Dsheiko\Extras v.1.0.0

https://dsheiko.github.io/extras/

Dsheiko\Extras\Arrays JavaScript

assign(array $array, ...$sources): array
```
$res = Arrays::assign(["foo" => 1, "bar" => 2], ["bar" => 3],
["foo" => 4], ["baz" => 5]);
// $res === ["foo" => 4, "bar" => 3, "baz" => 5]
```

concat(array $array, array ...$targets): array
```
$res = Arrays::concat([1, 2], [3, 4], [5, 6]);
// [1, 2, 3, 4, 5, 6]
```

copyWithin(array $array, int $targetIndex, int $beginIndex = 0, int $endIndex = null): array
```
$res = Arrays::copyWithin([1, 2, 3, 4, 5], 0, 3, 4);
// [4, 2, 3, 4, 5]
```

each(array $array, mixed $mixed)
```
$sum = 0;
Arrays::each([1, 2, 3], function ($val, $index, $array)
use(&$sum) {
    $sum += $val;
});
```

entries/pairs(array $array): array
```
$res = Arrays::entries([
            "foo" => "FOO",
            "bar" => "BAR",
        ]);
// [["foo", "FOO"], ["bar", "BAR"]]
```

every(array $array, mixed $mixed): bool
```
$res = Arrays::every([1, 2, 3], function($num, $index,
$array){ return $num > 1; }); // false
```

fill(array $array, $value, int $beginIndex = 0, int $endIndex = null): array
```
$res = Arrays::fill([1, 2, 3], 4); // [4, 4, 4]
$res = Arrays::fill([1, 2, 3], 4, 1); // [1, 4, 4]
```

filter(array $array, callable $predicate)
```
$array = Arrays::filter([1, 2, 3], function($num){ return $num
> 1; });
```

find(array $array, callable $predicate)
```
$value = Arrays::find([1, 2, 3], function($num){ return $num >
1; });
```

from/toArray($collection): array
```
$res = Arrays::from(new \ArrayObject([1,2,3])); // [1,2,3]

$obj = new \ArrayObject([1,2,3]);
$res = Arrays::from($obj->getIterator()); // [1,2,3]
```

hasOwnProperty/has(array $array, mixed $key): bool
```
$res = Arrays::hasOwnProperty(["foo" => "FOO"], "foo");// true
```

includes/contains(array $array, $searchElement, int $fromIndex = null): bool
```
$res = Arrays::includes([1, 2, 3], 2); // true
$res = Arrays::includes([1, 2, 3, 5, 6, 7], 2, 3); // false
```

indexOf(array $array, $searchElement, int $fromIndex = 0): int
```
$src = ["ant", "bison", "camel", "duck", "bison"];
$res = Arrays::indexOf($src, "bison"); // 1
$res = Arrays::indexOf($src, "bison", 2); // 4
```

is(array $array, array $arrayToCompare): bool
```
$a = [1,2,3];
$b = [1,2,3];
$res = Arrays::is($a, $b); // true
```

join(array $array, mixed $separator = ",")
```
$res = Arrays::join([1,2,3], ":"); // "1:2:3"
```

keys(array $array, $searchValue = null): array
```
$res = Arrays::keys(["foo" => "FOO", "bar" => "BAR"]); //
["foo", "bar"]
$res = Arrays::keys(["foo" => "FOO", "bar" => "BAR"], "BAR");
// ["bar"]
```

lastIndexOf(array $array, $searchElement, int $fromIndex = null): int
```
$src = [2, 5, 9, 2];
$res = Arrays::lastIndexOf($src, 2); // 3
$res = Arrays::lastIndexOf($src, 2, 2); // 0
```

map(array $array, mixed $mixed): array
```
$res = Arrays::map([1, 2, 3], function($num){ return $num + 1;
});
```

of(...$args): array
```
$res = Arrays::of(1, 2, 3); // [1, 2, 3]
```

pop(array &$array)
```
$src = [1, 2, 3];
$res = Arrays::pop($src); // 3
```

push(array $array, $value): array
```
$src = [1,2,3];
$res = Arrays::push($src, 4); // [1, 2, 3, 4]
```

reduceRight(array $array, mixed $mixed, $initial = null)
```
$res = Arrays::reduceRight([1,2,3], function(array $carry, int
$num){
  $carry[] = $num;
  return $carry;
}, []);
// [3,2,1]
```

reduce(array $array, mixed $mixed, $initial = null)
```
$res = Arrays::assign(["foo" => 1, "bar" => 2], ["bar" => 3],
["foo" => 4], ["baz" => 5]);
// $res === ["foo" => 4, "bar" => 3, "baz" => 5]
```

reverse(array $array): array
```
$res = Arrays::reverse([1,2,3]); // [3, 2, 1]
```

shift(array &$array)
```
$src = [1, 2, 3];
$res = Arrays::shift($src); // 1
```

slice(array $array, int $beginIndex, int $endIndex = null): array
```
$src = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
$res = Arrays::slice($src, 1, 3); // ["Orange","Lemon"]
```

some(array $array, mixed $mixed): bool
```
$res = Arrays::some([1, 2, 3], function($num){ return $num >
1; }); // true
```

sort(array $array, mixed $mixed = null): array
```
$res = Arrays::sort([3,2,1]);  // [1,2,3]
$res = Arrays::sort([3,2,1], function($a, $b){
            return $a <=> $b;
        });  // [1,2,3]
```

splice(array $array, int $beginIndex, int $deleteCount = null, ...$items): array
```
// remove 1 element from index 2, and insert "trumpet"
$src = ["angel", "clown", "drum", "sturgeon"];
$res = Arrays::splice($src, 2, 1, "trumpet");
// ["angel", "clown", "trumpet", "sturgeon"]
```

unshift(array &$array, ...$values)
```
$src = [1, 2];
$src = Arrays::unshift($src, 0);
// [0, 1, 2]
```

values(array $array): array
```
$res = Arrays::values([ 5 => 1, 10 => 2, 100 => 3]); //
[1,2,3]
```

Chaining

```
$res = Arrays::chain([1, 2, 3])
    ->map(function($num){ return $num + 1; })
    ->filter(function($num){ return $num > 1; })
    ->reduce(function($carry, $num){
        return $carry + $num; }, 0)
    ->value();
```

Dsheiko\Extras $^{\text{v.1.0.0}}$

https://dsheiko.github.io/extras/

Dsheiko\Extras\Arrays Underscore.js

where(array $array, array $conditions): array
```
$res = Arrays::where($listOfPlays, ["author" => "Shakespeare",
"year" => 1611]);
// [ ["title" => "Cymbeline", "author" => "Shakespeare",
"year" => 1611],
//    ["title" => "The Tempest", "author" => "Shakespeare",
"year" => 1611],]
```

size(array $array): int
```
$res = Arrays::size(["one" => 1, "two" => 2, "three" => 3]);
// 3
```

findWhere(array $array, array $props)
```
$res = Arrays::findWhere($listOfPlays, ["author" =>
"Shakespeare", "year" => 1611]);
// ["title" => "Cymbeline", "author" => "Shakespeare", "year"
=> 1611]
```

partition(array $array, mixed $mixed): array
```
$res = Arrays::partition([0, 1, 2, 3, 4, 5], function($val) {
    return $val % 2;
}); //  [[1, 3, 5], [0, 2, 4]]
```

reject(array $array, mixed $predicate)
```
$res = Arrays::reject([1, 2, 3, 4, 5, 6], function ($num){
    return $num % 2 == 0;
}); // [1,3,5]
```

first(array $array, $defaultValue = null)
```
$element = Arrays::first([1, 2, 3]);
$element = Arrays::first($arr, 1);
$element = Arrays::first($arr, function(){ return 1; });
```

invoke(array $array, mixed $iteratee, ...$args): array
```
$res = Arrays::invoke([[5, 1, 7], [3, 2, 1]], [Arrays::class,
"sort"]); // [ [1, 5, 7], [1, 2, 3] ]
```

initial(array $array, int $count = 1): array
```
$res = Arrays::initial([5, 4, 3, 2, 1]); // [5, 4, 3, 2]
$res = Arrays::initial([5, 4, 3, 2, 1], 3); // [5, 4]
```

pluck(array $array, mixed $key): array
```
$res = Arrays::pluck([
      ["name" => "moe",   "age" =>  40],
      ["name" => "larry", "age" =>  50],
      ["name" => "curly", "age" =>  60],
  ], "name"); // ["moe", "larry", "curly" ]
```

last(array $array)
```
$element = Arrays::last([1, 2, 3]);
```

max(array $array, mixed $iteratee = null, $context = null)
```
$res = Arrays::max([1,2,3]); // 3
$res = Arrays::max([
        ["name" => "moe",   "age" =>  40],
        ["name" => "larry", "age" =>  50],
        ["name" => "curly", "age" =>  60],
    ], function($stooge){
     return $stooge["age"];
    });
// ["name" => "curly", "age" =>  60]
```

rest(array $array, int $count = 1): array
```
$res = Arrays::rest([5, 4, 3, 2, 1]); // [4, 3, 2, 1]
//...
$res = Arrays::rest([5, 4, 3, 2, 1], 3); // [2, 1]
```

min(array $array, mixed $iteratee = null, $context = null)
```
$res = Arrays::min([1,2,3]); // 1
$res = Arrays::min([
        ["name" => "moe",   "age" =>  40],
        ["name" => "larry", "age" =>  50],
        ["name" => "curly", "age" =>  60],
    ], function($stooge){
     return $stooge["age"];
    }); // ["name" => "moe",   "age" =>  40]
```

compact(array $array): array
```
$res = Arrays::compact([0, 1, false, 2, '', 3]); // [1, 2, 3]
```

sortBy(array $array, $iteratee, $context = null): array
```
$res = Arrays::sortBy([1, 2, 3, 4, 5, 6], function($a){
    return \sin($a);
}); // [5, 4, 6, 3, 1, 2]
$res = Arrays::sortBy([
    ["name" => "moe",   "age" =>  40],
    ["name" => "larry", "age" =>  50],
    ["name" => "curly", "age" =>  60],
], "name"); // [["name" => "curly", "age" =>  60],...]
```

flatten(array $array, bool $shallow = false): array
```
$res = Arrays::flatten([1, [2], [3, [[4]]]]); // [1, 2, 3, 4]
//...
$res = Arrays::flatten([1, [2], [3, [[4]]]], true); // [1, 2,
3, [[4]]]
```

groupBy(array $array, $iteratee, $context = null): array
```
$res = Arrays::groupBy([1.3, 2.1, 2.4], function($num) {
return floor($num); });
// [1 => [ 1.3 ], 2 => [ 2.1, 2.4 ]]
```

without(array $array, ...$values): array
```
$res = Arrays::without([1, 2, 1, 0, 3, 1, 4], 0, 1);
// [2, 3, 4]
```

indexBy(array $array, $iteratee, $context = null): array
```
$res = Arrays::indexBy([
    ["name" => "moe",   "age" =>  40],
    ["name" => "larry", "age" =>  50],
    ["name" => "curly", "age" =>  60],
], "name");
// [ 40 => ["name" => "moe",   "age" =>  40], ...]
```

union(...$args): array
```
$res = Arrays::union(
    [1, 2, 3],
    [101, 2, 1, 10],
    [2, 1]
); // [1, 2, 3, 101, 10]
```

countBy(array $array, $iteratee, $context = null): array
```
$res = Arrays::countBy([1, 2, 3, 4, 5], function($num) {
    return $num % 2 == 0 ? "even": "odd";
});
// [ "odd => 3, "even" => 2 ]
```

intersection(array $array, ...$sources): array
```
$res = Arrays::intersection(
    ["a" => "green", "b" => "brown", "c" => "blue", "red"],
    ["a" => "green", "b" => "yellow", "blue", "red"]
); // [ "a" => "green" ]
```

shuffle(array $array): array
```
$res = Arrays::shuffle([1, 2, 3]); // [ 2, 1, 3 ]
```

difference(array $array, ...$sources): array
```
$res = Arrays::difference(
    ["a" => "green", "b" => "brown", "c" => "blue", "red"],
    ["a" => "green", "yellow", "red"]
); // [ "b" => "brown", "c" => "blue",  "red" ]
```

sample(array $array, int $count = null)
```
$res = Arrays::sample([1, 2, 3], 3); // [ 2, 1, 3 ]
```

uniq(array $array): array
```
$res = Arrays::uniq([1,2,3,1,1,2]); // [1,2,3]
```

Dsheiko\Extras\Arrays Underscore.js

zip(array $array, ...$sources): array

```
$res = Arrays::zip(
  ["moe", "larry", "curly"],
  [30, 40, 50],
  [true, false, false]
); //  [["moe", 30, true], ["larry", 40, false], ["curly", 50,
false]]
```

invert(array $array): array

```
$res = Arrays::invert([
    "Moe" => "Moses",
    "Larry" => "Louis",
    "Curly" => "Jerome",
]);
// ["Moses" => "Moe", "Louis" => "Larry", "Jerome" => "Curly"]
```

unzip(array $array, ...$sources): array

```
$res = Arrays::unzip([["moe", 30, true], ["larry", 40, false],
["curly", 50, false]]);
//  [["moe", "larry", "curly"], [30, 40, 50], [true, false,
false]]
```

defaults(array $array, array $defaults): array

```
$res = Arrays::defaults([
    "flavor" => "chocolate"
 ], [
     "flavor" => "vanilla",
     "sprinkles" => "lots",
 ]); //["flavor" => "chocolate", "sprinkles" => "lots", ]
```

object(array $array, array $values = null): PlainObject

```
$obj = Arrays::object([ "foo" =>
            [
                "bar" => [
                    "baz" => "BAZ"
                ]
            ]
        ]);
echo $obj->foo->bar->baz; // BAZ
```

property(string $prop): callable

```
$stooge = [ "name" => "moe" ];
$res = Arrays::property("name")($stooge); // "moe"
```

sortedIndex(array $array, $value, $iteratee = null, $context = null): int

```
$res = Arrays::sortedIndex([10, 20, 30, 40, 50], 35); // 3
```

propertyOf(array $array): callable

```
$stooge = [ "name" => "moe" ];
$res = Arrays::propertyOf($stooge)("name"); // "moe"
```

findIndex(array $array, $iteratee = null, $context = null): int

```
$inx = Arrays::findIndex([
            ["val" => "FOO"],
            ["val" => "BAR"],
        ], function ($item){
            return $item["val"] === "BAR";
        }); // 1
```

matcher(array $attrs): callable

```
$matcher = Arrays::matcher(["foo" => "FOO", "bar" => "BAR"]);
$res = Arrays::filter($src, $matcher);
```

findLastIndex(array $array, $iteratee = null, $context = null): int

```
$src = [
    [
        'id' => 1, 'name' => 'Ted', 'last' => 'White',
    ],
    [
        'id' => 2, 'name' => 'Bob', 'last' => 'Brown',
    ],
    [
        'id' => 3, 'name' => 'Ted', 'last' => 'Jones',
    ],
];

$res = Arrays::findLastIndex($src, [ "name" => "Ted" ]); // 2
```

findKey(array $array, $iteratee = null, $context = null): string

```
$src = [
    "foo" => [
        'name' => 'Ted',
        'last' => 'White',
    ],
    "bar" => [
        'name' => 'Frank',
        'last' => 'James',
    ],
    "baz" => [
        'name' => 'Ted',
        'last' => 'Jones',
    ],
];
$res = Arrays::findKey($src, [ "name" => "Ted" ]); // foo
```

range(int $start, int $end = null, int $step = 1): array

```
$res = Arrays::range(0, 30, 5); // [0, 5, 10, 15, 20, 25]
```

isEmpty(array $array): bool

```
$res = Arrays::isEmpty([]); // true
```

chain($array): Arrays

```
$res = Arrays::chain([1, 2, 3])
  ->map(function($num){ return $num + 1; })
  ->filter(function($num){ return $num > 1; })
  ->reduce(function($carry, $num){ return $carry + $num; }, 0)
  ->value();
```

pick(array $array, ...$keys): array

```
$res = Arrays::pick([
    'name' => 'moe',
    'age' => 50,
    'userid' => 'moe1',
  ], 'name', 'age'); // ['name' => 'moe', 'age' => 50, ]
```

mapObject(array $array, callable $iteratee, $context = null): array

```
<?php
$res = Arrays::mapObject([
    "start" => 5,
    "end" => 12,
], function($val){
    return $val + 5;
}); // [ "start" => 10, "end" => 17, ]
```

omit(array $array, ...$keys): array

```
<?php
$res = Arrays::omit([
    'name' => 'moe',
    'age' => 50,
    'userid' => 'moe1',
  ], 'userid');
// ['name' => 'moe', 'age' => 50, ]
```

isMatch(array $array, array $attrs): bool

```
$res = Arrays::isMatch([
        "foo" => "FOO",
        "bar" => "BAR",
        "baz" => "BAZ",
    ],
    [
        "foo" => "BAZ",
    ]); // false
```

isEqual(array $array, array $target): bool

```
$res = Arrays::isEqual([
        "name" => "moe",
        "luckyNumbers" => [13, 27, 34],
        ], [
        "name" => "moe",
        "luckyNumbers" => [13, 27, 34],
]); // true
```

isArray(array $array): bool

```
$res = Arrays::isArray([ 1, 2, 3 ]); // true
```

Dsheiko\Extras v.1.0.0

https://dsheiko.github.io/extras/

Dsheiko\Extras\Functions JavaScript

apply(mixed $source, $context = null, array $args = [])
```
$obj = Arrays::object(["foo" => "FOO"]);
$source = function( $input ){ return $input . "_" . $this-
>foo; };
$res = Functions::apply($source, $obj, ["BAR"]); // "BAR_FOO"
```

bind(mixed $source, $context = null): mixed
```
$obj = Arrays::object(["foo" => "FOO"]);
$source = function( $input ){ return $input . "_" . $this-
>foo; };
$func = Functions::bind($source, $obj);
echo $func("BAR"); // "BAR_FOO"
```

call(mixed $source, $context = null, ...$args)
```
$obj = Arrays::object(["foo" => "FOO"]);
$source = function( $input ){ return $input . "_" . $this-
>foo; };
$res = Functions::call($source, $obj, "BAR"); // "BAR_FOO"
```

toString(mixed $source)
```
echo Functions::toString("strlen");
```

Dsheiko\Extras\Functions Underscore.js

bindAll($obj, ...$methodNames)
```
$foo = (object)[
    "value" => 1,
    "increment" => function(){
        $this->value++;
    },
    "reset" => function(){
        $this->value = 0;
    }
];
Functions::bindAll($foo, "increment", "reset");
($foo->increment)();
echo $foo->value; // 2
($foo->reset)();
echo $foo->value; // 0
```

once(mixed $source)
```
function increment()
{
    static $count = 0;
    return ++$count;
}
$func = Functions::once("increment");
$func(); // 1
$func(); // 1
$func(); // 1
```

partial(mixed $source, ...$boundArgs)
```
$subtract = function($a, $b) { return $b - $a; };
$sub5 = Functions::partial($subtract, 5);
$res = $sub5(20); // 15
```

memoize($source, $hasher = null)
```
$counter = Functions::memoize("fixtureCounter::increment");
$counter($foo); // 1
$counter($foo); // 1
$counter($bar); // 2
```

delay(mixed $source, int $wait, ...$args)
```
$counter = Functions::memoize("fixtureCounter::increment");
$counter($foo); // 1
$counter($foo); // 1
$counter($bar); // 2
```

negate(mixed $source)
```
$func = Functions::negate(function(){ return false; });
$func(): // true
```

throttle(mixed $source, int $wait)
```
function increment()
{
    static $count = 0;
    return ++$count;
}
$func = Functions::throttle("increment", 20);
$func(); // 1
$func(); // false
usleep(20000);
$func();  // 2
$func();  // false
```

debounce(mixed $source, int $wait)
```
function increment()
{
    static $count = 0;
    return ++$count;
}
$func = Functions::debounce("increment", 20);
$func(); // false
$func(); // false
usleep(20000);
$func();  // 1
$func();  // false
```

after(mixed $source, int $count)
```
function increment()
{
    static $count = 0;
    return ++$count;
}
$func = Functions::after("increment", 2);
$func(); // false
$func(); // false
$func(); // 1
```

before(mixed $source, int $count)
```
function increment()
{
    static $count = 0;
    return ++$count;
}
$func = Functions::before("increment", 2);
$func(); // 1
$func(); // 2
$func(); // 2
```

wrap(mixed $source, mixed $transformer)
```
function increment()
{
    static $count = 0;
    return ++$count;
}
$func = Functions::wrap("increment", function($func){
    return 10 + $func();
});
$func(); // 11
```

compose(...$functions)
```
$greet = function(mixed $name){ return "hi: " . $name; };
$exclaim = function(mixed $statement){ return
strtoupper($statement) . "!"; };
$welcome = Functions::compose($greet, $exclaim);
$welcome("moe"); // "hi: MOE!"
```

times(callable $source, int $n = 1, $context = null)
```
$counter = 0;
Functions::times(function($value) use(&$counter){
    $counter += $value;
}, 5); // 15
```

chain(mixed $value): Functions
```
$res = Strings::chain( " 12345 " )
            ->replace("/1/", "5")
            ->replace("/2/", "5")
            ->trim()
            ->substr(1, 3)
            ->value();
echo $res; // "534"
```

Dsheiko\Extras v.1.0.0

https://dsheiko.github.io/extras/

Dsheiko\Extras\Strings

charAt(string $value, int $index = 0): string
```php
$res = Strings::charAt("ABC", 1); // "B"
```

charCodeAt(string $value, int $index = 0): int
```php
$res = Strings::charCodeAt("ABC", 0); // 65
```

concat(string $value, ...$strings): string
```php
$res = Strings::concat("AB", "CD", "EF"); // ABCDEF
```

endsWith(string $value, string $search): bool
```php
$res = Strings::endsWith("12345", "45"); // true
```

fromCharCode(...$codes): string
```php
$res = Strings::fromCharCode(65, 66, 67); // ABC
```

includes(string $value, string $search, int $position = 0): bool
```php
$res = Strings::includes("12345", "1"); // true
```

indexOf(string $value, string $searchStr, int $fromIndex = 0): int
```php
$res = Strings::indexOf("ABCD", "BC"); // 1
$res = Strings::indexOf("ABCABC", "BC", 3); // 4
```

lastIndexOf(string $value, string $searchStr, int $fromIndex = 0): int
```php
$res = Strings::lastIndexOf("canal", "a"); // 3
$res = Strings::lastIndexOf("canal", "a", 2); // 1
```

localeCompare(string $value, string $compareStr): int
```php
\setlocale (LC_COLLATE, 'de_DE');
$res = Strings::localeCompare("a", "c"); // -2
```

match(string $value, string $regexp): null|array
```php
$res = Strings::match("A1B1C1", "/[A-Z]/"); // ["A", "B", "C"]
```

padEnd(string $value, int $length, string $padString = " "): string
```php
$res = Strings::padEnd("abc", 10); // "abc       "
$res = Strings::padEnd("abc", 10, "foo"); // "abcfoofoof"
```

padStart(string $value, int $length, string $padString = " "): string
```php
$res = Strings::padStart("abc", 10); // "       abc"
$res = Strings::padStart("abc", 10, "foo"); // "foofoofabc"
```

remove(string $value, string $search): string
```php
$res = Strings::remove("12345", "1"); // "2345"
```

repeat(string $value, int $count): string
```php
$res = Strings::repeat("abc", 2); // abcabc
```

replace(string $value, string $pattern, string $replacement): string
```php
$res = Strings::replace("12345", "/\d/s", "*"); // "*****"
```

slice(string $value, int $beginIndex, int $endIndex = null): string
```php
$res = Strings::slice("The morning is upon us.", 1, 8);
// "he morn"
```

split(string $value, string $delimiter): array
```php
$res = Strings::split("a,b,c", ","); // ["a", "b", "c"]
```

startsWith(string $value, string $search): bool
```php
$res = Strings::startsWith("12345", "12"); // true
```

substr(string $value, int $start, int $length = null): string
```php
$res = Strings::substr("12345", 1, 3); // "234"
```

substring(string $value, int $beginIndex, int $endIndex = null): string
```php
$value = "Mozilla";
$res = Strings::substring($value, 0, 1); // "M"
$res = Strings::substring($value, 1, 0); // "M"
```

toLowerCase(string $value): string
```php
$res = Strings::toLowerCase("AbC"); // abc
```

toUpperCase(string $value): string
```php
$res = Strings::toUpperCase("AbC"); // ABC
```

trim(string $value, string $mask = " \t\n\r\0\x0B"): string
```php
$res = Strings::trim("  12345   "); // "12345"
```

chain(string $value): Strings
```php
$res = Strings::chain( " 12345 " )
            ->replace("/1/", "5")
            ->replace("/2/", "5")
            ->trim()
            ->substr(1, 3)
            ->value();// "534"
```

escape(string $string): string
```php
$res = Strings::escape("Curly, Larry & Moe");
// "Curly, Larry &amp; Moe"
```

unescape(string $string): string
```php
$res = Strings::unescape("Curly, Larry &amp; Moe");
// "Curly, Larry & Moe"
```

Dsheiko\Extras\Numbers

isFinite($source): bool
```php
$res = Numbers::isFinite(log(0)); // true
```

isInteger($source): bool
```php
$res = Numbers::isInteger(123); // true
```

isNaN($source): bool
```php
$res = Numbers::isNaN(\NAN); // true
```

parseFloat($source)
```php
$src = "4.567abcdefgh";
echo Numbers::isNaN(Numbers::parseFloat($src)); // true
```

parseInt ($source): int
```php
$res = Numbers::parseInt("0xF", 16); // 15
```

toFixed(float $value, int $digits = 0): float
```php
$res = Numbers::toFixed(12345.6789, 6); // 12345.678900
$res = Numbers::toFixed(12345.6789, 1); // 12345.7
```

toPrecision(float $value, int $precision = null): float
```php
$res = Numbers::toPrecision(5.123456); // 5.123456
$res = Numbers::toPrecision(5.123456, 2); // 5.1
```

isNumber($source): bool
```php
$res = Numbers::isNumber(1); // true
$res = Numbers::isNumber(1.1); // true
```

Dsheiko\Extras\Any

```php
use \Dsheiko\Extras\Any;

$res = Any::chain(new \ArrayObject([1,2,3]))
    ->toArray() // value is [1,2,3]
    ->map(function($num){ return [ "num" => $num ]; })
    // value is [[ "num" => 1, ..]]
    ->reduce(function($carry, $arr){
        $carry .= $arr["num"];
        return $carry;

    }, "") // value is "123"
    ->replace("/2/", "") // value is "13"
    ->then(function($value){
      if (empty($value)) {
        throw new \Exception("Empty value");
      }
      return $value;
    })
    ->value();
echo $res; // "13"
```

Dsheiko\Extras\Type\PlainObject

```php
use Dsheiko\Extras\Type\PlainObject;

$po = new PlainObject(["foo" => "FOO", "bar" => "BAR"]);
// $po = \Dsheiko\Extras\Array::object(["foo" => "FOO",
"bar" => "BAR"]);
echo $po->foo; // "FOO"
echo $po->bar; // "BAR"
```