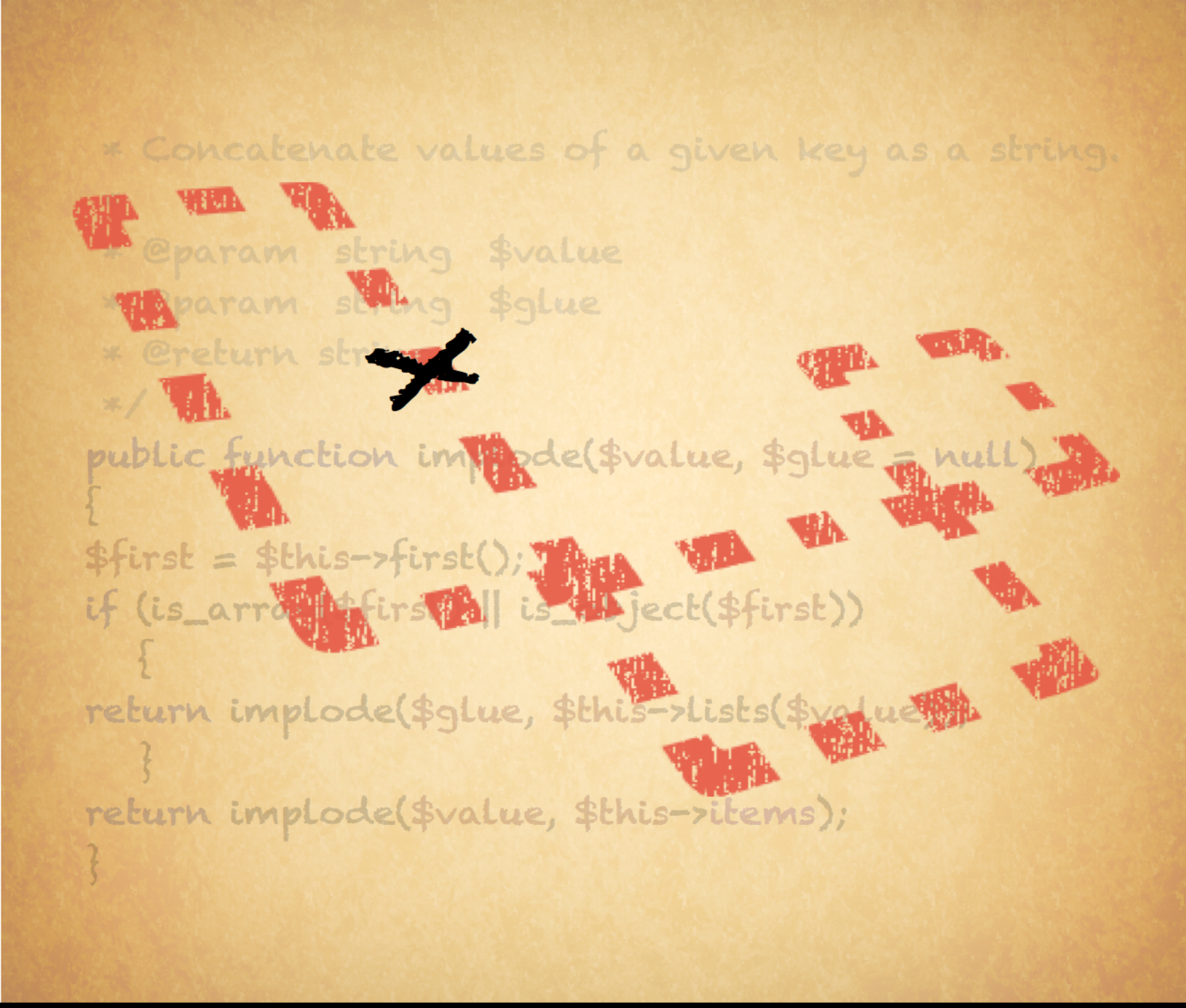


# The Hidden Treasures

## Laravel 5

Fakiolas Marios

# Laravel 5 - The Hidden Treasures

## with examples and tips

Marios Fakiolas

This book is for sale at http://leanpub.com/laravel-thehiddentreasures

This version was published on 2015-05-21


This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress ebook using lightweight tools and many iterations to get reader feedback, pivot until you have the right book and build traction once you do.

*You never stopped believing in me,*
*thank you Xristina*

# Contents

# Collections

Laravel5 offers a great way to manipulate arrays through the `Illuminate\Support\Collection` class and its enormous variety of methods. We can chain those methods and apply many changes with very little code which is great. Let's see a fast example:

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe', 'sex' => 'male'],
        ['id' => 2, 'name' => 'Jane Doe', 'sex' => 'female'],
        ['id' => 3, 'name' => 'Jack Doe', 'sex' => 'male'],
])->filter(function($item) {
        return $item['sex'] == 'male';
})->sortBy('name')->values()->all();

$collection = [
        ['id' => 3, 'name' => 'Jack Doe', 'sex' => 'male'],
        ['id' => 1, 'name' => 'John Doe', 'sex' => 'male']
];
```

Wow, what a flexibility? We managed to filter a collection in order to keep only the male users, we ordered this new collection by name in ascending order and then we forced a reset for the items keys. Imagine i didn't use `foreach()` at all. How cool is that?

Most of the times a new collection is created but there are cases we have to be careful because the changes are applied to the collection itself. Eloquent models collections are returned as Collection instances so it is easy to understand how important is to master this utility. Furthermore apart from the Eloquent collections we can use the Collection class to many other cases and provide nice, clean and testable code.

Below every method of Collection class is explained with some brief and declarative examples per method so you can grasp their functionality with ease.

## all()

Get all of the items in the collection.

**Method**:

```php
public function all();
```

**Example:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection->all();
```

**Result:**

```
[
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
];
```

**Notes:**

Be careful when you use `all()` then the result is not a collection anymore so you cannot chain another method to it so do this before it.

# chunk()

Chunk the underlying collection array.

**Method:**

```php
public function chunk($size, $preserveKeys = false);
```

**Example 1:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->chunk(2);
```

**Result:**

```
$collection2->all() = [collect([1,2]), collect([3,4]), collect([5])];
```

**Example 2:**

```
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->chunk(2, true);
```

**Result:**

```
$collection2->all() = [collect([1,2]), collect([2 => 3, 3 => 4]), collect([4 => \
5])];
```

**Notes:**

This method helps us to split a collection of items into small chunks. This is very useful when we handle extremely big collections of data or even when we want to present a collection in a responsive template and we use a CSS Framework like Bootstrap or Foundation. The second parameter forces by default the keys inside the arrays chunks to be regenerated from the start. If it is set to true then the old keys are used.

# collapse()

Collapse the collection of items into a single array.

**Method:**

```
public function collapse();
```

**Example 1:**

```
$collection1 = collect([
        [1,2,3,4,5],
        [6,7,8,9]
]);

$collection2 = $collection1->collapse();
```

**Result:**

```
$collection2->all() = [1,2,3,4,5,6,7,8,9];
```

**Example 2:**

```
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->collapse();
```

**Result:**

```
$collection2->all() = ['id' => 2, 'name' => 'Jane Doe'];
```

**Notes:**

Be careful when you use this method in real life with models that use strings as keys because the new pairs will overwrite the old ones so you 'll end up with the last model of the collection just like our second example shows.

# contains()

Check if a collection contains an item.

**Method:**

```
public function contains($key, $value = null);
```

**Example 1:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection->contains(['id' => 1, 'name' => 'John Doe']);
```

**Result:**

```
true
```

**Example 2:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection->contains(['id' => 111111, 'name' => 'John Doe']);
```

**Result:**

```
false
```

# count()

Count the number of items in the collection.

**Method:**

```
public function count();
```

**Example 1:**

```
$collection = collect([1,2,3,4,5]);

$count = $collection->count();
```

**Result:**

```
$count = 5;
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$count = $collection->count();
```

**Result:**

```php
$count = 3;
```

# diff()

Diff the collection with the given items.

**Method:**

```php
public function diff($items);
```

**Example 1:**

```php
$collection1 = collect([1,2,3,4,5]);
$collection2 = collect([1,4,5,6]);

$collection3 = $collection1->diff($collection2);
```

**Result:**

```php
$collection3->all() = [1 => 2, 2 => 3];
```

**Example 2:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->diff([1]);
```

**Result:**

```
$collection3->all() = [1 => 2, 2 => 3, 3 => 4, 4 => 5];
```

**Notes:**

A new collection is returned after removing all the common values. Unique values retain their keys. We could also just pass an array of values to check and not another collection.

# each()

Execute a callback over each collection's item.

**Method:**

```
public function each(callable $callback);
```

**Example:**

```
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = collect();

$collection1->each(function($item) use ($collection2) {
        if($item['id'] > 1) $collection2[] = $item;
});
```

**Result:**

```
$collection2->all() = [
        ['id' => 2, 'name' => 'Jane Doe']
]);
```

**Notes:**

Be careful it is just an iterator and it does return the collection itself without applying any changes to its items. Also since it uses a callback remember to use `use()` to call any variable you like inside the iterator.

# fetch()

Fetch a nested element of the collection.

**Method:**

```php
public function fetch($key);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->fetch('id');
```

**Result:**

```php
$collection2->all() = [1, 2]
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->fetch('name');
```

**Result:**

```php
$collection2->all() = ['John Doe', 'Jane Doe'];
```

**Notes:**

Very important method which can become extremely helpful if you want an array of the id keys of all collections models and many more.

# filter()

Run a filter over each of the items.

**Method:**

```php
public function filter(callable $callback);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe', 'sex' => 1],
        ['id' => 2, 'name' => 'Jane Doe', 'sex' => 2]
]);

$collection2 = $collection1->filter(function($item) {
        return $item['id'] > 1;
});
```

**Result:**

```php
$collection2->all() = [
        1 => ['id' => 2, 'name' => 'Jane Doe', 'sex' => 2]
]);
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe', 'sex' => 1],
        ['id' => 2, 'name' => 'Jane Doe', 'sex' => 2],
        ['id' => 3, 'name' => 'Jack Doe', 'sex' => 1],
]);

$men = $collection1->filter(function($item) {
        return $item['sex'] == 1;
})->values();

$women = $collection1->filter(function($item) {
        return $item['sex'] == 2;
})->values();
```

**Result:**

```
$men->all() = [
        ['id' => 1, 'name' => 'John Doe', 'sex' => 1],
        ['id' => 3, 'name' => 'Jack Doe', 'sex' => 1]
]);

$women->all() = [
        ['id' => 2, 'name' => 'Jane Doe', 'sex' => 2]
]);
```

**Notes:**

This is the big difference with `each`. Here we can assign the result to a new collection since the filtered results are returned. Be careful because the values of the returned collection retain their keys which in most cases is not what we want. For those cases we chain `values()` method at the end of our filter iterator so the keys are reproduced as we did on our second example.

# first()

Get the first item from the collection.

**Method:**

```
public function first(callable $callback = null, $default = null);
```

**Example 1:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$item = $collection->first();
```

**Result:**

```
$item = ['id' => 1, 'name' => 'John Doe'];
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$item = $collection->first(function($key, $value) {
        return $key > 0;
});
```

**Result:**

```php
$item = ['id' => 2, 'name' => 'Jane Doe'];
```

**Example 3:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$item = $collection->first(function($key, $value) {
        return $key > 10;
}, [1,2,3]);
```

**Result:**

```php
$item = [1,2,3];
```

**Notes:**

If no parameter is passed the very first item is returned. You can also pass a callback function with some login to filter first your items. Of course if no match is found then the default parameter is returned. Be careful with the order of the `$key` and the `$value` parameters inside the callback function.

# flatten()

Get a flattened array of the items in the collection.

**Method:**

```php
public function flatten();
```

**Example:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->flatten();
```

**Result:**

```php
$collection2->all() = [1, 'John Doe', 2, 'Jane Doe'];
```

# flip()

Flip the items in the collection.

**Method:**

```php
public function flip();
```

**Example:**

```php
$collection1 = collect(['id' => 1, 'name' => 'John Doe']);

$collection2 = $collection1->flip();
```

**Result:**

```php
$collection2->all() = ['1' => 'id', 'John Doe' => 'name'];
```

**Notes:**

Be careful this method doesn't work with multidimensional arrays.

# forget()

Remove an item from the collection by key.

**Method:**

```php
public function forget($key);
```

**Example:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection->forget(0);
```

**Result:**

```php
$collection->all() = [
        1 => ['id' => 2, 'name' => 'Jane Doe']
];

$collection->values()->all() = [
        ['id' => 2, 'name' => 'Jane Doe']
];
```

**Notes:**

Method `forget()` affects the collection itself without returning anything. So after we apply it we simply use the affected collection. Remember that `values()` updates the collection's keys.

# forPage()

"Paginate" the collection by slicing it into a smaller collection.

**Method:**

```php
public function forPage($page, $perPage);
```

**Example:**

```
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$collection2 = $collection1->forPage(2,2);
```

**Result:**

```
$collection2->all() = [
        ['id' => 3, 'name' => 'John Doe']
];
```

**Notes:**

The first parameter inicates the page we need and the second one the number of items per page.

# get()

Get an item from a collection by key.

**Method:**

```
public function get($key, $default = null);
```

**Example 1:**

```
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->get(0);
```

**Result:**

```
$collection2 = ['id' => 1, 'name' => 'John Doe'];
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->get(25);
```

**Result:**

```php
$collection2 = null;
```

**Example 3:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->get(25, 1);
```

**Result:**

```php
$collection2 = 1;
```

**Notes:**

In case the key we want doesn't exist the `get()` method returns the value of the second parameter as a fallback.

# groupBy()

Group an associative array by a field or using a callback.

**Method:**

```php
public function groupBy($groupBy);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$collection2 = $collection1->groupBy('name');
```

**Result:**

```php
$collection2->all() = [
        'John Doe' => [
                ['id' => 1, 'name' => 'John Doe'],
                ['id' => 3, 'name' => 'John Doe']
        ],
        'Jane Doe' => [
                ['id' => 2, 'name' => 'Jane Doe']
        ]
];
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$collection2 = $collection1->groupBy(function($item) {
        return $item['name'] == 'John Doe';
});
```

**Result:**

```
$collection2->all() = [
        [
                ['id' => 2, 'name' => 'Jane Doe']
        ],
        [
                ['id' => 1, 'name' => 'John Doe'],
                ['id' => 3, 'name' => 'John Doe']
        ]
];
```

**Notes:**

The parameter can be either a string like our first example either a callback function like our second example.

# has()

Determine if an item exists in the collection by key.

**Method:**

```
public function has($key);
```

**Example 1:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$result = $collection->has(0);
```

**Result:**

```
$result = true;
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$result = $collection->has(10);
```

**Result:**

```php
$result = false;
```

# implode()

Concatenate values of a given key as a string.

**Method:**

```php
public function implode($value, $glue = null);
```

**Example 1:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$result = $collection->implode('id');
```

**Result:**

```php
$result = '123';
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$result = $collection->implode('id', ',');
```

**Result:**

```php
$result = '1,2,3';
```

**Example 3:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$result = $collection->implode('name', ', ');
```

**Result:**

```php
$result = 'John Doe, Jane Doe, John Doe';
```

# intersect()

Intersect the collection with the given items. It returns a collection with common values.

**Method:**

```php
public function intersect($items);
```

**Example 1:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = collect([1,4,5,6]);

$collection3 = $collection1->intersect($collection2);
```

**Result:**

```php
$collection3->all() = [0 => 1, 3 => 4, 4 => 5];
```

**Example 2:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->intersect([1,4]);
```

**Result:**

```php
$collection2->all() = [0 => 1, 3 => 4];
```

**Example 3:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->intersect([1,4])->values();
```

**Result:**

```php
$collection2->all() = [0 => 1, 1 => 4];
```

**Notes:**

We can pass another collection or an array of values and then check for common values. The items of the returned collection retain their old keys so we can chain the `values()` method after the `intersect()` to refresh the items keys as we did on Example 3.

# isEmpty()

Determine if a collection is empty or not.

**Method:**

```php
public function isEmpty();
```

**Example 1:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$result = $collection->isEmpty();
```

**Result:**

```php
$result = false;
```

**Example 2:**

```php
$collection = collect();

$result = $collection->isEmpty();
```

**Result:**

```php
$result = true;
```

# jsonSerialize()

Convert the object into something JSON serializable.

**Method:**

```php
public function jsonSerialize();
```

**Example:**

```php
$collection = collect([1,2,3,4,5]);

$array = $collection->jsonSerialize();
```

**Result:**

```php
$array = [1,2,3,4,5];
```

# keys()

Get the keys of the collection items.

**Method:**

```php
public function keys();
```

**Example 1:**

```php
$collection1 = collect([1,2,3]);

$collection2 = $collection1->keys();
```

**Result:**

```php
$collection2->all() = [0,1,2];
```

**Example 2:**

```php
$collection1 = collect(['id' => 1, 'name' => 'John Doe']);

$collection2 = $collection1->keys();
```

**Result:**

```php
$collection2->all() = ['id', 'name'];
```

# keyBy()

Key an associative array by a field or using a callback.

**Method:**

```php
public function keyBy($keyBy);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$collection2 = $collection1->keyBy('name');
```

**Result:**

```php
$collection2->all() = [
        'John Doe' => ['id' => 3, 'name' => 'John Doe'],
        'Jane Doe' => ['id' => 2, 'name' => 'Jane Doe']
];
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$collection2 = $collection1->keyBy(function($item) {
        return $item['name'] == 'John Doe';
});
```

**Result:**

```php
$collection2->all() = [
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
];
```

**Notes:**

The parameter can be either a string like our first example either a callback function like our second example. The difference with `groupBy()` method is that here new results override previous ones if they share the same key.

# last()

Get the last item from the collection.

**Method:**

```php
public function last();
```

**Example 1:**

```php
$collection = collect([1,2,3]);

$item = $collection->last();
```

**Result:**

```php
$item = 3;
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$item = $collection->last();
```

**Result:**

```php
$item = ['id' => 3, 'name' => 'John Doe'];
```

# lists()

Get an array with the values of a given key.

**Method:**

```php
public function lists($value, $key = null);
```

**Example 1:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$list = $collection->lists('name');
```

**Result:**

```php
$list = ['John Doe', 'Jane Doe', 'John Doe'];
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$list = $collection->lists('name', 'id');
```

**Result:**

```php
$list = [
        1 => 'John Doe',
        2 => 'Jane Doe',
        3 => 'John Doe'
];
```

**Notes:**

If we don't pass the second parameter, an arrray is created with values the values of a selected items key. If we pass another key as second parameter then we get pairs with that key values as keys for our array. This method is extremely useful to create array and pass it to a select dropdown menu (maybe with `Form::select()` method if you use the illuminate HTML component).

# make()

Create a new collection instance if the value isn't one already.

**Method:**

```php
public static function make($items = null);
```

**Example 1:**

```php
$man = ['id' => 1, 'name' => 'John Doe'];
$woman = ['id' => 2, 'name' => 'Jane Doe'];
```

**Use method `make()`:**

```php
use Illuminate\Support\Collection;

$collection = Collection::make([$man, $woman]);
```

**Use helper function `collect()`:**

```php
$collection = collect([$man, $woman]);
```

**Result:**

```
$collection->all() = [
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
];
```

**Example 2:**

```php
$collection = collect();
```

**Result:**

```
$collection->all() = [];
```

# map()

Run a map over each of the items and returns a new collection.

**Method:**

```php
public function map(callable $callback);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'John Doe']
]);

$collection2 = $collection->map(function($item) {
        return ['id' => $item['id'] * 10, 'name' => $item['name'], 'age' => 30];
});
```

**Result:**

```php
$collection2->all() = [
        ['id' => 10, 'name' => 'John Doe', 'age' => 30],
        ['id' => 20, 'name' => 'Jane Doe', 'age' => 30],
        ['id' => 30, 'name' => 'John Doe', 'age' => 30]
];
```

**Example 2:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection->map(function($item) {
        return $item + 15;
});
```

**Result:**

```php
$collection2->all() = [16,17,18,19,20];
```

**Notes:**

There is a great difference between `each()` and `map()` function. You cannot assign `each()` method results to a new collection as it doesn't return anything, it just helps you iterate through a collection. On the other hand `map()` method helps you create a new collection faster by assigning its results to a new one. For `each()` method if you want to create a new collection you have to pass the new collection inside the iterator with `use()` like we did when we explained it before.

# merge()

Merge the collection with the given items or with another collection.

**Method:**

```
public function merge($items);
```

**Example 1:**

```
$collection1 = collect([1,2,3]);
$collection2 = collect([11,22,33]);

$collection3 = $collection1->merge($collection2);
```

**Result:**

```
$collection3->all() = [1,2,3,11,22,33];
```

**Example 2:**

```
$collection1 = collect([1,2,3]);

$collection2 = $collection1->merge([111,222]);
```

**Result:**

```
$collection2->all() = [1,2,3,111,222];
```

**Example 3:**

```
$collection1 = collect(['id' => 1, 'name' => 'John Doe']);

$collection2 = $collection1->merge(['id' => 111]);
```

**Result:**

```
$collection2->all() = ['id' => 111, 'name' => 'John Doe'];
```

**Notes:**

With `merge()` we can merge two collections or a collection with an array of values. The result is a new collection which has appended the new values. Be careful for key overrides if you use pairs with strings as keys.

# offsetExists()

Determine if an item exists at an offset.

**Method:**

```
public function offsetExists($key);
```

**Example 1:**

```
$collection = collect([1,2,3,4,5]);

$result = $collection->offsetExists(1);
```

**Result:**

```
$result = true;
```

**Example 2:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$result = $collection->offsetExists(5);
```

**Result:**

```
$result = false;
```

# offsetGet()

Get an item at a given offset.

**Method:**

```php
public function offsetGet($key);
```

**Example 1:**

```php
$collection = collect([1,2,3,4,5]);

$item = $collection->offsetGet(0);
```

**Result:**

```php
$item = 1;
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$item = $collection->offsetGet(2);
```

**Result:**

```php
$item = ['id' => 3, 'name' => 'Jack Doe'];
```

**Notes:**

If we pass a non-existent key then an ErrorException is thrown.

# offsetSet()

Set the item at a given offset.

**Method:**

```php
public function offsetSet($key, $value);
```

**Example 1:**

```php
$collection = collect([1,2,3,4,5]);

$collection->offsetSet(2, 123);
```

**Result:**

```php
$collection->all() = [1,2,123,4,5];
```

**Example 2:**

```php
$collection = collect([1,2,3,4,5]);

$collection->offsetSet(5, 123);
```

**Result:**

```php
$collection->all() = [1,2,3,4,5,123];
```

**Notes:**

This method updates the collection itself by replacing a value if the key exists or by adding a new key/value pair if the key doesn't exist.

# offsetUnset()

Unset the item at a given offset.

**Method:**

```php
public function offsetUnset($key);
```

**Example 1:**

```php
$collection = collect([1,2,3,4,5]);

$collection->offsetUnset(2);
```

**Result:**

```
$collection->all() = [0 => 1, 1 => 2, 3 => 4, 4 => 5];
```

**Example 2:**

```
$collection = collect([1,2,3,4,5]);

$collection->offsetUnset(22);
```

**Result:**

```
$collection->all() = [1,2,3,4,5];
```

**Notes:**

This method updates the collection itself by unseting a key if this key exists. If it doesn't then nothing happens.

# pop()

Get and remove the last item from the collection.

**Method:**

```
public function pop();
```

**Example:**

```
$collection = collect([1,2,3,4,5]);

$item = $collection->pop();
```

**Result:**

```
$item = 5;

$collection->all() = [1,2,3,4];
```

**Notes:**

The last item is returned while it is removed from the collection too.

# prepend()

Push an item onto the beginning of the collection.

**Method:**

```php
public function prepend($value);
```

**Example:**

```php
$collection = collect([1,2,3,4,5]);

$collection->prepend(111);
```

**Result:**

```php
$collection->all() = [111,1,2,3,4,5];
```

**Notes:**

This method doesn't return anything, just pushes a given item to the beginning of our collection.

## push()

Push an item onto the end of the collection.

**Method:**

```php
public function push($value);
```

**Example:**

```php
$collection = collect([1,2,3,4,5]);

$collection->push(111);
```

**Result:**

```php
$collection->all() = [1,2,3,4,5,111];
```

**Notes:**

This method doesn't return anything, just pushes a given item to the end of our collection. It is exactly the opposite of the `prepend()` method.

## pull()

Pulls an item from the collection.

**Method:**

```php
public function pull($key, $default = null);
```

**Example 1:**

```php
$collection = collect([1,2,3,4,5]);

$item = $collection->pull(1);
```

**Result:**

```php
$collection->all() = [0 => 1, 2 => 3, 3 => 4, 4 => 5];

$item = 2;
```

**Example 2:**

```php
$collection = collect([1,2,3,4,5]);

$item = $collection->pull(11, [1,2,3]);
```

**Result:**

```php
$collection->all() = [1,2,3,4,5];

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

**Notes:**

This method returns the value of the item we ‘ve just pulled and removes it also from the collection without updating the keys. If the key we want doesn’t exist a fallback value is returned and the collection stays as it is.

# put()

Put an item in the collection by key.

**Method:**

```php
public function put($key, $value);
```

**Example 1:**

```php
$collection = collect([1,2,3,4,5]);

$collection->put(1,11);
```

**Result:**

```php
$collection->all() = [1,11,3,4,5];
```

**Example 2:**

```php
$collection = collect(['id' => 1, 'name' => 'John Doe']);

$collection->put('id', 11);
```

**Result:**

```php
$collection->all() = ['id' => 11, 'name' => 'John Doe'];
```

**Notes:**

This method doesn't return anything but updates the collection itself. If the key exists already then it simply overrides it.

# random()

Get one or more items randomly from the collection.

**Method:**

```php
public function random($amount = 1);
```

**Example 1:**

```php
$collection1 = collect([1,2,3,4,5]);

$item = $collection->random();
```

**Result:**

```
$item = 4;
```

**Example 2:**

```
$collection1 = collect([1,2,3,4,5]);

$items = $collection->random(2);
```

**Result:**

```
$items = [1 => 2, 3 => 4];
```

**Notes:**

By default this method returns one item of our collection. If we pass as a parameter that we need more than one then an array of items is returned. Those items retain their old keys.

# reduce()

Reduce the collection to a single value.

**Method:**

```
public function reduce(callable $callback, $initial = null);
```

**Example 1:**

```
$collection = collect([1,2,3,4,5]);

$item = $collection->reduce(function($previous, $item) {
        return $previous.'/'.$item;
});
```

**Result:**

```
$item = "/1/2/3/4/5";
```

**Example 2:**

```php
$collection = collect([1,2,3,4,5]);

$item = $collection->reduce(function($previous, $item) {
        return $previous.'/'.$item;
}, 0);
```

**Result:**

```php
$item = "0/1/2/3/4/5";
```

**Notes:**

This method uses a callback function to iterate through the collection's items. The result of the first iteration is used as a basis for the next one and so on. This is how we concatenate the forward slash before any item for our first example. Since there is no previous result before we start the iterations with the first item we can use a default value to initialize our string result. This happens in our second example where we use zero before the first iteration takes place.

# reject()

Create a collection of all elements that do not pass a given truth test.

**Method:**

```php
public function reject($callback);
```

**Example 1:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->reject(function($item) {
        return $item > 3;
});
```

**Result:**

```php
$collection2->all() = [1,2,3];
```

**Example 2:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->reject(function($item) {
        return $item < 4;
});
```

**Result:**

```php
$collection2->all() = [3 => 4, 4 => 5];
```

**Example 3:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->reject(function($item) {
        return $item < 4;
})->values();
```

**Result:**

```php
$collection2->all() = [4,5];
```

**Notes:**

We use a callback function so we can filter our collection's items. The result is a collection with the items which fali to pass the test. Those items retain their keys so we can use the `values()` method to repopulate those.

# reverse()

Reverse items order.

**Method:**

```php
public function reverse();
```

**Example 1:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->reverse();
```

**Result:**

```php
$collection2->all() = [5,4,3,2,1];
```

**Example 2:**

```php
$collection1 = collect(['id' => 1, 'name' => 'John Doe', 'sex' => 1]);

$collection2 = $collection1->reverse();
```

**Result:**

```php
$collection2->all() = ['sex' => 1, 'name' => 'John Doe', 'id' => 1];
```

**Notes:**

This method reverses items order. If you use strings as keys then you get reverse alphabetical order.

## search()

Search the collection for a given value and return the corresponding key if successful.

**Method:**

```php
public function search($value, $strict = false);
```

**Example 1:**

```php
$collection = collect(['id' => 1, 'name' => 'John Doe', 'sex' => 1]);

$result = $collection->search(1);
```

**Result:**

```php
$result = 'id';
```

**Example 2:**

```php
$collection = collect(['id' => 1, 'name' => 'John Doe', 'sex' => 1]);

$collection->search('1', true);
```

**Result:**

```php
$result = false;
```

**Example 3:**

```php
$collection = collect(['id' => 1, 'name' => 'John Doe', 'sex' => 1]);

$result = $collection->search('1');
```

**Result:**

```php
$result = 'id';
```

**Notes:**

This method searches by default for the first key whose value is equal to the search value. If we set the second parameter to true then the method becomes more strict and searches for identical values. If the method doesn't find anything suitable returns false.

# shift()

Get and remove the first item from the collection.

**Method:**

```php
public function shift();
```

**Example 1:**

```
$collection = collect([1,2,3,4,5]);

$item = $collection->shift();
```

**Result:**

```
$item = 1;

$collection->all() = [2,3,4,5];
```

**Example 2:**

```
$collection = collect([1]);

$item = $collection->shift();
```

**Result:**

```
$item = 1;

$collection->all() =[];
```

**Notes:**

The first item is returned while it is removed from the collection too.

# shuffle()

Shuffle the items in the collection.

**Method:**

```
public function shuffle();
```

**Example:**

```
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->shuffle();
```

**Result:**

```
$collection2->all() = [2,1,5,3,4];
```

**Notes:**

The collection's items are placed in a random order.

## slice()

Slice the underlying collection array.

**Method:**

```
public function slice($offset, $length = null, $preserveKeys = false);
```

**Example 1:**

```
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->slice(2);
```

**Result:**

```
$collection2->all() = [3,4,5];
```

**Example 2:**

```
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->slice(2,2);
```

**Result:**

```
$collection2->all() = [3,4];
```

**Example 3:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->slice(2,2,true);
```

**Result:**

```php
$collection2->all() = [2 => 3, 3 => 4];
```

**Notes:**

This method helps us to create a sub-collection or even paginated results. Be careful because the third parameter helps us by resetting automatically new collections items keys. If it set to true though then the old keys are retained by the new collection's items.

## sort()

Sort through each item with a callback.

**Method:**

```php
public function sort(callable $callback);
```

**Example 1:**

```php
$collection1 = collect([9,12,125,19,123]);

$collection2 = $collection1->sort(function($b, $a) {
        return $b > $a;
});
```

**Result:**

```php
$collection2->all() = [0 => 9, 1 => 12, 3 => 19, 4 => 123, 2 => 125];
```

**Example 2:**

```php
$collection1 = collect([9,12,125,19,123]);

$collection2 = $collection1->sort(function($b, $a) {
        return $b > $a;
})->values();
```

**Result:**

```
$collection2->all() = [9,12,19,123,125];
```

**Example 3:**

```php
$collection1 = collect([9,12,125,19,123]);

$collection2 = $collection1->sort(function($b, $a) {
        return $b < $a;
})->values();
```

**Result:**

```
$collection2->all() = [125,123,19,12,9];
```

**Notes:**

This method uses `uasort()` so it needs a callback function to be used in order to sort the collection accordingly. If we don't use the `values()` function then the old keys are retained so i suppose we better use it.

# sortBy()

Sort the collection using the given callback.

**Method:**

```php
public function sortBy($callback, $options = SORT_REGULAR, $descending = false);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection2 = $collection1->sortBy('name');
```

**Result:**

```php
$collection2->all() = [
        2 => ['id' => 3, 'name' => 'Jack Doe'],
        1 => ['id' => 2, 'name' => 'Jane Doe'],
        0 => ['id' => 1, 'name' => 'John Doe']
];
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection2 = $collection1->sortBy('name')->values();
```

**Result:**

```php
$collection2->all() = [
        ['id' => 3, 'name' => 'Jack Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 1, 'name' => 'John Doe']
];
```

**Example 3:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection2 = $collection1->sortBy(function($item){
        return $item['name'];
})->values();
```

**Result:**

```php
$collection2->all() = [
        ['id' => 3, 'name' => 'Jack Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 1, 'name' => 'John Doe']
];
```

**Example 4:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection2 = $collection1->sortBy('name', SORT_REGULAR, true)->values();
```

**Result:**

```php
$collection2->all() = [
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
];
```

**Example 5:**

```
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection2 = $collection1->sortBy(function($item){
        return $item['name'];
}, SORT_REGULAR, true)->values();
```

**Result**:

```
$collection2->all() = [
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
];
```

**Notes**:

This method is useful for sorting a collection based on a callback function (examples 3,5) or a specific key (examples 1,2,4) in any order type. If you want updated keys for the returned collection you should use `values()` method. The `sortBy()` by default orders the collection items in ascending order but this can change if the third parameter is set to true. The second parameter is a sorting order flag and by default is used to compare the items normally.

More sorting type flags from php documentation[1]:

- SORT_REGULAR (integer): is used to compare items normally.
- SORT_NUMERIC (integer): is used to compare items numerically.
- SORT_STRING (integer): is used to compare items as strings.
- SORT_LOCALE_STRING (integer): is used to compare items as strings, based on the current locale.
- SORT_NATURAL (integer): is used to compare items as strings using "natural ordering" like natsort().
- SORT_FLAG_CASE (integer): can be combined (bitwise OR) with SORT_STRING or SORT_NATURAL to sort strings case-insensitively.

# sortByDesc()

Sort the collection in descending order using the given callback.

**Method**:

[1] http://php.net/manual/en/array.constants.php

```php
public function sortByDesc($callback, $options = SORT_REGULAR);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection2 = $collection1->sortByDesc('name');
```

**Result:**

```php
$collection2->all() = [
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
];
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection2 = $collection1->sortByDesc(function($item) {
        return $item['name'];
}, SORT_REGULAR)->values();
```

**Result:**

```php
$collection2->all() = [
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
];
```

**Notes:**

This method is based on `sortBy()` method by forcing its third parameter to true so only descending order sorting is returned. Same as before we can sort a collection by using a specific key or a callback. The `values()` method is needed so that the returned collections always reset their keys to consecutive integers. The second parameter is a sorting order flag and by default is used to compare the items normally. More about sorting flags you can check above on `sortBy()` method.

# splice()

Splice portion of the underlying collection array.

**Method:**

```
1 public function splice($offset, $length = 0, $replacement = []);
```

**Example 1:**

```
1 $collection1 = collect([1,2,3,4,5]);
2
3 $collection2 = $collection1->splice(1,2);
```

**Result:**

```
1 $collection1->all() = [1,4,5];
2
3 $collection2->all() = [2,3];
```

**Example 2:**

```
1 $collection1 = collect([1,2,3,4,5]);
2
3 $collection2 = $collection1->splice(1,2,[11,22]);
```

**Result:**

```
1 $collection1->all() = [1,11,22,4,5];
2
3 $collection2->all() = [2,3];
```

**Example 3:**

```php
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->splice(1,2,[11,22,33,44]);
```

**Result:**

```php
$collection1->all() = [1,11,22,33,44,4,5];

$collection2->all() = [2,3];
```

**Notes:**

This method is used to create a collection with some consecutive items and remove them form the collection too. If we pass a third parameter this is a replacement array of items. Those are inserted exactly where we removed the previous ones.

# sum()

Get the sum of the given values.

**Method:**

```php
public function sum($callback = null);
```

**Example 1:**

```php
$collection = collect([1,2,3,4,5]);

$sum = $collection->sum();
```

**Result:**

```php
$sum = 15;
```

**Example 2:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe', 'kids' => 2],                ['id' => 2, 'name' => 'Jane
e', 'kids' => 1],
        ['id' => 3, 'name' => 'Jack Doe', 'kids' => 3]
]);

$kids = $collection->sum('kids');
```

**Result:**

```
$kids = 6;
```

**Example 3:**

```
$collection = collect([1,2,3,4,5]);

$sum = $collection->sum(function($item) {
        return $item;
});
```

**Result:**

```
$sum = 15;
```

**Example 4:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe', 'kids' => 2],                ['id' => 2, 'name' => 'Jane
e', 'kids' => 1],
        ['id' => 3, 'name' => 'Jack Doe', 'kids' => 3]
]);

$kids = $collection->sum(function($item) {
        return $item['kids'];
});
```

**Result:**

```
$kids = 6;
```

**Notes:**

This method is very useful if we want to add a collections items. If we have a single array with numbers we don;t have to pass an argument. If we have a multidimensional array then we can pass the name of the key or we can use a callback function. A callback function can be also used when we have a simple array with numbers. You can use the most suitable of those for your needs.

# take()

Take a number of items form the top or the bottom of a collection.

**Method:**

```
public function take($limit = null);
```

**Example 1:**

```
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->take(2);
```

**Result:**

```
$collection2->all() = [1,2];
```

**Example 2:**

```
$collection1 = collect([1,2,3,4,5]);

$collection2 = $collection1->take(-2);
```

**Result:**

```
$collection2->all() = [4,5];
```

**Notes:**

This method can take an integer as a parameter that can be positive or negative. If it is positive then the items we take are from the collection's top and if not form the collection's bottom.

# toArray()

Get the collection of items as a plain array.

**Method:**

```
public function toArray();
```

**Example:**

```
$collection = collect([1,2,3,4,5]);

$array = $collection->toArray();
```

**Result:**

```
$array = [1,2,3,4,5];
```

# toJson()

Convert the collection to JSON format.

**Method:**

```
public function toJson($options = 0);
```

**Example:**

```
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$json = $collection->toJson();
```

**Result:**

```
$json = '[{"id":1,"name":"John Doe"},{"id":2,"name":"Jane Doe"},{"id":3,"name":"\
Jack Doe"}]';
```

**Notes:**

This method uses `json_encode()` and the $options parameter is a constant. Available parameters from php documentation[2]:

[2] http://php.net/manual/en/function.json-encode.php

- JSON_HEX_QUOT
- JSON_HEX_TAG
- JSON_HEX_AMP
- JSON_HEX_APOS
- JSON_NUMERIC_CHECK
- JSON_PRETTY_PRINT
- JSON_UNESCAPED_SLASHES
- JSON_FORCE_OBJECT
- JSON_PRESERVE_ZERO_FRACTION
- JSON_UNESCAPED_UNICODE

# transform()

Transform each item in the collection using a callback.

**Method**:

```
1 public function transform(callable $callback);
```

**Example 1**:

```
1 $collection = collect([1,2,3,4,5]);
2
3 $collection->transform(function($item) {
4         return $item * 2;
5 });
```

**Result**:

```
1 $collection->all() = [2,4,6,8,10];
```

**Example 2**:

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$collection->transform(function($item) {
        return ['id' => $item['id'], 'skill' => 'web developer'];
});
```

**Result:**

```php
$collection->all() = [
        ['id' => 1, 'skill' => 'web developer'],
        ['id' => 2, 'skill' => 'web developer'],
        ['id' => 3, 'skill' => 'web developer']
];
```

**Notes:**

This method uses a callback function to iterate through a collection's items and update the collection itself.

# unique()

Return only unique items from the collection array.

**Method:**

```php
public function unique();
```

**Example 1:**

```php
$collection1 = collect([1,2,1,4,5]);

$collection2 = $collection1->unique();
```

**Result:**

```
$collection2->all() = [0 => 1, 1 => 2, 3 => 4, 4 => 5];
```

**Example 2:**

```
$collection1 = collect([1,2,1,4,5]);

$collection2 = $collection1->unique()->values();
```

**Result:**

```
$collection2->all() = [1,2,4,5];
```

**Notes:**

This method returns a collection with the uniques items. If we need consecutive numbers as keys and not the old ones we have to use `values()` method.

# values()

Reset the keys on the underlying array to consecutive integers.

**Method:**

```
public function values();
```

**Example:**

```
$collection1 = collect([2 => 10, 'a' => 3, 1265 => 12]);

$collection2 = $collection1->values();
```

**Result:**

```
$collection2->all() = [10,3,12];
```

**Notes:**

This method uses `array_values()` function to reset an array's keys to consecutive integers. It is one of the most important methods offered as it can be chained after most of the other methods take place to reset returned collections keys.

# where()

Filter items by the given key value pair.

**Method:**

```php
public function where($key, $value, $strict = true);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->where('id', 1);
```

**Result:**

```php
$collection2->all() = [
        ['id' => 1, 'name' => 'John Doe']
];
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->where('id', '1', true);
```

**Result:**

```php
$collection2->all() = [];
```

**Example 3:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection->where('id', '2', false);
```

**Result:**

```
$collection2->all() = [
        1 => ['id' => 2, 'name' => 'Jane Doe']
];
```

**Example 4:**

```
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection->where('id', '2', false)->values();
```

**Result:**

```
$collection2->all() = [
        ['id' => 2, 'name' => 'Jane Doe']
];
```

**Notes:**

Be careful, there is a third parameter named `$strict` which is equal to true so by default the method tries to find for identical values and not just equal. If this changes then the method tries to find equal values. Also if you don't want the values of your new collection to retian their keys form their previous collection remember to chain `values()` method at the end of `where()` as we did on our fourth example.

# whereLoose()

Filter items by the given key value pair using loose comparison.

**Method:**

```
public function whereLoose($key, $value);
```

**Example 1:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->whereLoose('id', 1);
```

**Result:**

```
$collection2->all() = [
        ['id' => 1, 'name' => 'John Doe']
];
```

**Example 2:**

```php
$collection1 = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe']
]);

$collection2 = $collection1->whereLoose('id', '1');
```

**Result:**

```
$collection2->all() = [
        ['id' => 1, 'name' => 'John Doe']
];
```

**Notes:**

It is exactly the same method as before but in this case it doesn't accept a third parameter and returns results by searching only for equal values.

# __toString()

Convert the collection to its string representation.

**Method:**

```php
public function __toString();
```

**Example 1:**

```php
$collection = collect([1,2,3,4,5]);

$string = $collection->__toString();
```

**Result:**

```php
$string = '[1,2,3,4,5]';
```

**Example 2:**

```php
$collection = collect([
        ['id' => 1, 'name' => 'John Doe'],
        ['id' => 2, 'name' => 'Jane Doe'],
        ['id' => 3, 'name' => 'Jack Doe']
]);

$string = $collection->__toString();
```

**Result:**

```php
$string = '[{"id":1,"name":"John Doe"},{"id":2,"name":"Jane Doe"},{"id":3,"name"\
:"Jack Doe"}]';
```