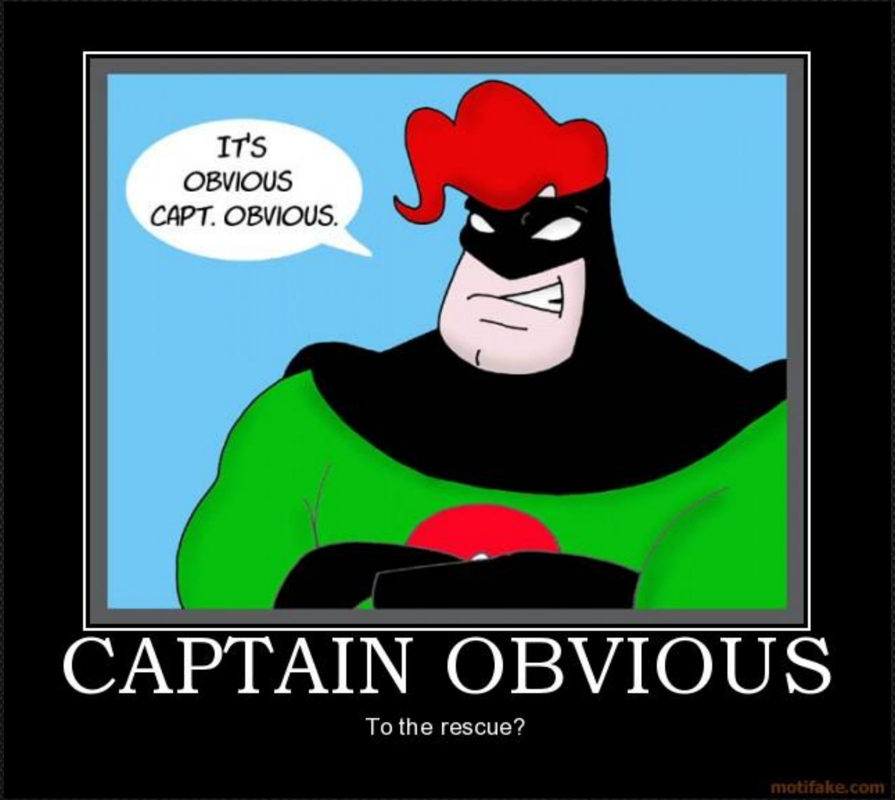
Array Tips

```
['Hip', 'Hip', 'Hip']
```

Benoit Viguier
@b_viguier

23/03/2016
AFUP Lyon

#0/3



It’s about array !

#1/3

Vanilla Php

#2/3

```php
$ids = [];
foreach($data as $d) {
    $ids[] = $d['id'];
}
```

```php
$ids = array_column(
    $data,
    'id'
);
```

Functional Oriented Programming

```php
$offset = 10;
$ids    = [];
foreach($data as $d) {
   $ids[] = $d['id'] + $offset;
}
```

```php
$offset = 10;
$ids = array_map(
   function($d) use($offset) {
       return $d['id'] + 10;
   },
   $data
);
```

No Anonymous Functions

#Bonus

Mind Blown

PARENTAL

ADVISORY

EXPLICIT PHP

#PHPorn

Data

```
$data = [
   ['id'=> 1, 'title'=> 'Hello', 'category_id'=> 1001],
   ['id'=> 2, 'title'=> 'World', 'category_id'=> 1001],
   ['id'=> 3, 'title'=> 'P h P', 'category_id'=> 1002],
   // ...
];
```

Select by ID

```php
$result = null;
foreach ($data as $d) {
    if ($d['id'] == $id) {
        $result = $d;
        break;
    }
}
```

```
array array_column ( array $input, mixed $column_key [, mixed $index_key = null ] )
: array
```

column_key

The column of values to return. This value may be an integer key of the column you wish to retrieve, or it may be a string key name for an associative array or property name.

It may also be **NULL** to return complete arrays or objects (this is useful together with index_key to reindex the array).

Select by ID

```
$result =
    array_column($data, null, 'id')[$id]
?? null;
```

Select by ID

```
$result =
    array_column($data, null, 'id')[$id]
?? null;
```

Select by ID

```
$result =
    array_column($data, null, 'id')[$id]
?? null;
```

Filter by category_id

```php
$results = [];
foreach ($data as $d) {
   if (in_array($d['category_id'], $category_ids)) {
       $results[] = $d;
       break;
   }
}
```

```
array array_intersect ( array $array1 , array $array2 [, array $... ] )
: array
```

array_intersect() returns an array containing all the values of `array1` that are present in all the arguments.

Note that keys are preserved.

Filter by category_id

```php
$results = array_intersect_key(
   $data,
   array_intersect(
       array_column($data, 'category_id'),
       $category_ids
   )
);
```

Filter by category_id

```
$results = array_intersect_key(
   $data,
   array_intersect(
       array_column($data, 'category_id'),
       $category_ids
   )
);
```

Filter by category_id

```
$results = array_intersect_key(
   $data,
   array_intersect(
       array_column($data, 'category_id'),
       $category_ids
   )
);
```

Filter by category_id

```
$results = array_intersect_key(
   $data,
   array_intersect(
       array_column($data, 'category_id'),
       $category_ids
   )
);
```

Data

```php
$data = [
    'bool' => [
        'must_not' => null,
        'must'     => [
            'term' => ['title' => 'afup'],
        ],
        'should'   => [
            ['term' => ['category' => 'talk']],
            null,
            ['term' => ['location' => 'lyon']],
        ],
    ],
];
```

Data

```
$data = [
    'bool' => [
        'must_not' => null,
        'must'     => [
            'term' => ['title' => 'afup'],
        ],
        'should'   => [
            ['term' => ['category' => 'talk']],
            null,
            ['term' => ['location' => 'lyon']],
        ],
    ],
];
```

Data

```php
$data = [
    'bool' => [
        'must_not' => null,
        'must'     => [
            'term' => ['title' => 'afup'],
        ],
        'should'   => [
            0 => ['term' => ['category' => 'talk']],
            1 => null,
            2 => ['term' => ['location' => 'lyon']],
        ],
    ],
];
```

Filter NULL and reindex integer keys

```php
function jsonFilter($a)
{
    if (!is_array($a)) {
        return $a;
    }
    $has_numerical_keys = true;
    $result = [];
    foreach ($a as $key => $value) {
        $has_numerical_keys = $has_numerical_keys && is_integer($key);
        if ($value) {
            $result[$key] = $value;
        }
    }
    $result = $has_numerical_keys ? array_values($result) : $result;

    return array_map(__FUNCTION__, $result);
}
```

```
array array_filter ( array $array [, callable $callback [, int $flag = 0 ]] ) : array
```

callback

The callback function to use

If no callback is supplied, all entries of array equal to **FALSE** (see converting to boolean) will be removed.

```
array array_merge ( array $array1 [, array $... ] )
: array
```

If the input arrays have the same string keys, then the later value for that key will overwrite the previous one. If, however, the arrays contain numeric keys, the later value will *not* overwrite the original value, but will be appended.

Values in the input array with numeric keys will be renumbered with incrementing keys starting from zero in the result array.

Filter NULL and reindex integer keys

```
function jsonFilter($a)
{
   return is_array($a) ?
        array_map(
            __FUNCTION__,
            array_merge(array_filter($a))
        )
        : $a;
}
```

Filter NULL and reindex integer keys

```
function jsonFilter($a)
{
   return is_array($a) ?
         array_map(
             __FUNCTION__,
             array_merge(array_filter($a))
         )
         : $a;
}
```

Filter NULL and reindex integer keys

```
function jsonFilter($a)
{
   return is_array($a) ?
       array_map(
           __FUNCTION__,
           array_merge(array_filter($a))
       )
       : $a;
}
```

Filter NULL and reindex integer keys

```php
function jsonFilter($a)
{
   return is_array($a) ?
        array_map(
            __FUNCTION__,
            array_merge(array_filter($a))
        )
        : $a;
}
```

Filter NULL and reindex integer keys

```php
function jsonFilter($a)
{
   return is_array($a) ?
       array_map(
           __FUNCTION__,
           array_merge(array_filter($a))
       )
       : $a;
}
```

Data

```
$data = [
    ['A1', 'A2', 'A3'],
    ['B1'],
    [],
    ['D1', 'D2', 'D3', 'D4'],
    // ...
];
// Expecting
['A1','B1','D1','A2','D2','A3','D3','D4'];
```

Mixing elements of several arrays

```
$result = [];
do {
    $modified = false;
    foreach ($data as $key => $d) {
        if (empty($d)) {
            unset($data[$key]);
            continue;
        }
        $result[] = array_pop($d);
        $modified = true;
    }
} while ($modified);
```

```
array array_map ( callable $callback , array $array1 [, array $... ] )
: array
```

An interesting use of this function is to construct an array of arrays, which can be easily performed by using **NULL** as the name of the callback function

Example #4 Creating an array of arrays

…

Mixing elements of several arrays

```
$result = array_filter(
    array_merge(
        ...array_map(null, ...$data)
    )
);
```

Mixing elements of several arrays

```
$result = array_filter(
    array_merge(
        ...array_map(null, ...$data)
    )
);
```

Mixing elements of several arrays

```php
$result = array_filter(
    array_merge(
        ...array_map(null, ...$data)
    )
);
```

Mixing elements of several arrays

```
$result = array_filter(
    array_merge(
        ...array_map(null, ...$data)
    )
);
```

Merci

Playing with objects...

```
$objects = [];
foreach ($ids as $id) {
   $objects[] = new Video($id);
}

$titles = [];
foreach ($objects as $obj) {
   $titles[] = $obj->getTitle();
}
```

... and reflection

```php
$objects = array_map(
   [new \ReflectionClass(Video::class), 'newInstance'],
   $ids
);

$titles = array_map(
   [
       new \ReflectionMethod(Video::class, 'getTitle'),
      'invoke'
   ],
   $objects
);
```

Boolean operations on arrays

- `array_intersect, array_intersect_keys`
- `array_diff, array_diff_keys`
- ~~`array_union`~~

and…

- `array_unique`
- `array_merge`
- `+ operator`