syntax.
- Added support for Traits.
- Added closure $this support back.
- Added callable typehint
- Added array dereferencing support.
- Added class member access on instantiation (e.g. (new foo)->bar()) support.

Version 5.5.0 (20 Jun 2013)

- Added generators and coroutines.
- Added "finally" keyword.
- Added simplified password hashing API.
- Added support for constant array/string dereferencing.
- Added Class Name Resolution As Scalar Via "class" Keyword
- Added support for non-scalar Iterator keys in foreach

Version 5.6.0 (28 Aug 2014)

- Added constant scalar expressions syntax.
- Added dedicated syntax for variadic functions.
- Added an exponentiation operator (**).
- Added use function and use const.

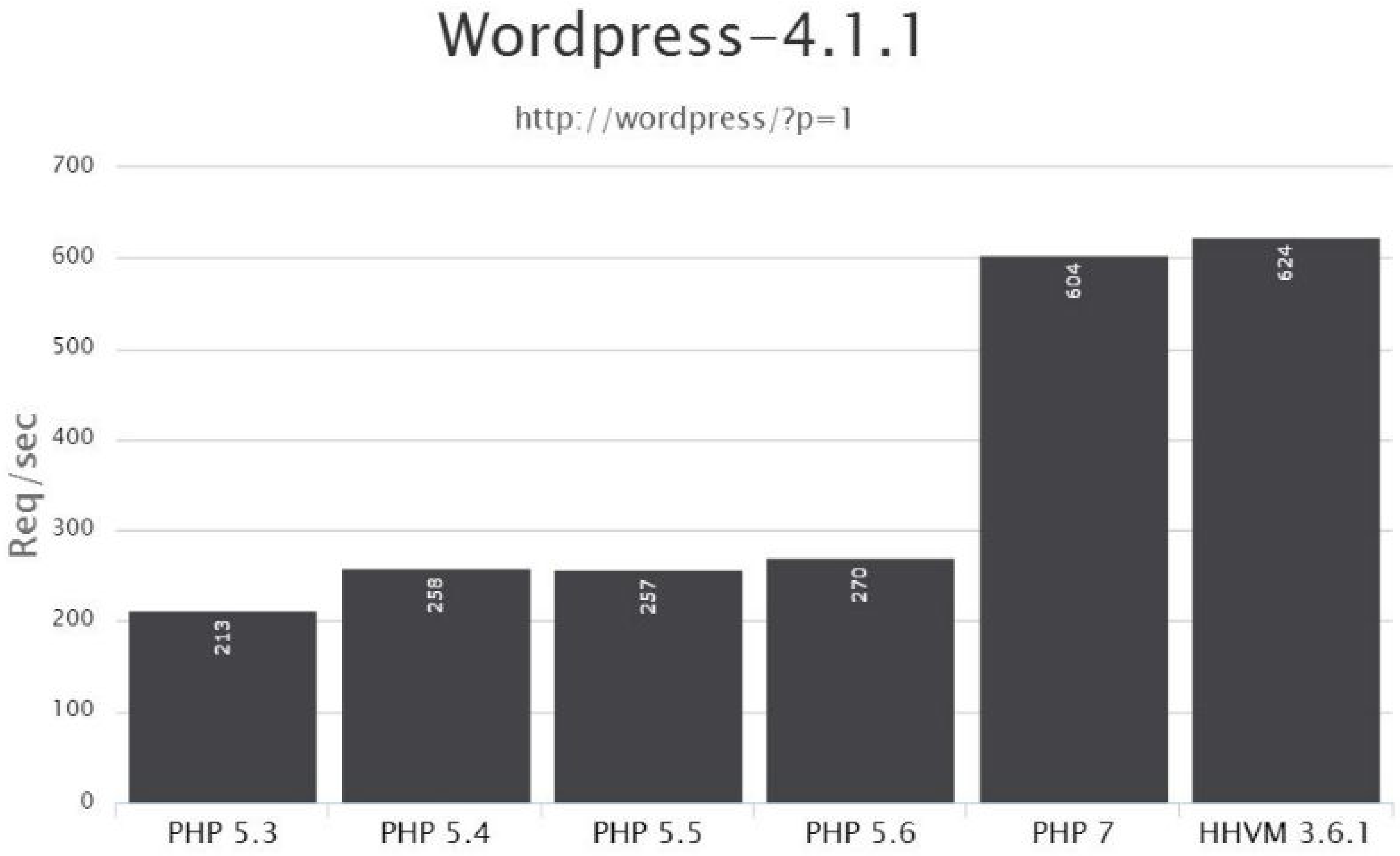
It's up to you

Chapter 5

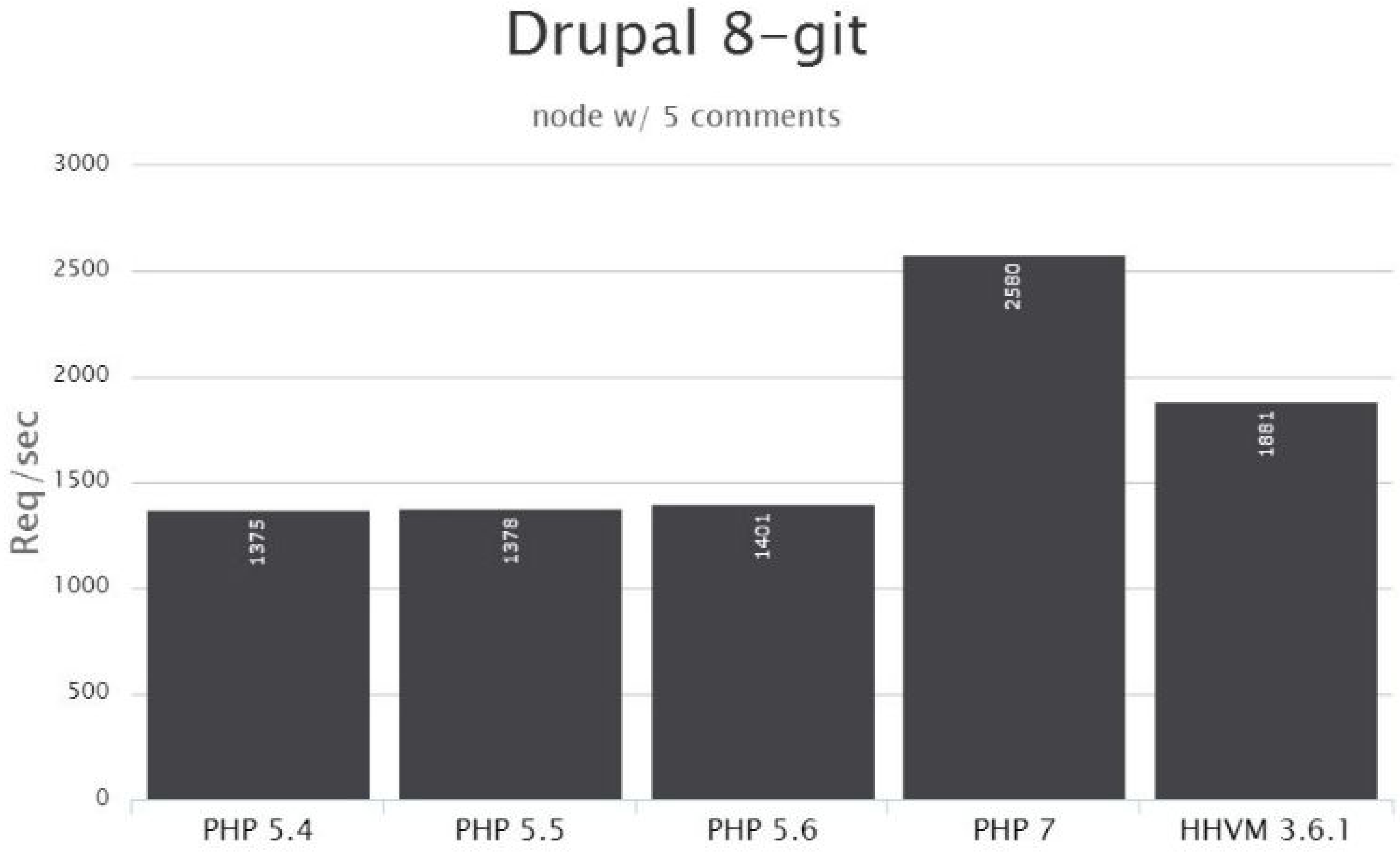
PHP 7

What is all fuzz about?

Oct 29 2015	PHP 7 RC 6
Nov 12 2015	PHP 7 Final



Wordpress-4.1.1
http://wordpress/?p=1
Req/sec
700
600
500
400
300
200
100
0
213
258
257
270
604
624
PHP 5.3
PHP 5.4
PHP 5.5
PHP 5.6
PHP 7
HHVM 3.6.1


Drupal 8–git
node w/ 5 comments
Req/sec
3000
2500
2000
1500
1000
500
0
1375
1378
1401
2580
1881
PHP 5.4
PHP 5.5
PHP 5.6
PHP 7
HHVM 3.6.1

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types

```php
<?php
(function(string $a) {
    var_dump($a);
})(15);
// string(2) "15"
```

```php
<?php
(function(array $a) {
    var_dump($a);
})(15);
// Fatal error: Uncaught TypeError:
//     Argument 1 passed to {closure}()
//     must be of the type array, integer given ...
```

```php
<?php
$a = (function(string $a) : int {
    return $a; // string
})(15);
var_dump($a); // int(15)
```

```php
<?php
declare(strict_types=1); // must be the first line

(function(string $a) {
    var_dump($a);
})(15);
// Fatal error: Uncaught TypeError:
//     Argument 1 passed to {closure}()
//     must be of the type string, integer given ...
```

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types
- **Combined Comparison Operator**

<=>

```php
$a <=> $b;
// instead of:
($a < $b) ? -1 : (($a > $b) ? 1 : 0);
```

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types
- Combined Comparison Operator
- **Null Coalesce Operator**

??

```php
$config = $this->config ?? static::$defaultConfig;
// instead of:
$config = isset($this->config)
    ? $this->config : static::$defaultConfig;
```

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types
- Combined Comparison Operator
- Null Coalesce Operator
- **Bind Closure on Call**

```php
class Person {
    public $name = 'Bob';
}
(function() {
    echo $this->name;
})->call(new Person);
// Bob
```

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types
- Combined Comparison Operator
- Null Coalesce Operator
- Bind Closure on Call
- **Grouped Use Declarations**

```php
use God\Save\{
    TheQueen,
    TheKing,
    Whatever
}
```

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types
- Combined Comparison Operator
- Null Coalesce Operator
- Bind Closure on Call
- Grouped Use Declarations
- **Generators return expressions and delegation**

```php
function hello() {
      yield "Hello";
      yield "World!";
      yield from goodbye();
      return "Whew!";
}
function goodbye() {
      yield "Goodbye";
      yield "Moon!";
}
$gen = hello();
foreach ($gen as $value) {
      echo $value, ' '; // Hello World! Goodbye Moon!
}
echo $gen->getReturn(); // Whew!
```

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types
- Combined Comparison Operator
- Null Coalesce Operator
- Bind Closure on Call
- Grouped Use Declarations
- Generators return expressions and delegation
- **New Exceptions Hierarchy**

Exceptions Hierarchy

```
Throwable                                 (Interface)
├── Exception                             (Predefined)
│   ├── ErrorException                    (Predefined)
│   ├── LogicException                    (SPL)
│   │   ├── BadFunctionCallException      (SPL)
│   │   │   └─ BadMethodCallException     (SPL)
│   │   ├── DomainException               (SPL)
│   │   ├── InvalidArgumentException      (SPL)
│   │   ├── LengthException               (SPL)
│   │   └── OutOfRangeException           (SPL)
│   └── RuntimeException                  (SPL)
│       ├── OutOfBoundsException          (SPL)
│       ├── RangeException                (SPL)
│       ├── OverflowException             (SPL)
│       ├── UnderflowException            (SPL)
│       └── UnexpectedValueException      (SPL)
└── Error                                 (Predefined)
    ├── ArithmeticError                   (Predefined)
    │   └── DivisionByZeroError           (Predefined)
    ├── AssertionError                    (Predefined)
    ├── ParseError                        (Predefined)
    └── TypeError                         (Predefined)
```

```php
try {
    nonExistentFunction();
} catch (\Exception $e) {
     echo 'Exception: '; var_dump($e);
} catch (\Error $e) {
     echo 'Error: '; var_dump($e);
}
```

```
Error: object(Error)#1 (7) {
  ["message":protected]=>
  string(48) "Call to undefined function nonExistentFunction()"
  ["string":"Error":private]=>
  string(0) ""
  ["code":protected]=>
  int(0)
  ...
}
```

PHP 7 New Features

- Abstract syntax tree & Uniform Variable Syntax
- Scalar Type Hints & Return Types
- Combined Comparison Operator
- Null Coalesce Operator
- Bind Closure on Call
- Grouped Use Declarations
- Generators return expressions and delegation
- New Exceptions Hierarchy
- ... and a lot of other stuff

That's all Folks!