
Phamda Documentation

Release

Mikael Pajunen

May 19, 2016

1	Introduction	3
2	Examples	5
3	Phamda functions	9

Phamda is a functional programming library for PHP. **Automatically curried** functions are the main feature.

Introduction

Phamda is a functional programming library for PHP. The main features are:

- A set mostly familiar functions, including basic ones like *filter*, *map* and *reduce*.
- Almost all of the functions are **automatically curried**. Calling a function with fewer parameters than are expected returns a new function.
- **Placeholder arguments** can be used with all the curried functions.
- The functions are designed to be **composable**. Specific functions like *compose* and *pipe* enable different composition patterns.

1.1 Requirements

- PHP 5.6+ or HHVM

1.2 Installation

Phamda can be installed easily in any project using Composer:

```
composer require phamda/phamda
```

Examples

These examples highlight the major features of Phamda. Basic usage examples can also be found on the [function list](#).

2.1 Currying

Nearly all of the functions use automatic partial application or **currying**. This means that you can call the *filter* function with only the predicate callback and get a new function:

```
use Phamda\Phamda as P;

$isPositive = function ($x) { return $x > 0; };
$list       = [5, 7, -3, 19, 0, 2];
$getPositives = P::filter($isPositive);
$result      = $getPositives($list); // => [5, 7, 19, 2]
```

The final result is the same as using two arguments directly. Of course this new function could now be used to filter other lists as well.

It's also possible to create new curried functions, including from native PHP functions. The *curry* function takes a function and initial parameters and returns a new function:

```
$replaceBad = P::curry('str_replace', 'bad', 'good');
$dayResult  = $replaceBad('bad day'); // => 'good day'
$notResult  = $replaceBad('not bad'); // => 'not good'
```

2.2 Composition

Phamda functions are **composable**. The basic functions can be used to create new, more complex functions. There are also several functions to help with function composition. For example the *compose* function that takes multiple argument functions and returns a new function. Calling this new function applies the argument functions in succession:

```
$double      = function ($x) { return $x * 2; };
$addFive     = function ($x) { return $x + 5; };
$addFiveAndDouble = P::compose($double, $addFive);
$result      = $addFiveAndDouble(16); // => 42
// Equivalent to calling $double($addFive(16));
```

2.3 Placeholders

Phamda also supports **placeholder arguments**. A placeholder can be created by calling the `_` (underscore) function. A placeholder can be used with any curried function, for example:

```
$_ = P::_();
$subtractTen = P::subtract($_, 10);
$result      = $subtractTen(22); // => 12
```

Placeholders also work with manually curried functions:

```
$slashCount = P::curry('substr_count', P::_(), '/');
$result     = $slashCount('ab/c/de//f/'); // => 5
```

2.4 Pipelines

Combining these techniques allows the building of function pipelines. In this example they are applied to processing a list of badly formatted product data using the *pipe* function. It's similar to *compose* but the argument functions are applied in reverse order:

```
$products = [
  ['category' => 'QDT', 'weight' => 65.8, 'price' => 293.5, 'number' => 15708],
  ['number' => 59391, 'price' => 366.64, 'category' => 'NVG', 'weight' => 15.5],
  ['category' => 'AWK', 'number' => 89634, 'price' => 341.92, 'weight' => 35],
  ['price' => 271.8, 'weight' => 5.3, 'number' => 38718, 'category' => 'ETW'],
  ['price' => 523.63, 'weight' => 67.9, 'number' => 75905, 'category' => 'YVM'],
  ['price' => 650.31, 'weight' => 3.9, 'category' => 'XPA', 'number' => 46289],
  ['category' => 'WGX', 'weight' => 75.5, 'number' => 26213, 'price' => 471.44],
  ['category' => 'KCF', 'price' => 581.85, 'weight' => 31.9, 'number' => 48160],
];

$formatPrice = P::curry('number_format', P::_(), 2);
$process     = P::pipe(
  P::filter( // Only include products that...
    P::pipe(
      P::prop('weight'), // ... weigh...
      P::lt(P::_(), 50.0) // ... less than 50.0.
    )
  ),
  P::map( // For each product...
    P::pipe(
      // ... drop the weight field and fix field order:
      P::pick(['number', 'category', 'price']),
      // ... and format the price:
      P::evolve(['price' => $formatPrice])
    )
  ),
  P::sortBy( // Sort the products by...
    P::prop('number') // ... comparing product numbers.
  )
);

$result = $process($products);
/* =>
[
  ['number' => 38718, 'category' => 'ETW', 'price' => '271.80'],
```

```
['number' => 46289, 'category' => 'XPA', 'price' => '650.31'],  
['number' => 48160, 'category' => 'KCF', 'price' => '581.85'],  
['number' => 59391, 'category' => 'NVG', 'price' => '366.64'],  
['number' => 89634, 'category' => 'AWK', 'price' => '341.92'],  
]  
*/
```

Phamda functions

Currently included functions (106):

3.1 `_`

Placeholder `P::_()`

Returns a placeholder to be used with curried functions.

```
$sub10 = P::subtract(P::_(), 10);  
$sub10(52); // => 42
```

3.2 `add`

`int|float P::add(int|float $x, int|float $y)`

Adds two numbers.

```
P::add(15, 27); // => 42  
P::add(36, -8); // => 28
```

3.3 `all`

`bool P::all(callable $predicate, array|\Traversable $collection)`

Returns `true` if all elements of the collection match the predicate, `false` otherwise.

```
$isPositive = function ($x) { return $x > 0; };  
P::all($isPositive, [1, 2, 0, -5]); // => false  
P::all($isPositive, [1, 2, 1, 11]); // => true
```

3.4 `allPass`

`callable P::allPass(callable[] $predicates)`

Creates a single predicate from a list of predicates that returns `true` when all the predicates match, `false` otherwise.

```
$isEven = function ($x) { return $x % 2 === 0; };
$isPositive = function ($x) { return $x > 0; };
$isEvenAndPositive = P::allPass([$isEven, $isPositive]);
$isEvenAndPositive(5); // => false
$isEvenAndPositive(-4); // => false
$isEvenAndPositive(6); // => true
```

3.5 always

callable P::always(mixed \$value)

Returns a function that always returns the passed value.

```
$alwaysFoo = P::always('foo');
$alwaysFoo(); // => 'foo'
```

3.6 any

bool P::any(callable \$predicate, array|\Traversable \$collection)

Returns true if any element of the collection matches the predicate, false otherwise.

```
$isPositive = function ($x) { return $x > 0; };
P::any($isPositive, [1, 2, 0, -5]); // => true
P::any($isPositive, [-3, -7, -1, -5]); // => false
```

3.7 anyPass

callable P::anyPass(callable[] \$predicates)

Creates a single predicate from a list of predicates that returns true when any of the predicates matches, false otherwise.

```
$isEven = function ($x) { return $x % 2 === 0; };
$isPositive = function ($x) { return $x > 0; };
$isEvenOrPositive = P::anyPass([$isEven, $isPositive]);
$isEvenOrPositive(5); // => true
$isEvenOrPositive(-4); // => true
$isEvenOrPositive(-3); // => false
```

3.8 append

array|Collection P::append(mixed \$item, array|Collection \$collection)

Return a new collection that contains all the items in the given collection and the given item last.

```
P::append('c', ['a', 'b']); // => ['a', 'b', 'c']
P::append('c', []); // => ['c']
P::append(['d', 'e'], ['a', 'b']); // => ['a', 'b', ['d', 'e']]
```

3.9 apply

mixed P::apply(callable \$function, array \$arguments)

Calls the function using the values of the given arguments list as positional arguments.

Effectively creates an unary function from a variadic function.

```
$concat3 = function ($a, $b, $c) { return $a . $b . $c; };
P::apply($concat3, ['foo', 'ba', 'rba']); // => 'foobarba'
```

3.10 assoc

array|object P::assoc(string \$property, mixed \$value, array|object \$object)

Returns a new array or object, setting the given value to the specified property.

```
P::assoc('bar', 3, ['foo' => 1]); // => ['foo' => 1, 'bar' => 3]
P::assoc('bar', 3, ['foo' => 1, 'bar' => 2]); // => ['foo' => 1, 'bar' => 3]
P::assoc('foo', null, ['foo' => 15, 'bar' => 7]); // => ['foo' => null, 'bar' => 7]
```

3.11 assocPath

array|object P::assocPath(array \$path, mixed \$value, array|object \$object)

Returns a new array or object, setting the given value to the property specified by the path.

```
P::assocPath(['bar'], 3, ['foo' => 1, 'bar' => 2]); // => ['foo' => 1, 'bar' => 3]
P::assocPath(['bar', 'baz'], 4, ['foo' => 1, 'bar' => []]); // => ['foo' => 1, 'bar' => ['baz' => 4]]
```

3.12 binary

callable P::binary(callable \$function)

Wraps the given function in a function that accepts exactly two parameters.

```
$add3 = function ($a = 0, $b = 0, $c = 0) { return $a + $b + $c; };
$add2 = P::binary($add3);
$add2(27, 15, 33); // => 42
```

3.13 both

callable P::both(callable \$a, callable \$b)

Returns a function that returns true when both of the predicates match, false otherwise.

```
$lt = function ($x, $y) { return $x < $y; };
$arePositive = function ($x, $y) { return $x > 0 && $y > 0; };
$test = P::both($lt, $arePositive);
$test(9, 4); // => false
```

```
$test(-3, 11); // => false
$test(5, 17); // => true
```

3.14 cast

```
mixed P::cast(string $type, mixed $value)
```

Return the given value cast to the given type.

```
P::cast('string', 3); // => '3'
P::cast('int', 4.55); // => 4
```

3.15 clone

```
object P::clone_(object $object)
```

Clones an object.

3.16 comparator

```
callable P::comparator(callable $predicate)
```

Creates a comparator function from a function that returns whether the first argument is less than the second.

```
$lt = function ($x, $y) { return $x < $y; };
$compare = P::comparator($lt);
$compare(5, 6); // => -1
$compare(6, 5); // => 1
$compare(5, 5); // => 0
```

3.17 compose

```
callable P::compose(callable ...$functions)
```

Returns a new function that calls each supplied function in turn in reverse order and passes the result as a parameter to the next function.

```
$add5 = function ($x) { return $x + 5; };
$square = function ($x) { return $x ** 2; };
$addToSquared = P::compose($add5, $square);
$addToSquared(4); // => 21
$hello = function ($target) { return 'Hello ' . $target; };
$helloUpper = P::compose($hello, 'strtoupper');
$upperHello = P::compose('strtoupper', $hello);
$helloUpper('world'); // => 'Hello WORLD'
$upperHello('world'); // => 'HELLO WORLD'
```


3.18 concat

```
string P::concat(string $a, string $b)
```

Returns a string concatenated of a and b.

```
P::concat('ab', 'cd'); // => 'abcd'
P::concat('abc', ''); // => 'abc'
```

3.19 construct

```
object P::construct(string $class, mixed ...$initialArguments)
```

Wraps the constructor of the given class to a function.

```
$date = P::construct(\DateTime::class, '2015-03-15');
$date->format('Y-m-d'); // => '2015-03-15'
```

3.20 constructN

```
object P::constructN(int $arity, string $class, mixed ...$initialArguments)
```

Wraps the constructor of the given class to a function of specified arity.

```
$construct = P::constructN(1, \DateTime::class);
$construct('2015-03-15')->format('Y-m-d'); // => '2015-03-15'
```

3.21 contains

```
bool P::contains(mixed $value, array|\Traversable $collection)
```

Returns true if the specified item is found in the collection, false otherwise.

```
P::contains('a', ['a', 'b', 'c', 'e']); // => true
P::contains('d', ['a', 'b', 'c', 'e']); // => false
```

3.22 curry

```
callable|mixed P::curry(callable $function, mixed ...$initialArguments)
```

Wraps the given function to a function that returns a new function until all required parameters are given.

```
$add = function ($x, $y, $z) { return $x + $y + $z; };
$addHundred = P::curry($add, 100);
$addHundred(20, 3); // => 123
```

3.23 curryN

```
callable|mixed P::curryN(int $length, callable $function, mixed ...$initialArguments)
```

Wraps the given function to a function of specified arity that returns a new function until all required parameters are given.

```
$add = function ($x, $y, $z = 0) { return $x + $y + $z; };
$addTen = P::curryN(3, $add, 10);
$addTen(10, 3); // => 23
$addTwenty = $addTen(10);
$addTwenty(5); // => 25
```

3.24 dec

```
int|float P::dec(int|float $number)
```

Decrements the given number.

```
P::dec(43); // => 42
P::dec(-14); // => -15
```

3.25 defaultTo

```
mixed P::defaultTo(mixed $default, mixed $value)
```

Returns the default argument if the value argument is null.

```
P::defaultTo(22, 15); // => 15
P::defaultTo(42, null); // => 42
P::defaultTo(15, false); // => false
```

3.26 divide

```
int|float P::divide(int|float $x, int|float $y)
```

Divides two numbers.

```
P::divide(55, 11); // => 5
P::divide(48, -8); // => -6
```

3.27 each

```
array|\Traversable|Collection P::each(callable $function,
array|\Traversable|Collection $collection)
```

Calls the given function for each element in the collection and returns the original collection.

The supplied function receives three arguments: item, index, collection.

```
$date = new \DateTime('2015-02-02');
$addCalendar = function ($number, $type) use ($date) { $date->modify("+$number $type"); };
P::each($addCalendar, ['months' => 3, 'weeks' => 6, 'days' => 2]);
$date->format('Y-m-d'); // => '2015-06-15'
```

3.28 either

callable P::either(callable \$a, callable \$b)

Returns a function that returns true when either of the predicates matches, false otherwise.

```
$lt = function ($x, $y) { return $x < $y; };
$arePositive = function ($x, $y) { return $x > 0 && $y > 0; };
$test = P::either($lt, $arePositive);
$test(-5, -16); // => false
$test(-3, 11); // => true
$test(17, 3); // => true
```

3.29 eq

bool P::eq(mixed \$x, mixed \$y)

Return true when the arguments are strictly equal.

```
P::eq('a', 'a'); // => true
P::eq('a', 'b'); // => false
P::eq(null, null); // => true
```

3.30 evolve

array|object P::evolve(callable[] \$transformations, array|object|\ArrayAccess \$object)

Returns a new object or array containing all the fields of the original object, using given transformations.

```
$object = ['foo' => 'bar', 'fiz' => 'buz'];
P::evolve(['foo' => 'strtoupper'], $object); // => ['foo' => 'BAR', 'fiz' => 'buz']
```

3.31 explode

string[] P::explode(string \$delimiter, string \$string)

Returns an array containing the parts of a string split by the given delimiter.

If the delimiter is an empty string, returns a char array.

```
P::explode('/', 'f/o/o'); // => ['f', 'o', 'o']
P::explode('', 'b/a/z'); // => ['b', '/', 'a', '/', 'z']
P::explode('.', ''); // => ['']
```

3.32 false

callable P::false()

Returns a function that always returns false.

```
$false = P::false();  
$false(); // => false
```

3.33 filter

array|Collection P::filter(callable \$predicate, array|\Traversable|Collection \$collection)

Returns a new collection containing the items that match the given predicate.

The supplied predicate receives three arguments: item, index, collection.

```
$gt2 = function ($x) { return $x > 2; };  
P::filter($gt2, ['foo' => 2, 'bar' => 3, 'baz' => 4]); // => ['bar' => 3, 'baz' => 4]
```

3.34 find

mixed|null P::find(callable \$predicate, array|\Traversable \$collection)

Returns the first item of a collection for which the given predicate matches, or null if no match is found.

```
$isPositive = function ($x) { return $x > 0; };  
P::find($isPositive, [-5, 0, 15, 33, -2]); // => 15
```

3.35 findIndex

int|string|null P::findIndex(callable \$predicate, array|\Traversable \$collection)

Returns the index of the first item of a collection for which the given predicate matches, or null if no match is found.

```
$isPositive = function ($x) { return $x > 0; };  
P::findIndex($isPositive, [-5, 0, 15, 33, -2]); // => 2
```

3.36 findLast

mixed|null P::findLast(callable \$predicate, array|\Traversable \$collection)

Returns the last item of a collection for which the given predicate matches, or null if no match is found.

```
$isPositive = function ($x) { return $x > 0; };  
P::findLast($isPositive, [-5, 0, 15, 33, -2]); // => 33
```

3.37 findLastIndex

```
int|string|null P::findLastIndex(callable $predicate, array|\Traversable
$collection)
```

Returns the index of the last item of a collection for which the given predicate matches, or null if no match is found.

```
$isPositive = function ($x) { return $x > 0; };
P::findLastIndex($isPositive, [-5, 0, 15, 33, -2]); // => 3
```

3.38 first

```
mixed P::first(array|\Traversable|Collection $collection)
```

Returns the first item of a collection, or false if the collection is empty.

```
P::first([5, 8, 9, 13]); // => 5
P::first([]); // => false
```

3.39 flatMap

```
array P::flatMap(callable $function, array $list)
```

Returns a list containing the flattened items created by applying the function to each item of the list.

```
$split = P::unary('str_split');
P::flatMap($split, ['abc', 'de']); // => ['a', 'b', 'c', 'd', 'e']
$getNeighbors = function ($x) { return [$x - 1, $x, $x + 1]; };
P::flatMap($getNeighbors, [1, 2, 3]); // => [0, 1, 2, 1, 2, 3, 2, 3, 4]
```

3.40 flatten

```
array P::flatten(array $list)
```

Returns an array that contains all the items on the list, with all arrays flattened.

```
P::flatten([1, [2, 3], [4]]); // => [1, 2, 3, 4]
P::flatten([1, [2, [3]], [[4]]]); // => [1, 2, 3, 4]
```

3.41 flattenLevel

```
array P::flattenLevel(array $list)
```

Returns an array that contains all the items on the list, with arrays on the first nesting level flattened.

```
P::flattenLevel([1, [2, 3], [4]]); // => [1, 2, 3, 4]
P::flattenLevel([1, [2, [3]], [[4]]]); // => [1, 2, [3], [4]]
```

3.42 flip

callable P::flip(callable \$function)

Wraps the given function and returns a new function for which the order of the first two parameters is reversed.

```
$sub = function ($x, $y) { return $x - $y; };
$flippedSub = P::flip($sub);
$flippedSub(20, 30); // => 10
```

3.43 fromPairs

array|Collection P::fromPairs(array|\Traversable|Collection \$list = null)

Creates a new map from a list of key-value pairs.

```
P::fromPairs([[ 'a', 'b'], [ 'c', 'd']]); // => [ 'a' => 'b', 'c' => 'd']
P::fromPairs([[3, 'b'], [5, null]]); // => [3 => 'b', 5 => null]
```

3.44 groupBy

array[]|Collection[] P::groupBy(callable \$function, array|\Traversable|Collection \$collection)

Returns an array of sub collections based on a function that returns the group keys for each item.

```
$firstChar = function ($string) { return $string[0]; };
$collection = ['abc', 'cbc', 'cab', 'baa', 'ayb'];
P::groupBy($firstChar, $collection); // => [ 'a' => [0 => 'abc', 4 => 'ayb'], 'c' => [1 => 'cbc', 2 => 'cab'], 'b' => [3 => 'baa'] ]
```

3.45 gt

bool P::gt(mixed \$x, mixed \$y)

Returns true if the first parameter is greater than the second, false otherwise.

```
P::gt(1, 2); // => false
P::gt(1, 1); // => false
P::gt(2, 1); // => true
```

3.46 gte

bool P::gte(mixed \$x, mixed \$y)

Returns true if the first parameter is greater than or equal to the second, false otherwise.

```
P::gte(1, 2); // => false
P::gte(1, 1); // => true
P::gte(2, 1); // => true
```

3.47 identity

```
mixed P::identity(mixed $x)
```

Returns the given parameter.

```
P::identity(1); // => 1
P::identity(null); // => null
P::identity('abc'); // => 'abc'
```

3.48 ifElse

```
callable P::ifElse(callable $condition, callable $onTrue, callable $onFalse)
```

Returns a function that applies either the `onTrue` or the `onFalse` function, depending on the result of the condition predicate.

```
$addOrSub = P::ifElse(P::lt(0), P::add(-10), P::add(10));
$addOrSub(25); // => 15
$addOrSub(-3); // => 7
```

3.49 implode

```
string P::implode(string $glue, string[] $strings)
```

Returns a string formed by combining a list of strings using the given glue string.

```
P::implode('/', ['f', 'o', 'o']); // => 'f/o/o'
P::implode('.', ['a', 'b', 'cd', '']); // => 'a.b.cd.'
P::implode('.', ['']); // => ''
```

3.50 inc

```
int|float P::inc(int|float $number)
```

Increments the given number.

```
P::inc(41); // => 42
P::inc(-16); // => -15
```

3.51 indexOf

```
int|string|false P::indexOf(mixed $item, array|\Traversable $collection)
```

Returns the index of the given item in a collection, or `false` if the item is not found.

```
P::indexOf(16, [1, 6, 44, 16, 52]); // => 3
P::indexOf(15, [1, 6, 44, 16, 52]); // => false
```

3.52 invoker

callable P::invoker(int \$arity, string \$method, mixed ...\$initialArguments)

Returns a function that calls the specified method of a given object.

```
$addDay = P::invoker(1, 'add', new \DateInterval('P1D'));
$addDay(new \DateTime('2015-03-15'))->format('Y-m-d'); // => '2015-03-16'
$addDay(new \DateTime('2015-03-12'))->format('Y-m-d'); // => '2015-03-13'
```

3.53 isEmpty

bool P::isEmpty(array|\Traversable|Collection \$collection)

Returns true if a collection has no elements, false otherwise.

```
P::isEmpty([1, 2, 3]); // => false
P::isEmpty([0]); // => false
P::isEmpty([]); // => true
```

3.54 isInstance

bool P::isInstance(string \$class, object \$object)

Return true if an object is of the specified class, false otherwise.

```
$isDate = P::isInstance(\DateTime::class);
$isDate(new \DateTime()); // => true
$isDate(new \DateTimeImmutable()); // => false
```

3.55 last

mixed P::last(array|\Traversable|Collection \$collection)

Returns the last item of a collection, or false if the collection is empty.

```
P::last([5, 8, 9, 13]); // => 13
P::last([]); // => false
```

3.56 lt

bool P::lt(mixed \$x, mixed \$y)

Returns true if the first parameter is less than the second, false otherwise.

```
P::lt(1, 2); // => true
P::lt(1, 1); // => false
P::lt(2, 1); // => false
```


3.57 lte

```
bool P::lte(mixed $x, mixed $y)
```

Returns true if the first parameter is less than or equal to the second, false otherwise.

```
P::lte(1, 2); // => true
P::lte(1, 1); // => true
P::lte(2, 1); // => false
```

3.58 map

```
array|Collection P::map(callable $function, array|\Traversable|Collection
$collection)
```

Returns a new collection where values are created from the original collection by calling the supplied function.

The supplied function receives three arguments: item, index, collection.

```
$square = function ($x) { return $x ** 2; };
P::map($square, [1, 2, 3, 4]); // => [1, 4, 9, 16]
$keyExp = function ($value, $key) { return $value ** $key; };
P::map($keyExp, [1, 2, 3, 4]); // => [1, 2, 9, 64]
```

3.59 max

```
mixed P::max(array|\Traversable $collection)
```

Returns the largest value in the collection.

```
P::max([6, 15, 8, 9, -2, -3]); // => 15
P::max(['bar', 'foo', 'baz']); // => 'foo'
```

3.60 maxBy

```
mixed P::maxBy(callable $getValue, array|\Traversable $collection)
```

Returns the item from a collection for which the supplied function returns the largest value.

```
$getFoo = function ($item) { return $item->foo; };
$a = (object) ['baz' => 3, 'bar' => 16, 'foo' => 5];
$b = (object) ['baz' => 1, 'bar' => 25, 'foo' => 8];
$c = (object) ['baz' => 14, 'bar' => 20, 'foo' => -2];
P::maxBy($getFoo, [$a, $b, $c]); // => $b
```

3.61 merge

```
array P::merge(array $a, array $b)
```

Returns an array that contains all the values in arrays a and b.

```
P::merge([1, 2], [3, 4, 5]); // => [1, 2, 3, 4, 5]
P::merge(['a', 'b'], ['a', 'b']); // => ['a', 'b', 'a', 'b']
```

3.62 min

mixed P::min(array|\Traversable \$collection)

Returns the smallest value in the collection.

```
P::min([6, 15, 8, 9, -2, -3]); // => -3
P::min(['bar', 'foo', 'baz']); // => 'bar'
```

3.63 minBy

mixed P::minBy(callable \$getValue, array|\Traversable \$collection)

Returns the item from a collection for which the supplied function returns the smallest value.

```
$getFoo = function ($item) { return $item->foo; };
$a = (object) ['baz' => 3, 'bar' => 16, 'foo' => 5];
$b = (object) ['baz' => 1, 'bar' => 25, 'foo' => 8];
$c = (object) ['baz' => 14, 'bar' => 20, 'foo' => -2];
P::minBy($getFoo, [$a, $b, $c]); // => $c
```

3.64 modulo

int P::modulo(int \$x, int \$y)

Divides two integers and returns the modulo.

```
P::modulo(15, 6); // => 3
P::modulo(22, 11); // => 0
P::modulo(-23, 6); // => -5
```

3.65 multiply

int|float P::multiply(int|float \$x, int|float \$y)

Multiplies two numbers.

```
P::multiply(15, 27); // => 405
P::multiply(36, -8); // => -288
```

3.66 nAry

callable P::nAry(int \$arity, callable \$function)

Wraps the given function in a function that accepts exactly the given amount of parameters.

```
$add3 = function ($a = 0, $b = 0, $c = 0) { return $a + $b + $c; };
$add2 = P::nAry(2, $add3);
$add2(27, 15, 33); // => 42
$add1 = P::nAry(1, $add3);
$add1(27, 15, 33); // => 27
```

3.67 negate

int|float P::negate(int|float \$x)

Returns the negation of a number.

```
P::negate(15); // => -15
P::negate(-0.7); // => 0.7
P::negate(0); // => 0
```

3.68 none

bool P::none(callable \$predicate, array|\Traversable \$collection)

Returns true if no element in the collection matches the predicate, false otherwise.

```
$isPositive = function ($x) { return $x > 0; };
P::none($isPositive, [1, 2, 0, -5]); // => false
P::none($isPositive, [-3, -7, -1, -5]); // => true
```

3.69 not

callable P::not(callable \$predicate)

Wraps a predicate and returns a function that return true if the wrapped function returns a falsey value, false otherwise.

```
$equal = function ($a, $b) { return $a === $b; };
$notEqual = P::not($equal);
$notEqual(15, 13); // => true
$notEqual(7, 7); // => false
```

3.70 partial

callable P::partial(callable \$function, mixed ...\$initialArguments)

Wraps the given function and returns a new function that can be called with the remaining parameters.

```
$add = function ($x, $y, $z) { return $x + $y + $z; };
$addTen = P::partial($add, 10);
$addTen(3, 4); // => 17
$addTwenty = P::partial($add, 2, 3, 15);
$addTwenty(); // => 20
```

3.71 partialN

```
callable P::partialN(int $arity, callable $function, mixed
...$initialArguments)
```

Wraps the given function and returns a new function of fixed arity that can be called with the remaining parameters.

```
$add = function ($x, $y, $z = 0) { return $x + $y + $z; };
$addTen = P::partialN(3, $add, 10);
$addTwenty = $addTen(10);
$addTwenty(5); // => 25
```

3.72 partition

```
array[]|Collection[] P::partition(callable $predicate, array|\Traversable|Collection
$collection)
```

Returns the items of the original collection divided into two collections based on a predicate function.

```
$isPositive = function ($x) { return $x > 0; };
P::partition($isPositive, [4, -16, 7, -3, 2, 88]); // => [[0 => 4, 2 => 7, 4 => 2, 5 => 88], [1 => -
```

3.73 path

```
mixed P::path(array $path, array|object $object)
```

Returns a value found at the given path.

```
P::path(['foo', 'bar'], ['foo' => ['baz' => 26, 'bar' => 15]]); // => 15
P::path(['bar', 'baz'], ['bar' => ['baz' => null, 'foo' => 15]]); // => null
```

3.74 pathEq

```
bool P::pathEq(array $path, mixed $value, array|object $object)
```

Returns true if the given value is found at the specified path, false otherwise.

```
P::pathEq(['foo', 'bar'], 44, ['foo' => ['baz' => 26, 'bar' => 15]]); // => false
P::pathEq(['foo', 'baz'], 26, ['foo' => ['baz' => 26, 'bar' => 15]]); // => true
```

3.75 pick

```
array P::pick(array $names, array $item)
```

Returns a new array, containing only the values that have keys matching the given list.

```
P::pick(['bar', 'fib'], ['foo' => null, 'bar' => 'bzz', 'baz' => 'bob']); // => ['bar' => 'bzz']
P::pick(['fob', 'fib'], ['foo' => null, 'bar' => 'bzz', 'baz' => 'bob']); // => []
P::pick(['bar', 'foo'], ['foo' => null, 'bar' => 'bzz', 'baz' => 'bob']); // => ['bar' => 'bzz', 'fo
```

3.76 pickAll

```
array P::pickAll(array $names, array $item)
```

Returns a new array, containing the values that have keys matching the given list, including keys that are not found in the item.

```
P::pickAll(['bar', 'fib'], ['foo' => null, 'bar' => 'bzz', 'baz' => 'bob']); // => ['bar' => 'bzz', 'fib' => null]
P::pickAll(['fob', 'fib'], ['foo' => null, 'bar' => 'bzz', 'baz' => 'bob']); // => ['fob' => null, 'fib' => null]
P::pickAll(['bar', 'foo'], ['foo' => null, 'bar' => 'bzz', 'baz' => 'bob']); // => ['bar' => 'bzz', 'foo' => null]
```

3.77 pipe

```
callable P::pipe(callable ...$functions)
```

Returns a new function that calls each supplied function in turn and passes the result as a parameter to the next function.

```
$add5 = function ($x) { return $x + 5; };
$square = function ($x) { return $x ** 2; };
$squareAdded = P::pipe($add5, $square);
$squareAdded(4); // => 81
$hello = function ($target) { return 'Hello ' . $target; };
$helloUpper = P::pipe('strtoupper', $hello);
$upperHello = P::pipe($hello, 'strtoupper');
$helloUpper('world'); // => 'Hello WORLD'
$upperHello('world'); // => 'HELLO WORLD'
```

3.78 pluck

```
array|Collection P::pluck(string $name, array|\Traversable|Collection $collection)
```

Returns a new collection, where the items are single properties plucked from the given collection.

```
P::pluck('foo', [['foo' => null, 'bar' => 'bzz', 'baz' => 'bob'], ['foo' => 'fii', 'baz' => 'pob']]);
P::pluck('baz', [['foo' => null, 'bar' => 'bzz', 'baz' => 'bob'], ['foo' => 'fii', 'baz' => 'pob']]);
```

3.79 prepend

```
array|Collection P::prepend(mixed $item, array|Collection $collection)
```

Return a new collection that contains the given item first and all the items in the given collection.

```
P::prepend('c', ['a', 'b']); // => ['c', 'a', 'b']
P::prepend('c', []); // => ['c']
P::prepend(['d', 'e'], ['a', 'b']); // => [['d', 'e'], 'a', 'b']
```

3.80 product

```
int|float P::product(int[]|float[] $values)
```

Multiplies a list of numbers.

```
P::product([11, -8, 3]); // => -264
P::product([1, 2, 3, 4, 5, 6]); // => 720
```

3.81 prop

```
mixed P::prop(string $name, array|object|\ArrayAccess $object)
```

Returns the given element of an array or property of an object.

```
P::prop('bar', ['bar' => 'fuz', 'baz' => null]); // => 'fuz'
P::prop('baz', ['bar' => 'fuz', 'baz' => null]); // => null
```

3.82 propEq

```
bool P::propEq(string $name, mixed $value, array|object $object)
```

Returns true if the specified property has the given value, false otherwise.

```
P::propEq('foo', 'bar', ['foo' => 'bar']); // => true
P::propEq('foo', 'baz', ['foo' => 'bar']); // => false
```

3.83 reduce

```
mixed P::reduce(callable $function, mixed $initial, array|\Traversable
$collection)
```

Returns a value accumulated by calling the given function for each element of the collection.

The supplied function receives four arguments: previousValue, item, index, collection.

```
$concat = function ($x, $y) { return $x . $y; };
P::reduce($concat, 'foo', ['bar', 'baz']); // => 'foobarbaz'
```

3.84 reduceRight

```
mixed P::reduceRight(callable $function, mixed $initial, array|\Traversable
$collection)
```

Returns a value accumulated by calling the given function for each element of the collection in reverse order.

The supplied function receives four arguments: previousValue, item, index, collection.

```
$concat = function ($accumulator, $value, $key) { return $accumulator . $key . $value; };
P::reduceRight($concat, 'no', ['foo' => 'bar', 'fiz' => 'buz']); // => 'nofizbuzfoobar'
```

3.85 reject

```
array|Collection P::reject(callable $predicate, array|\Traversable|Collection $collection)
```

Returns a new collection containing the items that do not match the given predicate.

The supplied predicate receives three arguments: item, index, collection.

```
$isEven = function ($x) { return $x % 2 === 0; };
P::reject($isEven, [1, 2, 3, 4]); // => [0 => 1, 2 => 3]
```

3.86 reverse

```
array|Collection P::reverse(array|\Traversable|Collection $collection)
```

Returns a new collection where the items are in a reverse order.

```
P::reverse([3, 2, 1]); // => [2 => 1, 1 => 2, 0 => 3]
P::reverse([22, 4, 16, 5]); // => [3 => 5, 2 => 16, 1 => 4, 0 => 22]
P::reverse([]); // => []
```

3.87 slice

```
array|Collection P::slice(int $start, int $end, array|\Traversable|Collection $collection)
```

Returns a new collection, containing the items of the original from index start (inclusive) to index end (exclusive).

```
P::slice(2, 6, [1, 2, 3, 4, 5, 6, 7, 8, 9]); // => [3, 4, 5, 6]
P::slice(0, 3, [1, 2, 3, 4, 5, 6, 7, 8, 9]); // => [1, 2, 3]
P::slice(7, 11, [1, 2, 3, 4, 5, 6, 7, 8, 9]); // => [8, 9]
```

3.88 sort

```
array|Collection P::sort(callable $comparator, array|\Traversable|Collection $collection)
```

Returns a new collection sorted by the given comparator function.

```
$sub = function ($a, $b) { return $a - $b; };
P::sort($sub, [3, 2, 4, 1]); // => [1, 2, 3, 4]
```

3.89 sortBy

```
array|Collection P::sortBy(callable $function, array|\Traversable|Collection $collection)
```

Returns a new collection sorted by comparing the values provided by calling the given function for each item.

```
$getFoo = function ($a) { return $a['foo']; };  
$collection = [['foo' => 16, 'bar' => 3], ['foo' => 5, 'bar' => 42], ['foo' => 11, 'bar' => 7]];  
P::sortBy($getFoo, $collection); // => [['foo' => 5, 'bar' => 42], ['foo' => 11, 'bar' => 7], ['foo'
```

3.90 stringIndexOf

int|false P::stringIndexOf(string \$substring, string \$string)

Returns the first index of a substring in a string, or false if the substring is not found.

```
P::stringIndexOf('def', 'abcdefdef'); // => 3  
P::stringIndexOf('a', 'abcdefgh'); // => 0  
P::stringIndexOf('ghi', 'abcdefgh'); // => false
```

3.91 stringLastIndexOf

int|false P::stringLastIndexOf(string \$substring, string \$string)

Returns the last index of a substring in a string, or false if the substring is not found.

```
P::stringLastIndexOf('def', 'abcdefdef'); // => 6  
P::stringLastIndexOf('a', 'abcdefgh'); // => 0  
P::stringLastIndexOf('ghi', 'abcdefgh'); // => false
```

3.92 substring

string P::substring(int \$start, int \$end, string \$string)

Returns a substring of the original string between given indexes.

```
P::substring(2, 5, 'foobarbaz'); // => 'oba'  
P::substring(4, 8, 'foobarbaz'); // => 'arba'  
P::substring(3, -2, 'foobarbaz'); // => 'barb'
```

3.93 substringFrom

string P::substringFrom(int \$start, string \$string)

Returns a substring of the original string starting from the given index.

```
P::substringFrom(5, 'foobarbaz'); // => 'rbaz'  
P::substringFrom(1, 'foobarbaz'); // => 'oobarbaz'  
P::substringFrom(-2, 'foobarbaz'); // => 'az'
```

3.94 substringTo

string P::substringTo(int \$end, string \$string)

Returns a substring of the original string ending before the given index.


```
P::substringTo(5, 'foobarbaz'); // => 'fooba'
P::substringTo(8, 'foobarbaz'); // => 'foobarba'
P::substringTo(-3, 'foobarbaz'); // => 'foobar'
```

3.95 subtract

```
int|float P::subtract(int|float $x, int|float $y)
```

Subtracts two numbers.

```
P::subtract(15, 27); // => -12
P::subtract(36, -8); // => 44
```

3.96 sum

```
int|float P::sum(int[]|float[] $values)
```

Adds together a list of numbers.

```
P::sum([1, 2, 3, 4, 5, 6]); // => 21
P::sum([11, 0, 2, -4, 7]); // => 16
```

3.97 tail

```
array|Collection P::tail(array|\Traversable|Collection $collection)
```

Returns a new collection that contains all the items from the original collection except the first.

```
P::tail([2, 4, 6, 3]); // => [4, 6, 3]
```

3.98 tap

```
mixed P::tap(callable $function, mixed $object)
```

Calls the provided function with the given value as a parameter and returns the value.

```
$addDay = function (\DateTime $date) { $date->add(new \DateInterval('P1D')); };
$date = new \DateTime('2015-03-15');
P::tap($addDay, $date); // => $date
$date->format('Y-m-d'); // => '2015-03-16'
```

3.99 times

```
array P::times(callable $function, int $count)
```

Calls the provided function the specified number of times and returns the results in an array.

```
$double = function ($number) { return $number * 2; };  
P::times($double, 5); // => [0, 2, 4, 6, 8]
```

3.100 toPairs

array|Collection P::toPairs(array|\Traversable|Collection \$map = null)

Creates a new list of pairs from from a list of key-value pairs.

```
P::toPairs(['a' => 'b', 'c' => 'd']); // => [['a', 'b'], ['c', 'd']]  
P::toPairs([3 => 'b', 5 => null]); // => [[3, 'b'], [5, null]]
```

3.101 true

callable P::true()

Returns a function that always returns true.

```
$true = P::true();  
$true(); // => true
```

3.102 unary

callable P::unary(callable \$function)

Wraps the given function in a function that accepts exactly one parameter.

```
$add2 = function ($a = 0, $b = 0) { return $a + $b; };  
$add1 = P::nAry(1, $add2);  
$add1(27, 15); // => 27
```

3.103 unapply

mixed P::unapply(callable \$function, mixed ...\$arguments)

Calls the function using the given arguments as a single array list argument.

Effectively creates an variadic function from a unary function.

```
$concat = function (array $strings) { return implode(' ', $strings); };  
P::unapply($concat, 'foo', 'ba', 'rba'); // => 'foo ba rba'
```

3.104 where

mixed P::where(array \$specification, array|object \$object)

Returns true if the given object matches the specification.

```
P::where(['a' => 15, 'b' => 16], ['a' => 15, 'b' => 42, 'c' => 88, 'd' => -10]); // => false
P::where(['a' => 15, 'b' => 16], ['a' => 15, 'b' => 16, 'c' => -20, 'd' => 77]); // => true
```

3.105 zip

array P::zip(array \$a, array \$b)

Returns a new array of value pairs from the values of the given arrays with matching keys.

```
P::zip([1, 2, 3], [4, 5, 6]); // => [[1, 4], [2, 5], [3, 6]]
P::zip(['a' => 1, 'b' => 2], ['a' => 3, 'c' => 4]); // => ['a' => [1, 3]]
P::zip([1, 2, 3], []); // => []
```

3.106 zipWith

array P::zipWith(callable \$function, array \$a, array \$b)

Returns a new array of values created by calling the given function with the matching values of the given arrays.

```
$sum = function ($x, $y) { return $x + $y; };
P::zipWith($sum, [1, 2, 3], [5, 6]); // => [6, 8]
```