loophp/fpt Documentation

Release 1.0.0

Pol Dellaiera

Dec 21, 2022

CONTENTS

Functional Programming Toolbox (**FPT** for the friends) is a set of stateless helper classes to facilitate the use of functional programming concepts.

CHAPTER

ONE

DOCUMENTATION

The API will give you a pretty good idea of the existing methods and what you can do with it.

I'm doing my best to keep the documentation up to date, if you found something odd, please let me know in the issue queue.

CHAPTER

TWO

CODE QUALITY, TESTS AND BENCHMARKS

Every time changes are introduced into the library, Github run the tests.

The library has tests written with PHPSpec. Feel free to check them out in the `spec` directory. Run `composer phpspec` to trigger the tests.

Before each commit some inspections are executed with GrumPHP, run `composer grumphp` to check manually.

The quality of the tests is tested with Infection a PHP Mutation testing framework, run `composer infection` to try it.

Static analysers are also controlling the code. PHPStan and PSalm are enabled to their maximum level.

CHAPTER

THREE

CONTRIBUTING

Feel free to contribute by sending Github pull requests. I'm quite reactive :-)

If you can't contribute to the code, you can also sponsor me on Github or Paypal.

CHAPTER

FOUR

CHANGELOG

See CHANGELOG.md for a changelog based on git commits.

For more detailed changelogs, please check the release changelogs.

4.1 Requirements

4.1.1 PHP

PHP greater than 7.4 is required.

4.1.2 Dependencies

No dependency is required.

4.2 Installation

The easiest way to install it is through Composer

```
composer require loophp/fpt
```

4.3 Usage

TODO

4.4 API

4.4.1 Static constructors

compose

Performs right-to-left function composition. The last argument may have any arity; the remaining arguments must be unary.

Warning: The result of *compose* is not automatically curried.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

// phpcs:disable Generic.Files.LineLength.TooLong

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;

$closure = static fn (string $first, string $second): string => sprintf('My cats names
↪are %s and %s.', $first, $second);

$composedClosure = FPT::compose()('strtoupper', $closure);

$composedClosure('Izumi', 'Nakano'); // "MY CATS NAMES ARE IZUMI AND NAKANO."
```

curryLeft

Returns a curried equivalent of the provided function.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;

$curryLeft = FPT::curryLeft()('explode');

[$firstName, $lastName] = $curryLeft(' ')('James Bond');

$curriedLeft = FPT::curryLeft()('explode', 3);

[$evil, $good] = $curriedLeft(',')('Jaws,James,Bond')(2);
```

curryRight

Returns a curried equivalent of the provided function.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;

$curryRight = FPT::curryRight()('explode');

[$firstName, $lastName] = $curryRight('James Bond')(' ');

$curryRight = FPT::curryRight()('explode', 3);

[$evil, $good] = $curryRight(2)('Jaws,James,Bond')(',');
```

filter

flip

Returns a new function much like the supplied one, except that the first two arguments' order is reversed.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;

$f = static fn (string ...$x): string => implode('', $x);

$flip = FPT::flip()($f)('a', 'b', 'c'); // bac
```

fold

Returns a single item by iterating through the list, successively calling the reducer function and passing it an accumulator value and the current value from the array, and then passing the result to the next call.

The iterator function receives two values: `$acc` and `$value`.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

include __DIR__ . '/../../../vendor/autoload.php';

use loophp\fpt\FPT;

$reducer = static fn (callable $a) => static fn (callable $b) => static fn (...$xs) =>
↪$a($b(...$xs));
$accumulator = static fn ($v): string => sprintf('[%s]', $v);
$callable = ['strtoupper', static fn ($s): string => sprintf('%s%s', $s, $s)];

FPT::fold()($reducer)($accumulator)(...$callable)('hello'); // [HELLOHELLO]
```

identity

A function that does nothing but return the parameter supplied to it.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;

FPT::identity()('foo'); // foo
```

map

Takes a function and an iterable, applies the function to each of the iterable values, and yield the result.

nary

Wraps a function of any arity in a function that accepts exactly n parameters. Any extraneous parameters will not be passed to the supplied function.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;

$closure = static fn (...$args): string => implode(';', $args);

$newClosure = FPT::nary()(1)($closure);
$newClosure('a', 'b'); // a

$newClosure = FPT::nary()(2)($closure);
$newClosure('a', 'b'); // a;b
```

not

Wraps a function in a function that returns the ! of the original function return value.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;

$closure = static fn (string $left, string $right): bool => $left === $right;

$closure('a', 'b'); // false
$closure('a', 'a'); // true
```

(continues on next page)

(continued from previous page)

```php
$notClosure = FPT::not()($closure);
$notClosure('a', 'b'); // true

$notClosure = FPT::not()($closure);
$notClosure('a', 'a'); // false
```

operator

Apply an operator on two arguments.

This method is curried and take first the operator, then its left and right members.

Available operators are constants in Operator class.

```php
<?php

/**
 * For the full copyright and license information, please view
 * the LICENSE file that was distributed with this source code.
 */

declare(strict_types=1);

namespace Example;

use loophp\fpt\FPT;
use loophp\fpt\Operator;

FPT::operator()(Operator::OP_PLUS)(40)(2); // 42
```

partialLeft

partialRight

reduce

thunk

uncurry

4.5 Tests, code quality and code style

Every time changes are introduced into the library, Github Actions run the tests.

Tests are written with PHPSpec and you can find the coverage percentage on a badge on the README file.

PHPInfection is also triggered used to ensure that your code is properly tested.

The code style is based on PSR-12 plus a set of custom rules. Find more about the code style in use in the package drupol/php-conventions.

A PHP quality tool, Grumphp, is used to orchestrate all these tasks at each commit on the local machine, but also on the continuous integration tools.

To run the whole tests tasks locally, do

```
composer grumphp
```

or

```
./vendor/bin/grumphp run
```

Here's an example of output that shows all the tasks that are setup in Grumphp and that will check your code

```
$ ./vendor/bin/grumphp run
GrumPHP is sniffing your code!
Running task  1/14: SecurityChecker... ✓
Running task  2/14: Composer... ✓
Running task  3/14: ComposerNormalize... ✓
Running task  4/14: YamlLint... ✓
Running task  5/14: JsonLint... ✓
Running task  6/14: PhpLint... ✓
Running task  7/14: TwigCs... ✓
Running task  8/14: PhpCsAutoFixerV2... ✓
Running task  9/14: PhpCsFixerV2... ✓
Running task 10/14: Phpcs... ✓
Running task 11/14: Psalm... ✓
Running task 12/14: PhpStan... ✓
Running task 13/14: Phpspec... ✓
Running task 14/14: Infection... ✓
$
```

4.6 Contributing

See the file CONTRIBUTING.md but feel free to contribute to this library by sending Github pull requests.