# FUNCTIONAL PHP

SIMON HOLYWELL

12/05/2014

# WHO?

- Lead developer
- Mosaic in Brighton
- Full service agency
- @Treffynnon on Twitter
- SimonHolywell.com

# WORK

- Web apps
- Obligatory CMS!

# PROJECTS

- Maintainer of Idiorm and Paris
- ssdeep - PECL and HHVM extension
- Navigator - geo library for PHP
- PHP-at-Job-Queue-Wrapper
- njsbot - Node.js bot for GTalk - now defunct :(

# FUNCTIONS IN PHP

- Rarely employed
- Discouraged by PHP's evolution
- Naming collisions
- The old name spacing object

# SIMPLE BEASTS

PHP functions are:

- Easy
- Contained
- Unaware of global state
- Can now be namespaced

# FUNCTION

```php
<?php
function hello($arg = 'World') {
    echo __FUNCTION__ . " $arg";
}
hello(); // hello World
```

# A LITTLE BIT OF HISTORY

- I started with PHP3
- Functions were the norm
- Classes merely wrapped them
- OOP was very poorly understood

# DRY

- Most mashed code with HTML
- Functions promoted code reuse
- HTML then had functions in it!

# EVOLVE

- Come PHP4 you get better OOP
- Functions no longer in focus
- They become seen poorer cousins
- Can't carry context vs. class properties

# PHP5

- Functions are still marginalised
- Still seen as spaghetti
- The bad way

# PHP5.3+

- Proper anonymous functions
- We get closures!

# THE BASICS ARE IN PLACE

¡Functional is possible!

# JUST WHAT IS FUNCTIONAL?

- Programming without assignment
- Stateless
- Pure

# WITHOUT ASSIGNMENT...

Say goodbye to

- `$var = 'value';`
- `$ClassInstance->property = 'value';`

# STATELESS AND PURE

No:

- Globals
- Side effects
- Incrementors/accumulators
- For, while or foreach loops

# PASSING VALUES

- From function to function as
  - Arguments
  - Return values

# 10

How would you sum all the integers from one to ten?

# IMPERATIVE WAY

```php
<?php
$sum = 0;
for($i = 1; $i <= 10; $i++) {
    $sum += $i;
}
// $sum = 55
```

# FUNCTIONAL WAY

```php
<?php
array_sum(range(1, 10));
```

- More descriptive
- Removes state (`$sum`)
- Reusable components

# LIKE WELL-DESIGNED OOP

- Encapsulated
- Broken down into small chunks
- Reusable

# CAN IT BE NEATER?

```php
<?php
array_sum(range(1, 10));
```

- Lua - `s=0; for i=1,10 do s=s+i end`
- Erlang - `lists:sum(lists:seq(1, 10)).`
- Clojure - `(reduce + (range (inc 10)))`
- Ruby - `(1..10).inject(0) {|s,i| s + i}`
- Python - `sum(range(1, 10 + 1))`
- F# - `List.sum [1 .. 10]`
- Scala - `(1 to 10).sum`
- Haskell - `sum[1..10]`

# THE SCALE

- Pure - Haskell
- Middle ground - Scala and Clojure
- Dabbling - PHP and Python
- Hopeless - Brainfuck

%

# FUNCTIONAL HISTORY

- Kurt Gödel - recursively calculated functions - 1930s
- Haskell Curry - combinatory logic - 1927
- Alan Turing - machines calculating from inputs - 1936
- Alonzo Church - lambda calculus - 1930s

# LAMBDA CALCULUS

- Mathematical functions - abstraction
- Accept functions as parameters
- Return other functions
- Basis of functional programming

# COMBINATORY LOGIC

- Haskell Curry
- Combinatory calculus - 1927
- Primitive functions combined to build others
- Named after him
  - Currying
  - Haskell

# WORLD WAR II

- Vienna/Göttingen unsafe
- Great minds move to US or UK
- Turing involved in code breaking

# ACADEMIC

- Little activity 'til late 1950s

# LISP

- 1958
- Jon McCarthy
- MIT
- Became the standard for AI

# TELEPHONES

- Fault tolerant system
- Erlang by Ericsson
- 1980s

# ERICSSON

- Ericsson
- Erlang
- Highly symbolic to improve productivity
- Lisp and Prolog discounted
- Focused on
  - Reliability
  - Concurrency

# HASKELL

- Formed by committee
- Pure functional language

# COMMERCIAL ADOPTION

- Domain specific languages (DSL)
- Type handling
- Pattern matching
- Fast to build complex parsers
- Less code = easier to maintain

# CASESTUDY

- Barclays bank
- Functional Payout Framework
- Written in Haskell
- DSL for mathematicians
- Builds trading applications
- Functional eased scope creap

^

# NONE SHALL PASS!

```php
<?php
$allowable_ips = array(
  '192.168.0.',
  '8.8.8.8',
);

array_filter($allowable_ips, function($ip) {
  return $ip ==
    substr($_SERVER['REMOTE_ADDR'], 0, strlen($ip));
}) ?: die('Denied.');
```

# ARRAY_FILTER()

```php
<?php
array_filter($allowable_ips, function($ip) {
```

- Takes a list and predicate
- "Removes" items when predicate fails
- λ or anonymous function
- And hey, it's a loop!

# THA PREDICATE

```php
<?php
function($ip) {
  return $ip ==
    substr($_SERVER['REMOTE_ADDR'], 0, strlen($ip));
}
```

- IP v.4 addresses
- Chops user IP to same length as allowable (`$ip`)
- Compares result for equality

# KILLER CONDITION

```php
<?php
 ?: die('Denied.');
```

- Empty arrays == false
- Terminates application
- Ternary shortcut

```php
<?php
expr1 ? expr2 : expr3;
expr1 ?: expr3;
```

# WHAT SAY THE BLACK KNIGHT?

```php
<?php
// $_SERVER['REMOTE_ADDR'] = '192.168.'
$allowable_ips = array(
  '192.168.0.',
  '8.8.8.8',
);
array_filter($allowable_ips, function($ip) {
  return $ip ==
    substr($_SERVER['REMOTE_ADDR'], 0, strlen($ip));
}) ?: die('Denied.');
```

# BONUS: ARRAY CLEANER

- `array_filter()` without a predicate
- Entries == false dropped
- false == "" == 0 == array() == null == '0' == 0.0

```php
<?php
array_filter(array('', 's', 0));
// array('s')
```

# BENEFITS OF FUNCTIONAL

- Techniques useful for OOP
- Abstraction = more readable
  - No distracting guff
  - Just the problem solving stuff

# ...AND MORE

- Shorter code = quicker to debug
- No global state to assemble in mind
  - Focus on the hard problems

# HEY! WAIT THERE'S STILL MORE

- Testing is easier
- No globals again
- Values are final

# REFERENTIAL TRANSPARENCY

- Replace any function with it's return value
- Algorithm still works!

```php
<?php
array_sum(array(1,2,3,4,5,6,7,8,9,10));
// Is still 55!
```

Can you do that to a for loop?

# NO!

```php
<?php
$sum = 0;
for($i = 1; $i <= 10; $i++) {
    $sum += $i;
}
// $sum = 55
```

Closest:

```php
<?php
$sum = 0;
$sum += 1;
$sum += 2;
...
$sum += 10;
// $sum = 55
```

# SO WHAT ELSE?

- Easier parallel processing
- Who's using the other cores in your CPU?
- Not common in PHP - see pthreads

# IF PHP WERE REALLY FUNCTIONAL

- Machine optimisation of code
- Lazy-evaluation
- Hot code deployment (Erlang!)

%

# STEP BACK - λ

- λ
- lambda function
- anonymous function

```php
<?php
$func_name = function($param) {
  return $param;
};
```

# SHOULD BE FAMILIAR

```php
<?php
spl_autoload_register(function ($class) {
    include 'classes/' . $class . '.class.php';
});
```

If not - how about in JS:

```javascript
button.onclick = function() {
  alert('Lambda');
};
```

# EXTEND IT - CLOSURE

- Just like a lambda
- , with context/data ( `$value` )

```php
<?php
$value = 5;
$func_name = function($param) use ($value) {
  return $param + $value;
};
echo $func_name(3); // 8
```

- Vars can be other closures or lambdas too
- No aliasing unfortunately

# MAP

- `array_map()`
- Applies function to each element
- Returns new array with adjusted elements

```php
<?php
array_map(
  function($v) { return strtoupper(++$v); },
  $array
);
```

`array('o', 'g', 'o')` becomes `array('P', 'H', 'P')`

# REDUCE

- `array_reduce()`
- Fold array down to one value
- Applies function to each element

```php
<?php
array_reduce(
  $array,
  function($result, $v) { return $result .= $v; }
);
```

`array('o', 'g', 'o')` becomes `ogo`

# MAP AND REDUCE

- Combined to great effect

```php
<?php
$a = array('o', 'g', 'o');
array_reduce(
  array_map(function($v) { return strtoupper(++$v); }, $a),
  function($result, $v) { return $result .= $v; }
);
// PHP
```

# RECURSION

- Just like the meaning of PHP
- A function that calls itself
  - Directly or
  - Indirectly
- Can be used for loops
- Loose or forgotten condition = blown stack

# HEADS OR TAILS

PHP = heads

```php
<?php
function head_sum($x) {
  return ($x == 1)
    ? $x
    : $x + head_sum($x - 1);
}
head_sum(10); // 55
```

Other languages have optimised tails

# HIGHER ORDER FUNCTIONS

- Functions that have other functions as
  - Call parameters
  - Return values
- Can be used to form expressions

# HIGHER EXAMPLE

```php
<?php
$data = array(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
function get_algorithm($rand_seed_fnc) {
   return (odd_even($rand_seed_fnc())) ?
      function($value) {
         return $value * $value;
      } :
      function($value) use ($rand_seed_fnc) {
         return ($value * $value / $rand_seed_fnc()) + 10;
      };
}
function odd_even($value) {
    return ($value % 2 === 0);
}
$rand_seed_fnc = function() { return rand(); };
$results = array_map(get_algorithm($rand_seed_fnc), $data);
```

# PARTIAL FUNCTION APPLICATION

Essentially set defaults

```php
<?php
$first_char = function($string) {
  return substr($string, 0, 1);
};
```

Works nicely with `array_map()`

```php
<?php
array_map($first_char, array('foo','bar','baz'));
// array('f','b','b')
```

# CURRYING

- Each parameter becomes a function
- More flexible PFA

```php
<?php
$first_char = function($start) {
  return function($length) use ($start) {
    return function($string) use ($start, $length) {
      return substr($string, $start, $length);
    };
  };
};
$a = $first_char(0);
$b = $a(1);
$b('foo'); // f
$c = $a(2);
$c('foo'); // fo
```

# MEMOIZATION

Function local cache

```php
<?php
function demo() {
  static $_c;
  if(is_null($_c)) {
    $_c = get_some_expensive_operation();
  }
  return $_c;
}
```

Handy in class methods too!

# WHEN TO USE FUNCTIONAL?

- Practice
- Innate understanding
- Turing complete methodology

# SO MUCH MORE...

- Function composition
- Tail recursion
- Function objects
- New language features
- Functors
- Applicatives
- Monads
- Event driven programming
- Null handling
- & more.

# EXIT()

- Functional PHP book
  - Follow @FunctionalPHP for tips
  - FunctionalPHP.com
- Simon Holywell
  - Follow @Treffynnon
  - SimonHolywell.com