Functional PHP

A GLIMPSE INTO THE FUTURE

By Andrew Caya, ZCE, ZCA

Who am I?

- I am Andrew Caya
- Started out with GW-BASIC and QBASIC in 1991
- C, C++ (Qt), Perl
- Linux System Administration
- PHP developer since 2009
  - Zend Certified Engineer since 2015
  - Zend Certified Architect since 2016
- « Mercenary » developer since 2010 (thanks to Tim Lytle for the term)
- Technical Reviewer for Packt Publishing since 2016
- Upcoming projects :
  - Author of a book on the Faster Web (due to be published later this year)
  - Creator of Linux for PHP (will be published in a few days from now)

Functional programming

What is all the hype about?

- What is so new about functional programming?
- How does this programming paradigm concern PHP?
- How can it help us on a day to day basis as developers?

Is functional programming really a new thing?

NOPE!

Let's start with a glimpse into the past…

A glimpse into the past

- A brief history
  - Aristotle (4th Century BC)

Principle of identity – « What is is, what is not is not. »
VALUE – ex. 1

Principle of the excluded middle – « A thing is or is not. »
STATE – ex. $x = 1

Principle of contradiction – « A thing cannot belong and not belong to the same thing at the same time and in the same respect. »
CONDITIONAL REASONING – ex. if ($x === 1)

A glimpse into the past

- A brief history
  - 1854 : George Boole's *The Laws of Thought* (birth of pure mathematics by applying Cartesian symbols to Aristotelian logic in order to determine the validity of any statement)

x (1 − x) = 0;

"[…] a class whose members are at the same time men and not men does not exist."

A glimpse into the past

- A brief history
  - Origin of FP : Lambda calculus (Alonzo Church -1932)
    - Mathematicians and logicians were hard at work to develop a logical system in order to help us formalize the way we describe the world using pure mathematics (Boole's *The Laws of Thought)*.
    - Lambda calculus marks the end of this attempt to describe the world in this way.

"There may, indeed be other applications of the system than its use as a logic."

  - 1940s : Birth of effective computation
    - Lambda calculus
    - Turing machine
    - Kurt Godel's recursive functions
    - Haskell Curry's combinatory logic

Functional programming allowed for easier effective computation

Functional programming predates all other major programming paradigms!

So, why were other paradigms invented?

A question of efficiency

A glimpse into the past

- Declarative (pure functional) vs Imperative
  - Declarative programming is value-oriented and is based on expressions and declarations
  - Imperative programming is concerned with efficiency rather than suitability of the language, is state-oriented and is based on the use of statements

Mutually exclusive in the absolute sense

A glimpse into the past

- Functional vs Structural vs Object-Oriented
  - Functional programming considers computational design as being based on mathematical functions, avoids changing state and making data mutable
  - Structural programming makes extensive use of subroutines, block structures, for and while loops
  - Object-oriented programming organizes code in easily reusable and maintainable units called objects.

Not mutually exclusive

A glimpse into the past

- A brief history
  - Functional languages (functional hybrids) :
    - LISP (1958), ML (1973), Erlang (1986), Scala (2001), F# (ML family) (2005), Clojure (LISP dialect) (2007)
  - Declarative languages (pure functional) :
    - Prolog (1972), SQL (1974), Miranda (1985), Haskell (1990), Mercury (1995), Agda (2007)
  - Imperative languages (mostly structural and object-oriented paradigms) :
    - ALGOL 58 (1958), ALGOL 60 (1960), BASIC (1964), C (1972), C++ (1983), Perl (1987), Python (1990), PHP (1994), Java (1994), Ruby (1995)

Why are we talking about FP now?

Why are we talking about FP now?

- Modern problems:
  - Complex application critical paths (burden for the developer)
  - Complexity when unit testing
  - Complexity when refactoring legacy code
  - Distributed systems
  - Parallelization/Multithreading (coming soon to a PHP server near you!)

- Functional programming is the solution to these problems
  - Simpler critical paths - Lighten the developer's burden (1 function = 1 action)
  - Easier unit testing
  - Avoids ugly stuff like race conditions and application state conflicts between threads
  - Free code optimizations (compiler optimizations and memoization)
  - Future performance boosts…

What is functional programming (FP)?

What is FP?

- In computer science, functional programming is a programming paradigm—a style of building the structure and elements of computer programs—that treats computation as the evaluation of mathematical functions and avoids changing-state and mutable data. It is a declarative programming paradigm, which means programming is done with expressions[1] or declarations[2] instead of statements.

  - Wikipedia, https://en.wikipedia.org/wiki/Functional_programming

- Programming where your entire program is a single referentially transparent expression composed of other referentially transparent expressions. No side effects. No mutability. No global mutable state.

  - Runar Bjarnason (Functional programmer, *@runarorama*)

Is PHP functional?

What is functional PHP?

- PHP is :
  - Imperative in nature
    - Structural (procedural)
    - Partly object-oriented since PHP 3

What is functional PHP?

- PHP 5.3 … 7.1
  - Anonymous functions (lambda functions)
  - Generators (infinite lists)
  - Functors (not real FP functors though!)
  - Anonymous classes
  - Variadics
  - PHP 7's strict mode

What is functional PHP?

- Functional PHP libraries
  - lstrojny/functional-php
  - phpoption/phpoption
  - widmogrod/php-functional
  - qaribou/immutable.php
  - etc.

Most functional programming patterns and properties are available in PHP!

Functional programming properties

FP Properties

- Pure functions and referential transparency
- Immutability
- First-class citizen functions
  - Higher-order functions
  - Function composition (currying)

FP Properties

Pure functions

- Same input, same output
- Evaluation does not cause an observable side effect or output (modifying out of scope variables or any other interaction with I/O devices for example)

```
// pure function

function addTwo (int $value) : int
{
    return $value + 2;
}
```

```
// two side-effects

function addTwo (int $value)
{
    global $value = $value + 2;
    echo $value;
}
```

FP Properties

Referential transparency

- Functional expressions and values must be interchangeable

```
function addTwo (int $value) : int
{
    return $value + 2;
}

4 === addTwo(2); // Interchangeable
```

```
function addRandom (int $value) : int
{
    return $value + rand();
}

? === addRandom(2); // NOT!
```

FP Properties

Immutability

- A variable must not change its value in order to avoid changing the application's state from beginning to end of its runtime
- RFC for PHP 7.2 (immutable objects)

  https://wiki.php.net/rfc/immutability

```
// Global scope – the right way

$value2 = $value + 1;
```

```
// Global scope – the wrong way

$value++;
```

FP Properties

First-Class Citizen Functions

- Functions must be considered just like any other data type.

- This allows:
  - Higher-order functions whereby functions can be passed to and returned by other functions;
  - Function composition whereby functions can be combined in order to dynamically generate new functions.

FP Properties

Higher-order functions (passing functions)

```
function sum($carry, $item)
{
    $carry += $item;
    return $carry;
}

$total = array_reduce($array, 'sum');
```

FP Properties

Higher-order functions (returning functions and currying)

```
function curryAdd($a)
{
   return function ($b) use ($a) {
      return $b + $a;
   };
}

$curryAdd2 = curryAdd(2);
$curryAdd3 = curryAdd(3);

$value = $curryAdd2(3); // 5
$value2 = $curryAdd3(3); // 6
```

Functional programming patterns

FP Patterns

MAP

- Higher-order function that allows us to map a callback to each element of a collection

```
$array = [1, 2, 3];

$newArr = array_map('addTwo', $array);

// $newArr === [3, 4, 5];
```

FP Patterns

FILTER

- Higher-order function that allows us to distinguish and keep only certain elements of a collection based on a Boolean predicate

```
$array = [1, 2, 3];

$newArr = array_filter($array , 'odd');

// $newArr = [1, 3];
```

FP Patterns

REDUCE

- Higher-order function that allows us to combine elements of a collection into a single returned value based on a combining function

```
$array = [1, 2, 3];

$value = array_reduce($array , 'sum');

// $value === 6
```

Let's look at some code!

Now your code is pure!

Let's all go home !

Simple!
Just get rid of the world!

Then, no need to say hello, right?

But wait!
I'm trying to do a « Hello World » program in a functional programming style without losing referential transparency and purity…

Let’s find a more viable solution!

In FP, we can use « monads » to interact with the world while preserving purity and referential transparency

What are monads

Monads are a way to encapsulate values that will remain unknown until runtime while still allowing us to use them as mappables.

What are monads

Functors

- Pattern allowing us to map a function to one or more wrapped values

```
interface Functor
{
    public function map(callable $f) : Functor;
}
```

What are monads

Applicative

- Pattern allowing us to map a wrapped function to one or more wrapped values

Let's not and say we did… :-)

Ref. :
Functional Programming in PHP
by Simon Holywell
https://www.functionalphp.com/

What are monads

Monad

- Pattern allowing us to map a wrapped function that returns a monad of the same type as itself to one or more wrapped values

```
abstract class Monad
{
    protected $value = null;

    public function __construct($value) {
        $this->value = $value;
    }

    public static function pack($value) {
        return new static($value);
    }

    public function map(callable $function) {
        return $function($this->value);
    }
}
```

The future is now !

Takeaways

- Try replacing if-else structures, while loops, switches with FP patterns as much as possible
- Make all dependencies explicit in your function signatures and avoid setter injection
- Create new variables, don't modify existing ones (clone objects)
- Using Zend Framework, Symphony or Laravel?
  - Try containing your side-effects within your controllers
  - Isolate your pure computational code within services and entities
  - Create a Doctrine repository in order to encapsulate results in Maybe monads
  - Avoid using façades (Laravel)
- Using Drupal?
  - Isolate impure code in the main module file
- Using Wordpress?
  - Create many files for your plugins and isolate the impure code in one main file

References

- Functional PHP, Gilles Crettenand
  https://www.packtpub.com/application-development/functional-php
- Functional Programming in PHP, Second Edition, Simon Holywell
  https://www.functionalphp.com/
- Clean Coder, Robert C. Martin (Uncle Bob)
  Mr. Martin will be in Montreal, May 17-18
  https://sites.google.com/site/unclebobconsultingllc/

Thank you!

https://joind.in/talk/3fbb6

Andrew Caya, ZCE, ZCA

@AndrewSCaya

https://ca.linkedin.com/in/andrewscaya