Arrays and Array Functions

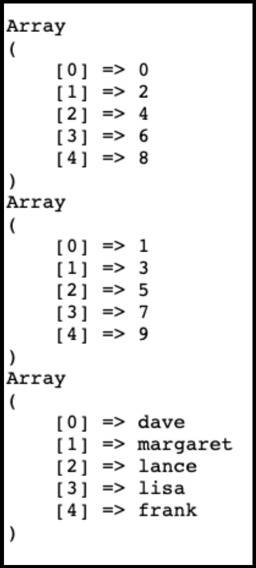
Intro PHP

Download the code from Nexus

Naturally indexed arrays

By default, arrays are naturally indexed with integers that autoincrement as new elements are added.

```
$even = [0,2,4,6,8];
$odd = array(1,3,5,7,9);
$users = [];
$users[] = 'dave';
$users[] = 'margaret';
$users[] = 'lance';
$users[] = 'lisa';
$users[] = 'frank';
```



```
Array
(
    [0] => 0
    [1] => 2
    [2] => 4
    [3] => 6
    [4] => 8
)
Array
(
    [0] => 1
    [1] => 3
    [2] => 5
    [3] => 7
    [4] => 9
)
Array
(
    [0] => dave
    [1] => margaret
    [2] => lance
    [3] => lisa
    [4] => frank
)
```

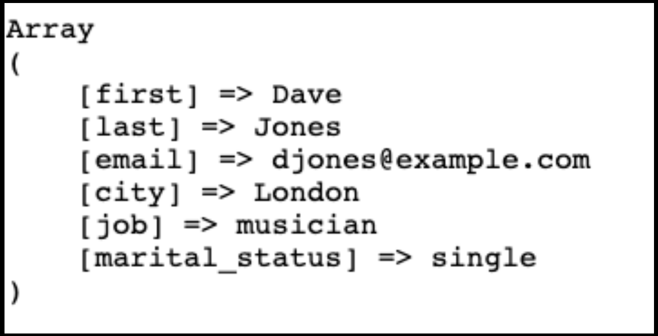
01_indices1.php

PHP arrays aren’t real arrays. They are collections of key/value pairs. They do not require contiguous memory addresses for each element as in C.

Associative Arrays

Associative arrays in PHP behave like arrays and have all the expected array functionality.

```
$user['first'] = 'Dave';
$user['last'] = 'Jones';
$user['email'] = 'djones@example.com';
$user['city'] = 'London';
$user['job'] = 'musician';
$user['marital_status'] = 'single';
```



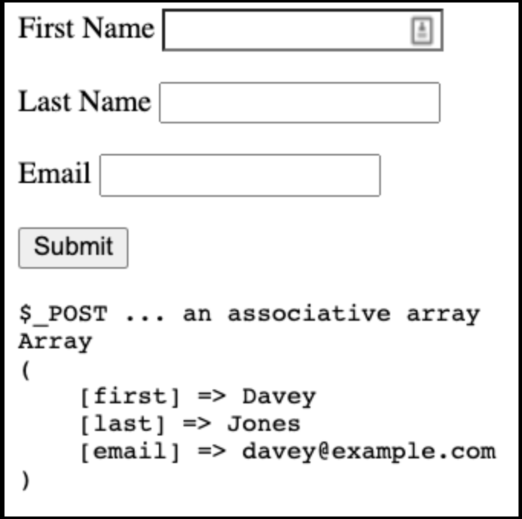
```
Array
(
    [first] => Dave
    [last] => Jones
    [email] => djones@example.com
    [city] => London
    [job] => musician
    [marital_status] => single
)
```

02_indices2.php

SuperGlobal Arrays

Submitted form data is accessible in one of three SuperGlobal associative arrays. $_POST, $_GET, or $_REQUEST

```
<form method="post">
    <input id="first" name="first" />
    <input id="last" name="last" />
    <input id="email" name="email" />
    <input type="submit">
</form>
```



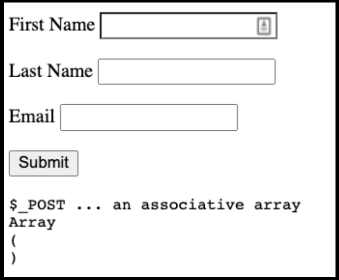
First Name

Last Name

Email

Submit

```
$_POST ... an associative array
Array
(
    [first] => Davey
    [last] => Jones
    [email] => davey@example.com
)
```



First Name

Last Name

Email

Submit

```
$_POST ... an associative array
Array
(
)
```

Forms can submit using the GET or POST methods only. Data will be available in the $_GET or $_POST SuperGlobals respectively.

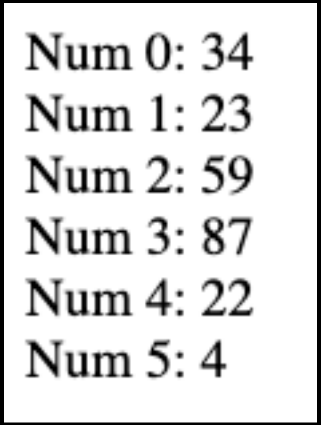
03_indices3.php

Accessing array elements

Array elements can be accessed by using their index.

```
$nums = [34,23,59,87,22,4];
```

```
Num 0: <?php echo $nums[0]; ?> <br />
Num 1: <?php echo $nums[1]; ?> <br />
Num 2: <?php echo $nums[2]; ?> <br />
Num 3: <?php echo $nums[3]; ?> <br />
Num 4: <?php echo $nums[4]; ?> <br />
Num 5: <?php echo $nums[5]; ?>
```



```
Num 0: 34
Num 1: 23
Num 2: 59
Num 3: 87
Num 4: 22
Num 5: 4
```

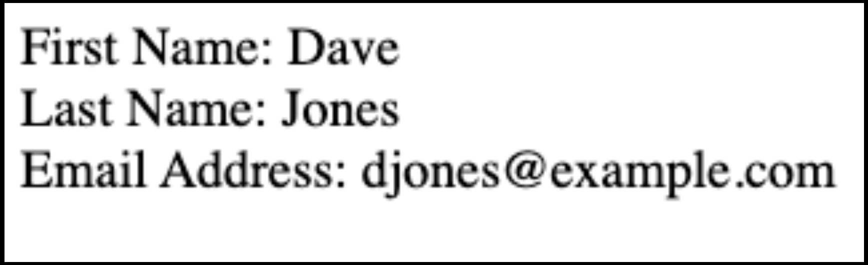
04_elements1.php

Accessing array elements

Array elements can be accessed by using their index.

```
<?php
$user['first'] = 'Dave';
$user['last'] = 'Jones';
$user['email'] = 'djones@example.com';
?>

<p>
First Name: <?php echo $user['first'] ?><br />
Last Name: <?php echo $user['last'] ?><br />
Email Address: <?php echo $user['email'] ?>
</p>
```



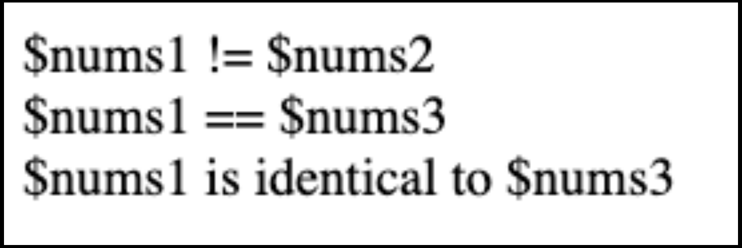
```
First Name: Dave
Last Name: Jones
Email Address: djones@example.com
```

05_elements2.php

Comparing arrays

Arrays can be compared using standard equality operators.

```
$nums1 = [1,2,3,4];
$nums2 = [4,3,2,1];
$nums3 = [1,2,3,4];
```



```
$nums1 != $nums2
$nums1 == $nums3
$nums1 is identical to $nums3
```

When comparing arrays, the values and order matter.

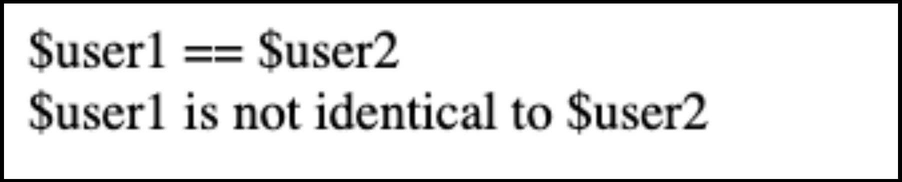
06_comparison1.php

Comparing associative arrays

Arrays can be compared using standard equality operators.

```
$user1 = [
'first' => 'Tom',
'last' => 'Jones',
'email' => 'tom@hotmail.com'
];

$user2 = [
'last' => 'Jones',
'email' => 'tom@hotmail.com',
'first' => 'Tom'
];
```



```
$user1 == $user2
$user1 is not identical to $user2
```

When comparing associative arrays, the values and order of the keys matter.

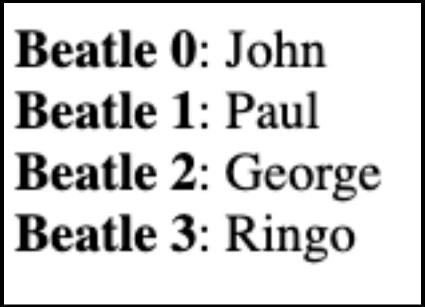
07_comparison2.php

Iterating arrays using loops

The easiest way to access all array elements is with a loop.

```
$band = ['John','Paul','George','Ringo'];


for($i=0;$i<sizeof($band);$i++) {
    echo "<strong>Beatle $i</strong>: {$band[$i]}<br
/>";
}
```


Beatle 0: John
Beatle 1: Paul
Beatle 2: George
Beatle 3: Ringo

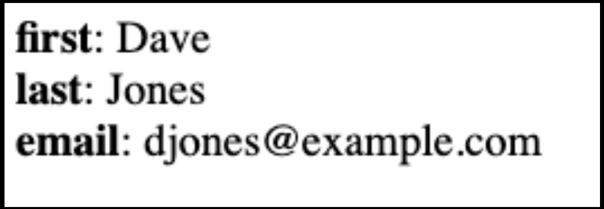
08_looping1.php

Iterating arrays using loops

the foreach loop is most commonly used for iterating over arrays.

```
$user['first'] = 'Dave';
$user['last'] = 'Jones';
$user['email'] = 'djones@example.com';

foreach($user as $key => $value) {
    echo "<strong>$key</strong>: $value <br />";
}
```



first: Dave
last: Jones
email: djones@example.com

Note that the foreach loop allows us to access both the key and the value of each element, making it particularly useful for working with associative arrays.

09_looping2.php

Looping through multiple arrays

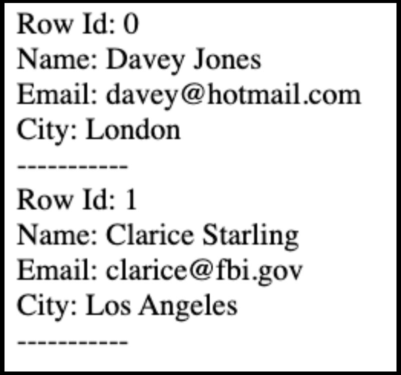
Loops can be used to loop through arrays that contain other arrays.

```
$user1 = [
'name' => 'Davey Jones',
'email' => 'davey@hotmail.com',
'city' => 'London'
];

$user2 = [
'name' => 'Clarice Starling',
'email' => 'clarice@fbi.gov',
'city' => 'Los Angeles'
];


$users = [$user1, $user2];
```

```
foreach($users as $key => $user) {
echo 'Row Id: ' . $key . '<br />';
echo 'Name: ' . $user['name'] . '<br />';
echo 'Email: ' . $user['email'] . '<br />';
echo 'City: ' . $user['city'] . '<br />';
echo '-----------<br />';
}
```



```
Row Id: 0
Name: Davey Jones
Email: davey@hotmail.com
City: London
-----------
Row Id: 1
Name: Clarice Starling
Email: clarice@fbi.gov
City: Los Angeles
-----------
```

10_looping3.php

Note that in this case the $key refers to the outer array... the array containing the inner arrays (the $value). It is naturally indexed with integers.

Looping over nested arrays

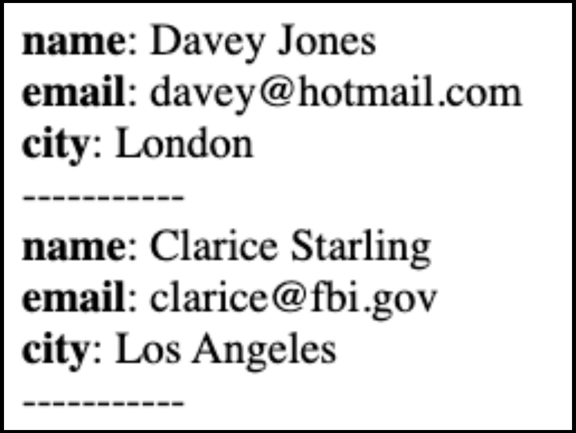
An outer loop and an inner loop can access every array in this collection.

```
$user1 = [
'name' => 'Davey Jones',
'email' => 'davey@hotmail.com',
'city' => 'London'
];

$user2 = [
'name' => 'Clarice Starling',
'email' => 'clarice@fbi.gov',
'city' => 'Los Angeles'
];

$users = [$user1, $user2];
```

```
foreach($users as $user) {
  foreach($user as $key => $value ) {
    echo "<strong>$key</strong>: $value <br />";
  }
    echo '-----------<br />';
}
```



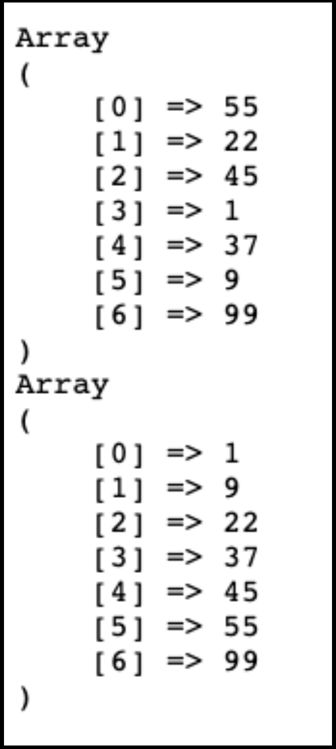
```
name: Davey Jones
email: davey@hotmail.com
city: London
-----------
name: Clarice Starling
email: clarice@fbi.gov
city: Los Angeles
-----------
```

11_looping4.php

Sorting arrays using sort()

Sort() orders the array values, but does not maintain key/value pairings.

```
$nums = [55,22,45,1,37,9,99];
print_r($nums);
sort($nums);
print_r($nums);
```



```
Array
(
        [0] => 55
        [1] => 22
        [2] => 45
        [3] => 1
        [4] => 37
        [5] => 9
        [6] => 99
)
Array
(
        [0] => 1
        [1] => 9
        [2] => 22
        [3] => 37
        [4] => 45
        [5] => 55
        [6] => 99
)
```

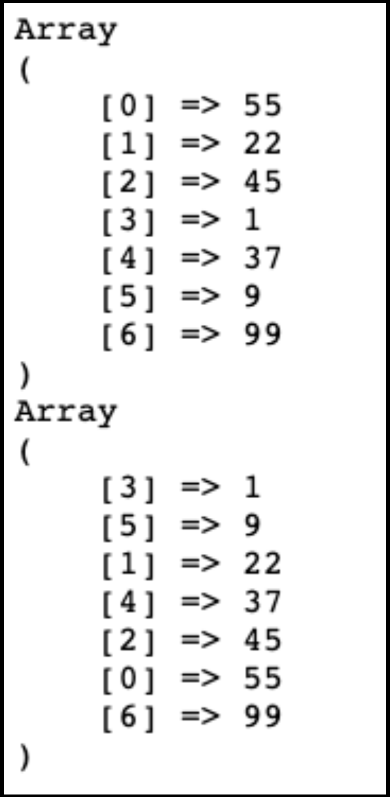
Most PHP array sorting functions work on a reference to the original array... meaning that the original array is modified by the function.

13_sorting2.php

Sorting arrays using asort()

asort() orders the array values and maintains key/value pairings. Best used for associative arrays.

```
$nums = [55,22,45,1,37,9,99];
print_r($nums);
asort($nums);
print_r($nums);
```



```
Array
(
    [0] => 55
    [1] => 22
    [2] => 45
    [3] => 1
    [4] => 37
    [5] => 9
    [6] => 99
)
Array
(
    [3] => 1
    [5] => 9
    [1] => 22
    [4] => 37
    [2] => 45
    [0] => 55
    [6] => 99
)
```

Most PHP array sorting functions work on a reference to the original array... meaning that the original array is modified by the function.

14_sorting3.php

Counting array elements

The count() function returns the number of elements in an array. The sizeof() function is simply an alias of count()

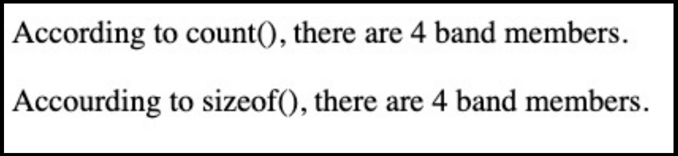
```
$band = ['John', 'Paul', 'George', 'Ringo'];

$num1 = count($band);

$num2 = sizeof($band);

echo "<p>According to count(),
    there are $num1 band members.</p>";

echo "<p>Accourding to sizeof(),
    there are $num2 band members.</p>";
```



```
According to count(), there are 4 band members.

Accourding to sizeof(), there are 4 band members.
```

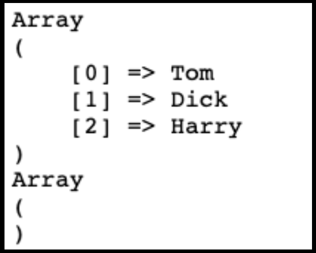
16_counting.php

Push and Pop

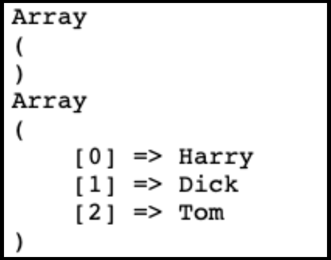
Remember that PHP array functions are procedural. They act on arrays... they are not array methods as in Javascript.

```
$old = ['Tom', 'Dick','Harry’];
$new = [];

while($current = array_pop($old)) {
     array_push($new, $current);
}
```



```
Array
(
    [0] => Tom
    [1] => Dick
    [2] => Harry
)
Array
(
)
```



```
Array
(
)
Array
(
    [0] => Harry
    [1] => Dick
    [2] => Tom
)
```

We can also use literal push syntax to add an element to the end of an array

```
$new[] = 'Dave';
```

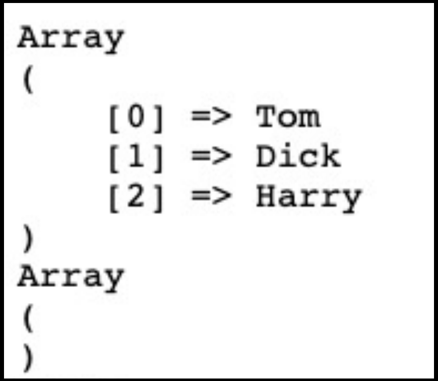
17_push_pop.php

Shift and Unshift

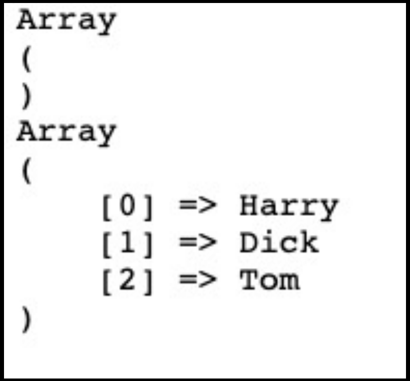
Remember that PHP array functions are procedural. They act on arrays... they are not array methods as in Javascript.

```
$old = ['Tom', 'Dick', 'Harry'];
$new = [];


while($current = array_shift($old)) {
    array_unshift($new, $current);
}
```



```
Array
(
    [0] => Tom
    [1] => Dick
    [2] => Harry
)
Array
(
)
```



```
Array
(
)
Array
(
    [0] => Harry
    [1] => Dick
    [2] => Tom
)
```

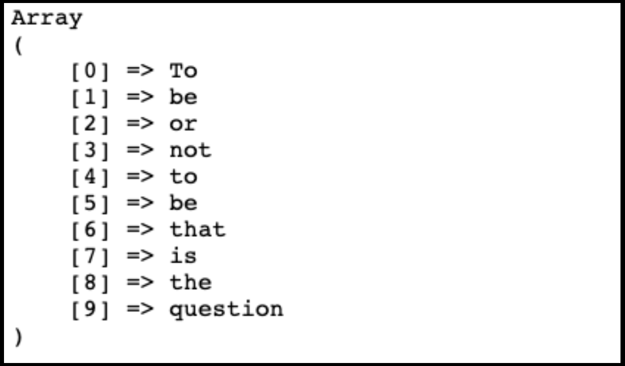
18_shift_unshift.php

String to Array, Array to String

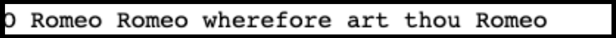
PHP has two simple functions to convert arrays to strring, and to convert strings to arrays.

```
$string = 'To be or not to be that is the question';
$array = ['O', 'Romeo', 'Romeo', 'wherefore', 'art', 'thou', 'Romeo'];

$new_array = explode(' ', $string);
$new_string = implode(' ', $array);
```



```
Array
(
    [0] => To
    [1] => be
    [2] => or
    [3] => not
    [4] => to
    [5] => be
    [6] => that
    [7] => is
    [8] => the
    [9] => question
)
```



```
O Romeo Romeo wherefore art thou Romeo
```

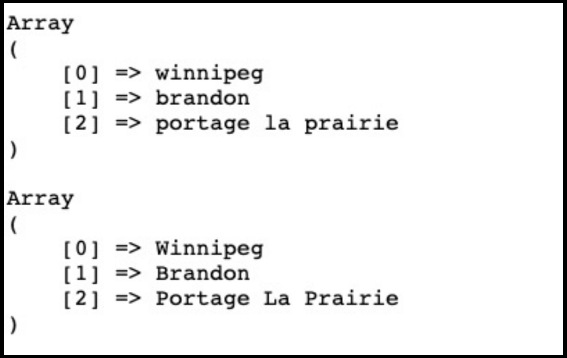
19_conversion.php

Mapping array elements

The array_map() function applies a method (strategy) to all elements of an array, creating a new array.

```
$cities = ['winnipeg', 'brandon', 'portage la prairie'];

$capitalized = array_map(function($el) {
   return ucwords($el);
}, $cities);
```



```
Array
(
    [0] => winnipeg
    [1] => brandon
    [2] => portage la prairie
)

Array
(
    [0] => Winnipeg
    [1] => Brandon
    [2] => Portage La Prairie
)
```

In most cases, you can accomplish the same effect using a loop. Filter, however, is more elegant, and takes less code to write!

21_mapping.php

Filtering arrays

The array_filter() function returns elements based on a condition, creating a new array.

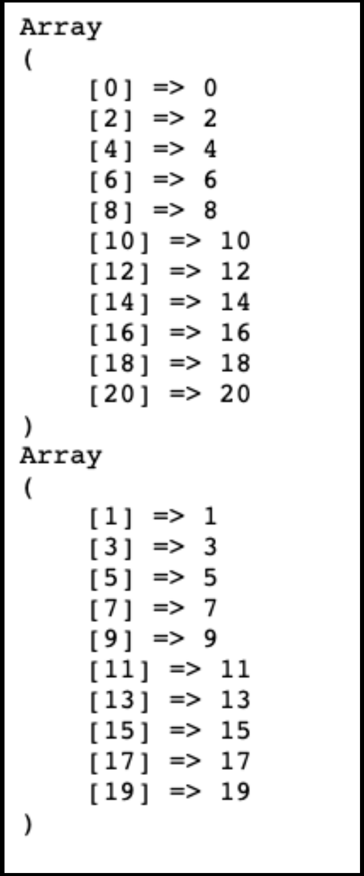
```php
$nums = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];

$even = array_filter($nums, fn($el) => $el % 2 == 0);

$odd = array_filter($nums, fn($el) => $el % 2 != 0);
```

In most cases, you can accomplish the same effect using a loop. Filter, however, is more elegant, and takes less code to write!

22_filtering.php



```
Array
(
    [0] => 0
    [2] => 2
    [4] => 4
    [6] => 6
    [8] => 8
    [10] => 10
    [12] => 12
    [14] => 14
    [16] => 16
    [18] => 18
    [20] => 20
)
Array
(
    [1] => 1
    [3] => 3
    [5] => 5
    [7] => 7
    [9] => 9
    [11] => 11
    [13] => 13
    [15] => 15
    [17] => 17
    [19] => 19
)
```

Filtering arrays

The array_filter() function returns elements based on a condition, creating a new array.

Note: you don't need to use fat arrow functions

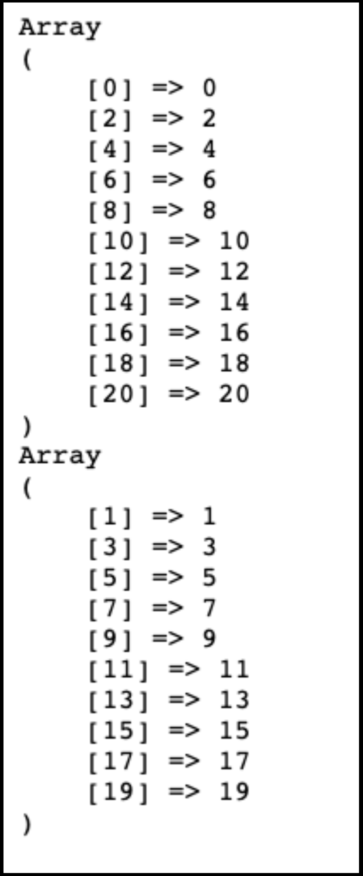
```php
$nums = [0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20];

$even = array_filter($nums, function($el) {
   return $el % 2 == 0;
});

$odd = array_filter($nums, function($el) {
   return $el % 2 != 0;
});
```

In most cases, you can accomplish the same effect using a loop. Filter, however, is more elegant, and takes less code to write!

22_filtering.php



```
Array
(
    [0] => 0
    [2] => 2
    [4] => 4
    [6] => 6
    [8] => 8
    [10] => 10
    [12] => 12
    [14] => 14
    [16] => 16
    [18] => 18
    [20] => 20
)
Array
(
    [1] => 1
    [3] => 3
    [5] => 5
    [7] => 7
    [9] => 9
    [11] => 11
    [13] => 13
    [15] => 15
    [17] => 17
    [19] => 19
)
```

Example: Shuffling an array

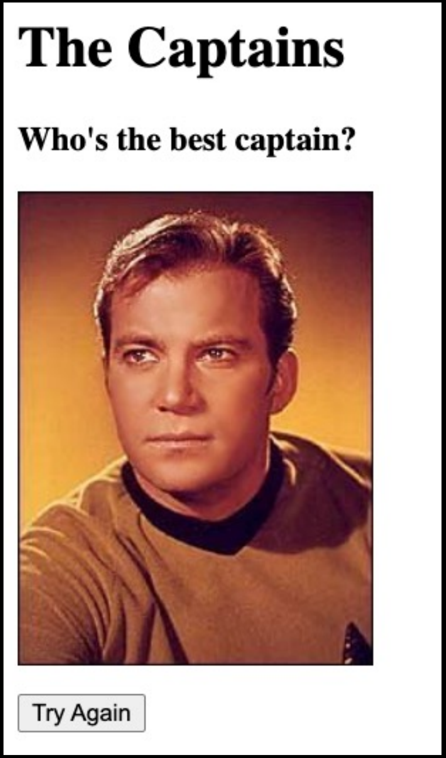
The shuffle() method randomizes the order of elements in an array...

```
$captains = [
  "kirk.jpg",
  "picard.jpg",
  "janeway.jpg",
  "archer.jpg"
];

shuffle($captains);
```

Note: shuffle() does not produce true randomness and should not be used in any situation where true randomness is required.

23_the_captains.php



The Captains

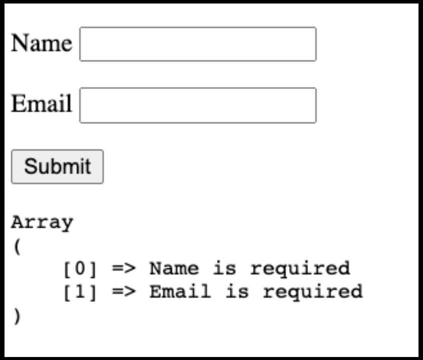
Who's the best captain?

Example: Counting errors

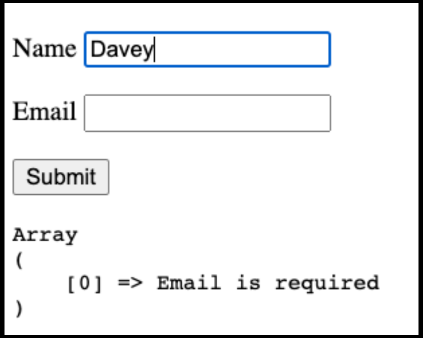
Using the count method, we can determine if there are any errors after a form submission:

```
$errors = []; // count 0

if(empty($_POST['name'])) {
    $errors[] = 'Name is required’;
  }
  if(empty($_POST['email'])) {
    $errors[] = 'Email is required’;
  }
   // Test for errors
  if(count($errors) == 0) {
    print_r($_POST);
  } else {
    print_r($errors);
 }
```



```
Array
(
    [0] => Name is required
    [1] => Email is required
)
```



```
Array
(
    [0] => Email is required
)
```

24_counting_errors.php

Next:

Forms