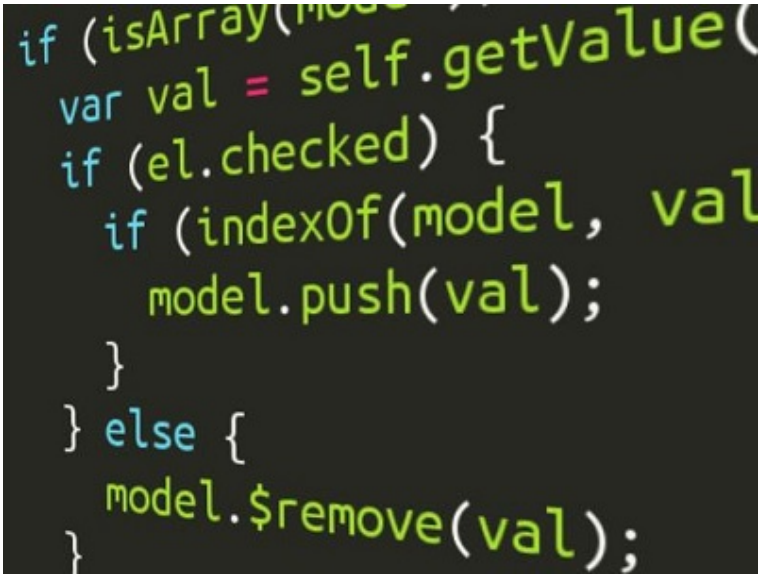
Most Useful JavaScript Array Functions (http://vegibit.com/most-useful-javascript-array-functions/)



(http://vegibit.com/most-useful-javascript-array-functions/)

In this JavaScript Tutorial, we'll take a look at the Most Useful JavaScript Array Functions. For this tutorial, we took the source code from the pros and put it all in one bucket. The source code from the pros would be the JavaScript that powers the most popular JavaScript frameworks today, such as Angular, React, Vue, Mithril, and countless more. Once we have all of this source code in the same file, we check for the number of times each array function was made use of in the code. Of course JavaScript array functions that had a very high number of calls to them are featured first, as this would suggest that those particular functions are very useful to the pros. Array functions that were not used all that often are placed lower on the list. Let's see which ones made it to the top of our most useful JavaScript array functions list.

Array.prototype.filter()	
Array.prototype.join()	7
Array.prototype.splice()	8
Array.prototype.forEach()	9
Array.prototype.concat()	10
Array.prototype.shift()	11
Array.prototype.unshift()	12
Array.prototype.map()	13
Array.prototype.sort()	14
Array.prototype.pop()	15
Array.prototype.reduce()	16
Array.prototype.some()	17
Array.prototype.lastIndexOf()	18
Array.prototype.reduceRight()	19
Array.prototype.every()	20
Array.prototype.reverse()	21

1. What Are JavaScript Arrays?

In JavaScript, an Array represents an ordered collection of values. The array consists of one or more elements, each having a numbered position inside of the array. This is also called the index. Any given array element can hold any

type offered by the JavaScript language. In other words, you can mix and match the various types inside of the same array. Just like most

match the various types inside of the same array. Just like most programming languages, arrays in JavaScript are zero based. This means they start at 0 and move up from there. Arrays in JavaScript grow and shrink like magic. That is to say, you do not have to specify the array size ahead of time before placing values in the array. As you add elements, the array dynamically grows for you. The same happens when you remove elements. All JavaScript arrays have a link to Array.prototype (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/prototype) which is how we get access to all of the very useful functions we'll cover here in this tutorial. In true JavaScript style, these functions work not only on any array, but also any "array-like" object.

---

2. Array.prototype.push()

```
array.push(element1, ..., elementN)
```

Overview

The `push()` function appends the arguments given to it, in order, to the end of the array. It modifies the array directly, rather than creating a new array.

What push() returns

When you make a call to `push()`, it returns the total number of items in the array, after any provided arguments have been added to the end of the array.

array.push() function examples

```
var characters = ['Darth Vader', 'Yoda', 'Boba Fett'];
var total = characters.push('Luke Skywalker', 'Han Solo');

console.log(characters); // ["Darth Vader", "Yoda", "Boba Fett", "Luke Skywalker", "
console.log(total);      // 5
```

---

3. Array.prototype.indexOf()

```
array.indexOf(searchElement[, fromIndex = 0])
```

Overview

The JavaScript `indexOf()` function searches an array for an element that contains a specific value, then returns the very first index that contains that value. You can provide a starting index to the function if you like, otherwise searching will begin from index 0.

What indexOf() returns

The lowest index that is greater than or equal to the start of the array at which the element is equal to the given value. If no match is found, the `indexOf()` function returns -1.

array.indexOf() function examples

```
var starwars = ['Episode 4: A New Hope',
   'Episode 5: The Empire Strikes Back',
   'Episode 6: Return of the Jedi',
   'Episode 1: The Phantom Menace',
   'Episode 2: Attack of the Clones',
   'Episode 3: Revenge of the Sith',
   'Episode 7: The Force Awakens'];

console.log(starwars.indexOf('Episode 7: The Force Awakens'));          // 6
console.log(starwars.indexOf('Episode 3: Revenge of the Sith'));        // 5
console.log(starwars.indexOf('Episode 2: Attack of the Clones', 3));    // 4
console.log(starwars.indexOf('Episode 1: The Phantom Menace', -4));  // 3
```

```
var locatedat = [];
var enemies = ['Kylo Ren', 'Darth Vader', 'Storm Trooper', 'General Hux', 'Emperor P
var enemy = 'Storm Trooper';
var enemylocation = enemies.indexOf(enemy);
while (enemylocation != -1) {
   locatedat.push(enemylocation);
   enemylocation = enemies.indexOf(enemy, enemylocation + 1);
}
console.log(locatedat); //  2, 5
```

The example above is a little more tricky. We have an array of enemies, and there might be multiple of a given enemy. By making use of `indexOf()`, we can find all instances of 'Storm Trooper' in our array of enemies. Running the

code shows us that we have found a Storm Trooper at index 2 and index 5 of our array.

```
function theGoodSide(goodguys, goodguy) {
    if (goodguys.indexOf(goodguy) === -1) {
        goodguys.push(goodguy);
        console.log('The new goodguys are ' + goodguys);
    } else if (goodguys.indexOf(goodguy) > -1) {
        console.log(goodguy + ' is already part of the goodguys.');
    }
}

var thegoodguys = ['Luke Skywalker', 'Han Solo', 'Princess Leia'];

theGoodSide(thegoodguys, 'Han Solo');  // Han Solo is already part of the goodguys
theGoodSide(thegoodguys, 'Yoda');  //  The new goodguys are Luke Skywalker,Han
theGoodSide(thegoodguys, 'Ben Kenobi');  //  The new goodguys are Luke Skywalke
theGoodSide(thegoodguys, 'R2-D2');  //  The new goodguys are Luke Skywalker,Har
theGoodSide(thegoodguys, 'Luke Skywalker');  //  Luke Skywalker is already part of
```

This example makes use of a custom function that we can use to make sure our good guys are all accounted for. We check to see if the character exists in our collection, and if not, we add them to our good side.

---

4. Array.prototype.slice()

```
array.slice([begin[, end]])
```

Overview

The JavaScript `slice()` function takes a start and end argument. It uses these start and end points to reach into the array, and remove a specific number of elements. The position you reference by start is included in the results, but the end is not included. In other words all elements are returned

from the start right up to, but not including the end element. If you do not provide an end value, `slice()` will return the rest of the array starting from the given starting point. The original array is left in tact.

What slice() returns

The `slice()` function returns a new array that contains all of the elements of the original array from the element specified by the starting point provided, and up to but not including, the element specified by the ending point provided.

array.slice() function examples

```
var vaderparts = ['helmet', 'left arm', 'right arm', 'left leg', 'right leg'];
var chopped = vaderparts.slice(2,3);

console.log(chopped); // right arm
```

5. Array.prototype.toString()

```
array.toString()
```

The `toString()` function converts all elements in an array to strings and outputs one big string as a comma separated list of items.

array.toString() function examples

```
var spacecraft = ['X Wing', 'Death Star', 'Millenium Falcon', 'Jedi Interceptor'];
var allspacecraft = spacecraft.toString();

console.log(allspacecraft);
//  X Wing,Death Star,Millenium Falcon,Jedi Interceptor
```

6. Array.prototype.filter()

```
array.filter(callback[, thisArg])
```

Overview

The JavaScript `filter()` function is very useful for doing exactly what it says, filtering down a collection of elements based on a given test. When you call the `filter()` function, you need to pass it a callback. This callback is executed against every element in the array. If that callback results in a `true` value, that particular element is added to a new array. The original array is

value, that particular element is added to a new array. The original array is left unchanged.

What filter() returns

`filter()` returns a new array that contains only the elements of the original array that returned true when the provided callback ran.

array.filter() function examples

```
var summer = [10, 20, 30, 40, 50, 60, 70, 80].filter(function niceweather(tempera
    return temperature >= 70;
});
console.log(summer);  //  70, 80
```

```
var spacecraft = [
    {name: 'Tie Fighter', speed: 200000},
    {name: 'Super Star Destroyer', speed: 300000},
    {name: 'Death Star', speed: 400000},
    {name: 'T65 X Wing Star fighter', speed: 500000},
    {name: 'T47 Air (snow) Speeder', speed: 600000},
    {name: '74-Z Speeder Bike', speed: 700000},
    {name: 'Millenium Falcon', speed: 800000}
];

function filterBySpeed(spacecraft) {
    return spacecraft.speed > 450000;
}

var fleet = spacecraft.filter(filterBySpeed);

fleet.forEach(function (craft) {
    console.log(craft.name);
});

// T65 X Wing Star fighter
// T47 Air (snow) Speeder
// 74-Z Speeder Bike
// Millenium Falcon
```

7. Array.prototype.join()

```
string = array.join([separator = ',])
```

Overview

The `join()` function converts each element of an array to a string and then

The `join()` function converts each element of an array to a string and then concatenates those strings together. If the separator string was provided, it gets inserted between all elements in the final string.

What join() returns

The string that results from converting each element of the original array to a string and then joining them together, with the separator string between elements.

array.join() function examples

```
var wisdom = ['Yoda Says', 'Fear is the path to the dark side. Fear leads to anger.
wisdom = wisdom.join(': ');
console.log(wisdom);

// Yoda Says: Fear is the path to the dark side. Fear leads to anger. Anger leads t
```

```
var wisdom = ['Do. Or do not. There is no try', 'Luminous beings are we...not this c
var wisdom = wisdom.join(' <=***=> ');
console.log(wisdom);

// Do. Or do not. There is no try <=***=> Luminous beings are we...not this crude
```

```
years = [1977, 1980, 1983, 1999, 2002, 2005, 2015];
years = years.join(', ');

console.log('New Star Wars movies came out in ' + years);
// New Star Wars movies came out in 1977, 1980, 1983, 1999, 2002, 2005, 2015
```

8. Array.prototype.splice()

```
array.splice(start, deleteCount[, item1[, item2[, ...]]])
```

Overview

The `splice()` function deletes zero or more elements beginning from the provided start location, and replaces those elements with zero or more new values that were provided. Existing items in the array are shifted as needed

to make room for any added or deleted elements. The thing to remember about the `splice()` function, is that it **does** *modify the original array directly*.

Returns

An array which contains any elements that may have been deleted from the array.

array.splice() function examples

```
var darkSide = ['Darth Vader', 'Grand Moff Tarkin', 'Boba Fett', 'Emperor Palpatine'];

//-------------------------------------------------//

// removes 0 elements from index 2, and inserts 'Jabba the Hutt'
var removed = darkSide.splice(2, 0, 'Jabba the Hutt');
console.log(darkSide);
// ['Darth Vader', 'Grand Moff Tarkin', 'Jabba the Hutt', 'Boba Fett', 'Emperor Palpati
console.log(removed);
// [], no elements removed
```

```
//-----------------------------------------------------//

// removes 1 element from index 3
removed = darkSide.splice(3, 1);
console.log(darkSide);
// ['Darth Vader', 'Grand Moff Tarkin', 'Jabba the Hutt', 'Emperor Palpatine']
console.log(removed);
// ['Boba Fett']

//-----------------------------------------------------//

// removes 1 element from index 2, and inserts 'Darth Maul'
removed = darkSide.splice(2, 1, 'Darth Maul');
console.log(darkSide);
// ['Darth Vader', 'Grand Moff Tarkin', 'Darth Maul', 'Emperor Palpatine']
console.log(removed);
// ['Jabba the Hutt']

//-----------------------------------------------------//

// removes 2 elements from index 0, and inserts 'Count Dooku', 'General Grievous' a
removed = darkSide.splice(0, 2, 'Count Dooku', 'General Grievous', 'Asajj Ventress')
console.log(darkSide);
// ['Count Dooku', 'General Grievous', 'Asajj Ventress', 'Darth Maul', 'Emperor Palpat
console.log(removed);
// ['Darth Vader', 'Grand Moff Tarkin']

//-----------------------------------------------------//

// removes 2 elements from index 3
removed = darkSide.splice(3, darkSide.length);
console.log(darkSide);
// ['Count Dooku', 'General Grievous', 'Asajj Ventress']
console.log(removed);
// ['Darth Maul', 'Emperor Palpatine']
```

9. Array.prototype.forEach()

```
array.forEach(callback[, thisArg])
```

Overview

The `forEach()` function executes a provided function on each element of the array. `forEach()` has no return value and does not return the original array. `forEach()` is related to the `map()`, `filter()`, `every()`, and `some()` functions

and as such they share some related details. They all expect a callback function as the first argument. The optional second argument is a way to specify the `this` value for the callback function. All of these functions check the length of the array before looping. If the provided callback adds elements to the array during it's execution, those elements are not included for looping by the `forEach()`. Also of interesting note is that if the callback changes values in the original array before they are looped over, those changed values will be passed during their callback execution.

array.forEach() function examples

```
var planetsAndMoons = ['Alderaan', 'Bespin', 'Coruscant', 'DQar', 'Dagobah', 'Endor'

function listPlanets(element, index, array) {
   console.log('planetsAndMoons[' + index + '] = ' + element);
}

planetsAndMoons.forEach(listPlanets);

// planetsAndMoons[0] = Alderaan
// planetsAndMoons[1] = Bespin
// planetsAndMoons[2] = Coruscant
// planetsAndMoons[3] = DQar
// planetsAndMoons[4] = Dagobah
```

```
//  planetsAndMoons[5] = Endor
//  planetsAndMoons[6] = Geonosis
//  planetsAndMoons[7] = Hosnian Prime
//  planetsAndMoons[8] = Hoth
//  planetsAndMoons[9] = Jakku
//  planetsAndMoons[10] = Kamino
//  planetsAndMoons[11] = Kashyyyk
//  planetsAndMoons[12] = Lothal
//  planetsAndMoons[13] = Mustafar
//  planetsAndMoons[14] = Naboo
//  planetsAndMoons[15] = Sullust
//  planetsAndMoons[16] = Takodana
//  planetsAndMoons[17] = Tatooine
//  planetsAndMoons[18] = Utapau
//  planetsAndMoons[19] = Yavin
//  planetsAndMoons[20] = Yavin 4
```

```
var numbers = [2, 4, 6];

numbers.original = [];

numbers.forEach(function (element, index, numbers) {
    numbers[index] *= 2;
    numbers.original.push(element);
});

function listNumbers(element, index) {
    console.log('index' + index + ' has element ' + element);
}

numbers.original.forEach(listNumbers);
```

```
numbers.forEach(listNumbers);

// index0 has element 2
// index1 has element 4
// index2 has element 6
// index0 has element 4
// index1 has element 8
// index2 has element 12
```

10. Array.prototype.concat()

```
var new_array = old_array.concat(value1[, value2[, ...[, valueN]]])
```

Overview

`concat()` is a function that creates a new array which is the result of adding the supplied arguments to the original array. The original array is left intact, and the new array is the original plus any added elements. If any of the arguments to the `concat()` function are themselves an array, then the elements of that supplied array are added, rather than the entire array itself.

Returns

A new array, which is created by adding each of the supplied arguments to the original array.

array.concat() function examples

```
var legoStarWarsGames = ['Empire Vs Rebels', 'Ultimate Rebel', 'The Quest for R2-
var legoStarWarsSets = [10188, 75111, 75110, 75109];

var gamesAndSets = legoStarWarsGames.concat(legoStarWarsSets);

console.log(gamesAndSets);

// ["Empire Vs Rebels", "Ultimate Rebel", "The Quest for R2-D2", 10188, 75111, 75
```

```
var legoStarWarsGames = ['Empire Vs Rebels', 'Ultimate Rebel', 'The Quest for R2-
var legoStarWarsSets = [10188, 75111, 75110, 75109];
var availablePlatforms = ['Wii', 'Xbox', 'Playstation'];
```

```
var gamesSetsPlatforms = legoStarWarsGames.concat(legoStarWarsSets, available

console.log(gamesSetsPlatforms);

// ["Empire Vs Rebels", "Ultimate Rebel", "The Quest for R2-D2", 10188, 75111, 75
```

11. Array.prototype.shift()

> array.shift()

Overview

The `shift()` function *removes and returns* the **first element** of the original array. All remaining elements in the array get shifted one slot to the left in order to fill the hole created by removing the first element. If you try to apply this function to an empty array, it will do nothing and simply return `undefined`. The `shift()` function does not create a new array, it modifies the original array directly.

Returns

`shift()` returns the first element from the original array.

array.shift() function examples

```
var starWarsCreatures = ['Tach', 'Kath hound', 'Ruggers', 'Spice spider'];

console.log('starWarsCreatures before: ' + starWarsCreatures);
// "starWarsCreatures before: Tach,Kath hound,Ruggers,Spice spider"

var shifted = starWarsCreatures.shift();

console.log('starWarsCreatures after: ' + starWarsCreatures);
// "starWarsCreatures after: Kath hound,Ruggers,Spice spider"

console.log('Removed this element: ' + shifted);
// "Removed this element: Tach"
```

12. Array.prototype.unshift()

```
array.unshift([element1[, ...[, elementN]]])
```

Overview

A direct opposite to the `shift()` function which we just discussed, is the `unshift()` function. `unshift()` inserts any arguments you pass to it into the beginning of the array. All of the existing elements need to shift to the right in order to make room for the new elements. Each argument passed to `unshift()` gets added in order starting from index 0. In addition, `unshift()` is modifying the original array directly.

Returns

The new length of the array.

array.unshift() function examples

```
var starWarsCreatures = ['Eopie', 'Acklay', 'Talortai', 'Boma'];

console.log('starWarsCreatures before: ' + starWarsCreatures);
// starWarsCreatures before: Eopie,Acklay,Talortai,Boma

var newcount = starWarsCreatures.unshift('Climbing tauntaun');

console.log('starWarsCreatures after: ' + starWarsCreatures);
// starWarsCreatures after: Climbing tauntaun,Eopie,Acklay,Talortai,Boma

console.log('The array now has: ' + newcount + ' elements');
// The array now has: 5 elements
```

```
var starWarsCreatures = ['Eopie', 'Acklay', 'Talortai', 'Boma'];

console.log('starWarsCreatures before: ' + starWarsCreatures);
```

```
// starWarsCreatures before: Eopie,Acklay,Talortai,Boma

var newcount = starWarsCreatures.unshift('Climbing tauntaun');

console.log('starWarsCreatures after: ' + starWarsCreatures);
// starWarsCreatures after: Climbing tauntaun,Eopie,Acklay,Talortai,Boma

console.log('The array now has: ' + newcount + ' elements');
// The array now has: 5 elements

newcount = starWarsCreatures.unshift(['Gullipuds', 'Mynock']);


console.log('starWarsCreatures after: ' + starWarsCreatures);
//  starWarsCreatures after: Gullipuds,Mynock,Climbing tauntaun,Eopie,Acklay,Talor

console.log('The array now has: ' + newcount + ' elements');
//  The array now has: 6 elements
```

13. Array.prototype.map()

> array.map(callback[, thisArg])

Overview

This is a very useful JavaScript function, and made use of all the time in professional software. The `map()` function loops over every element in the array, executing a callback function on each item. Once `map()` has completed looping through the array, it takes the results from each callback applied to each element, and returns those results in their entirety as an array. You are left with a new array with updated values, and an old array which has the original values. These two arrays are equal in length. When the callback is invoked, it is done so with three arguments. Those are the value of the element, the index of the element, and the Array object being traversed. If the optional second parameter is provided to `map()`, it will be used as the `this` value for each execution of the callback.

note: The `map()` and `forEach()` functions seem like they are the same, but

they are in fact different. The difference is that `forEach()` iterates over an array and applies some operation with side effects to each array member such as saving each one to the database, or some other side effect. `map()` on the other hand iterates over an array, updates each member of that array, and returns another array of the same size with the transformed members (such as converting an array of strings to all lowercase).

Returns

`map()` returns a new array with elements computed by the provided callback function.

array.map() function examples

```
var starwars = ['star wars', 'the phantom menace', 'the force awakens'];
var swModified = starwars.map(function (element) {
   return element.toUpperCase();
});

console.log(swModified);
//   ["STAR WARS", "THE PHANTOM MENACE", "THE FORCE AWAKENS"]
```

```
var one = [3, 5, 7];
var two = one.map(function (number) {
   return number * number;
});

console.log(one); // [3, 5, 7]
console.log(two); //  [9, 25, 49]
```

```
var str = 'The Clone Wars';
var arr = Array.prototype.map.call(str, function (el) {
   return el;
});

console.log(str); // The Clone Wars
console.log(arr); //  ["T", "h", "e", " ", "C", "l", "o", "n", "e", " ", "W", "a", "r", "s"]
```

14. Array prototype sort()

14. Array.prototype.sort()

```
array.sort([compareFunction])
```

Overview

If you want to sort an array in JavaScript, then you can do so with the built in `sort()` function. `sort()` applies the sort directly to the original array, no copy of the array is made. You can optionally provide a callback function that will determine the behavior of the sort. If you do not provide one, the `sort()` function will first convert all elements to strings, and then sort based on something called the Unicode code point value. Basically that means alphabetical, but with some special rules such as capital letters coming before lowercase.

Returns

`sort()` returns a reference to the original array.

array.sort() function examples

```
var creatures = ['Pug jumper', 'Rong boars', 'Wamba', 'Oslet'];
creatures.sort(); //

var nums = [1, 10, 2, 21, 33, 04, 12, 09, 300];
nums.sort(); // [1, 10, 2, 21]
// Watch out that 10 comes before 2,
// because '10' comes before '2' in Unicode code point order.

var spacecraft = ['Lambda Class Shuttle', 'imperial landing craft', '4 Tantive'];
spacecraft.sort();
// In Unicode, numbers come before upper case letters,
// which come before lower case letters.

console.log(creatures);
//  ["Oslet", "Pug jumper", "Rong boars", "Wamba"]

console.log(nums);
//   [1, 10, 12, 2, 21, 300, 33, 4, 9]

console.log(spacecraft);
//   ["4 Tantive", "Lambda Class Shuttle", "imperial landing craft"]
```

As we can see from the above example, when you do not supply a callback function to the sort method, ordering may not be what you expect. The rules for ordering in Unicode are a bit unique. For example, numbers come before upper case letters and uppercase letters come before lowercase letters. In addition, an array of numbers is converted to strings before sorting, so this is why we see 33 coming before 4 and 9 in the example above. Here is a callback function example to show ordering numbers how you might expect.

```
var nums = [1, 10, 2, 21, 33, 04, 12, 09, 300];
nums.sort(numsasexpected);

function numsasexpected(one, two) {
   return one - two;
}

console.log(nums);
//   [1, 2, 4, 9, 10, 12, 21, 33, 300]
```

15. Array.prototype.pop()

```
array.pop()
```

Overview

The opposite of the `push()` function would be the `pop()` function. The `pop()` function removes the last element from an array, decrements the length of the array, and returns the value which was removed from the array. If you try to apply the `pop()` to an empty array, you will simply get `undefined` returned.

Returns

`pop()` returns the last element of the array it is called on.

array.pop() function examples

```
var usefulFunctions = ['push', 'indexOf', 'slice', 'toString', 'filter', 'join'];

console.log('usefulFunctions initially has: ' + usefulFunctions);
// usefulFunctions initially has: push,indexOf,slice,toString,filter,join
```

```
// usefulFunctions initially has: push,indexOf,slice,toString,filter,join

var firstpop = usefulFunctions.pop();

console.log('after one pop, usefulFunctions has: ' + usefulFunctions);
// after one pop, usefulFunctions has: push,indexOf,slice,toString,filter

var poptwo = usefulFunctions.pop();

console.log('after two pops, usefulFunctions has: ' + usefulFunctions);
// after two pops, usefulFunctions has: push,indexOf,slice,toString

console.log('popone contains: ' + firstpop);
// popone contains: join

console.log('poptwo contains: ' + poptwo);
// poptwo contains: filter
```

16. Array.prototype.reduce()

```
array.reduce(callback[, initialValue])
```

Overview

`reduce()` accepts a function as the first parameter which acts like a binary operator. This callback function takes two values, operates on them, and returns a result. The number of times the callback function runs is always one less than the total length of the original array. For example if the original array has a length of 10, the callback will run 9 times. The final result is one combined value.

Returns

The reduced value of the array, which is the return value of the last time the callback function is executed.

array.reduce() function examples

In this example, we will use `array.reduce()` to operate on values contained within a simple array of objects. `array.reduce()` is a little more challenging than some of the other useful array functions, so spend some extra time on this one if it doesn't click immediately.

```
var skills = [
   {
      name: 'Tom',
      skill: 'CSS',
      yearsExperience: 3,
      category: 'Web Design'
   },
   {
      name: 'Jim',
      skill: 'HTML',
      yearsExperience: 10,
      category: 'Web Design'
   },
   {
      name: 'Sue',
      skill: 'JavaScript',
      yearsExperience: 5,
      category: 'Web Development'
   },
   {
      name: 'Maria',
      skill: 'PHP',
      yearsExperience: 7,
      category: 'Web Development'
   },
   {
      name: 'John',
      skill: 'Photoshop',
      yearsExperience: 1,
      category: 'Web Design'
   },
   {
      name: 'David',
      skill: 'Writing',
      yearsExperience: 12,
      category: 'Content'
   },
   {
      name: 'Ellen',
      skill: 'Editor',
      yearsExperience: 5,
      category: 'Content'
   }
];


var totalexperience = skills.reduce(function (prev, current) {
   return prev + current.yearsExperience;
}, 0);
```

```
console.log('The team has a cumulative experience of: ' + totalexperience + ' years
//  The team has a cumulative experience of: 43 years!

var categoryExperienceTotals = skills.reduce(function (groupedByCategory, worker
    var category = worker.category;
    if (!groupedByCategory[category]) {
        groupedByCategory[category] = 0;
    }
    groupedByCategory[category] += worker.yearsExperience;
    return groupedByCategory;
}, {});

console.log('Our workers have ' + categoryExperienceTotals['Web Design'] + ' years
    + categoryExperienceTotals['Web Development'] + ' years of Web Development
    + categoryExperienceTotals['Content'] + ' years of Content production Experienc
);
//  Our workers have 14 years of Web Design Experience, 12 years of Web Developr
//  and 17 years of Content production Experience!

var workersByCategory = skills.reduce(function (groupedByWorkers, worker) {
    if (!groupedByWorkers[worker.category]) {
        groupedByWorkers[worker.category] = 0;
    }
    groupedByWorkers[worker.category]++;
    return groupedByWorkers;
}, {});

console.log('We have ' + workersByCategory['Web Design'] + ' employees in Web D
    + workersByCategory['Web Development'] + ' in Web Development, and '
    + workersByCategory['Content'] + ' working in Content production!'
);
//  We have 3 employees in Web Design, 2 in Web Development, and 2 working in C
```

17. Array.prototype.some()

```
array.some(callback[, thisArg])
```

Overview

As we can see from the signature above, the `some()` function takes a callback as the first argument, and an optional second argument. The second argument, if provided, specifies the `this` value for invocations of the supplied callback. The `some()` function runs the provided callback on each element in the array, and if the callback returns `true`, then the `some()` function stops right away and returns `true`. If all callback iterations return

`false`, then `some()` returns `false`.

Returns

`some()` either returns `true` or `false`. At least one item in the array must return `true` when the callback is applied in order for `some()` to return `true`.

array.some() function examples

```
function jsarrayfuncs(element, index, array) {
   return element === 'javascript array functions';
}

var one = ['the good parts', 'the better parts', 'has no parts', 'atwood loves php'];
var two = ['life is good', 'arrays for all', 'node for win', 'javascript array functions'];

console.log(one.some(jsarrayfuncs)); // false
console.log(two.some(jsarrayfuncs)); // true
```

18. Array.prototype.lastIndexOf()

```
array.lastIndexOf(searchElement[, fromIndex = arr.length – 1])
```

Overview

The `lastIndexOf()` function searches through an array backwards for a supplied value. Once the function finds that value, it returns the index position of where that value exists in the array. If you provide the optional second argument to the `lastIndexOf()` function, searching will begin from that starting point and go backwards. If you do not supply this second argument, searching starts at the end of the array. If no match is found, `-1` is returned.

Returns

The highest index position that is less than or equal to the start of the array where the element is === to the value you are looking for, or -1 if there are no matches found.

array.lastIndexOf() function examples

```
var starwars = ['Episode 4: A New Hope',
   'Episode 5: The Empire Strikes Back',
```

```
    'Episode 6: Return of the Jedi',
    'Episode 1: The Phantom Menace',
    'Episode 2: Attack of the Clones',
    'Episode 3: Revenge of the Sith',
    'Episode 7: The Force Awakens'];

console.log(starwars.lastIndexOf('Episode 7: The Force Awakens'));    // 6
console.log(starwars.lastIndexOf('Episode 3: Revenge of the Sith')); // 5
console.log(starwars.lastIndexOf('Episode 2: Attack of the Clones', 3)); // -1
console.log(starwars.lastIndexOf('Episode 1: The Phantom Menace', -4)); // 3
```

```
var locatedat = [];
var enemies = ['Kylo Ren', 'Darth Vader', 'Storm Trooper', 'General Hux', 'Emperor P
var enemy = 'Storm Trooper';
var enemylocation = enemies.lastIndexOf(enemy);
while (enemylocation != -1) {
    locatedat.push(enemylocation);
    enemylocation = enemies.lastIndexOf(enemy, enemylocation - 1);
}
console.log(locatedat);  // 5, 2
```

19. Array.prototype.reduceRight()

```
array.reduceRight(callback[, initialValue])
```

Overview

The `reduceRight()` function works just like `reduce()` with one key difference. `reduceRight()` enumerates array elements from right to left (from highest index to lowest) rather than left to right (lowest to highest).

array.reduceRight() function example

```
var sum = [0, 10, 20, 30].reduceRight(function (one, two) {
    return one + two;
});

console.log(sum); // 60

var flat = [[0, 10], [20, 30, [1, 2, 3, 4]], [40, 50]].reduceRight(function (one, two)
    return one.concat(two);
}, []);

console.log(flat);
//   [40, 50, 20, 30, [1, 2, 3, 4], 0, 10]
```

20. Array.prototype.every()

```
array.every(callback[, thisArg])
```

Overview

You can use the `every()` function to test whether a condition is `true` for all elements in an array. As like with most looping functions in JavaScript, you are expected to provide a callback function which will run on each element in the array from lowest to highest index. If all iterations return `true` when the callback runs, then the `every()` function itself will return `true`. If however an interation of the callback returns a `false` value, then `every()` stops right away and returns `false`.

array.every() function example

```
function greaterThan100(element, index, array) {
   return element > 100;
}

arr1 = [200, 150, 550, 300, 101];
arr2 = [330, 120, 100, 508, 102];

console.log(arr1.every(greaterThan100));  // true
console.log(arr2.every(greaterThan100));  // false
```

Array.prototype.reverse()

```
array.reverse()
```

Overview

The `reverse()` function does exactly what you think it would do. It reverses the order of the elements of they array it is applied to. The `reverse()` function does this right on the original array, no new array is created or returned. In addition keep in mind that if there are many references to a particular array in your program, and you reverse that array, all references now also see an array which has been reversed. Perhaps that is why this function appears lower on our most useful JavaScript array functions list.

Remember, the lower on the list a function appears, the less times we found it in use in popular open source software.

array.reverse() function example

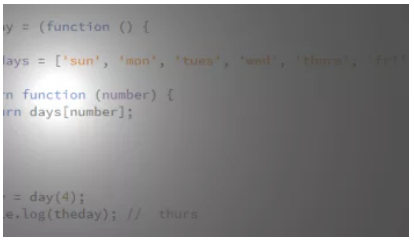
```
var javascriptArray = ['eat', 'sleep', 'breathe', 'javascript'];
javascriptArray.reverse();

console.log(javascriptArray);
//   ["javascript", "breathe", "sleep", "eat"]
```

Most Useful JavaScript Array Functions Summary

In this episode we took a look at all of the many useful JavaScript Array Functions you're likely to make use of and find helpful during your JavaScript programming. We used a method of analyzing popular JavaScript source code repositories hosted on Github, the well known open source software sharing and collaboration website. We found that the most useful JavaScript functions according to the number of times they were used in popular JavaScript frameworks to be Array.**push()**, Array.**indexOf()**, Array.**slice()**, Array.**toString()**, Array.**filter()**, Array.**join()**, Array.**splice()**, Array.**forEach()**, Array.**concat()**, Array.**shift()**, Array.**unshift()**, Array.**map()**, Array.**sort()**, Array.**pop()**, Array.**reduce()**, Array.**some()**, Array.**lastIndexOf()**, Array.**reduceRight()**, Array.**every()**, and finally, Array.**reverse()**. It would make sense to memorize as many of these as you can, or at least be very familiar with the ones that appear at the top of the list.

Related



(http://vegibit.com/javascr(http://vegibit.com/javascr(http://vegibit.com/javascript-

string-functions/)
The Top 15 Most Popular JavaScript String Functions (http://vegibit.com/javascript-string-functions/)
In "web development"

tutorials-for-beginners/)
JavaScript Tutorials For Beginners (http://vegibit.com/javascript-tutorials-for-beginners/)
In "web development"

functions-tutorial/)
JavaScript Functions Tutorial (http://vegibit.com/javascript-functions-tutorial/)
In "web development"

javascript (http://vegibit.com/tag/javascript/)

← Previous (http://vegibit.com/douglas-crockford-the-good-parts-examples/)

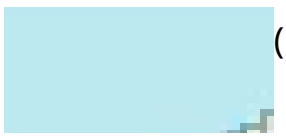
Next → (http://vegibit.com/javascript-string-functions/)

Search

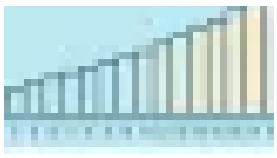
Top Posts & Pages

(http://vegibit.com/vue-js-tutorial/)
Vue.js Tutorial (http://vegibit.com/vue-js-tutorial/)

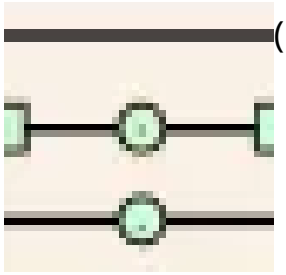
(http://vegibit.com/laravel-repository-pattern/)
Laravel Repository Pattern (http://vegibit.com/laravel-repository-pattern/)



(http://vegibit.com/create-a-bar-chart-with-d3-javascript/)
Create A Bar Chart With D3 JavaScript



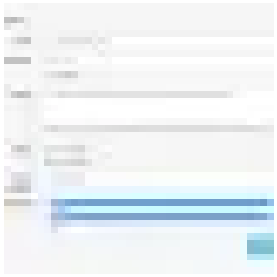
(http://vegibit.com/create-a-bar-chart-with-d3-javascript/)



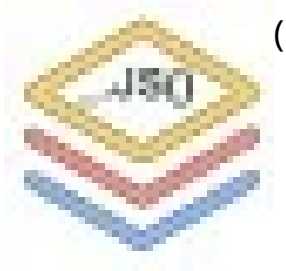
(http://vegibit.com/json-in-laravel/)

JSON in Laravel (http://vegibit.com/json-in-laravel/)

(http://vegibit.com/create-form-elements-using-laravel-and-bootstrap/)

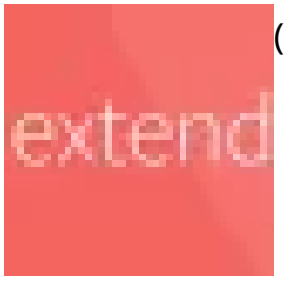


Create Form Elements Using Laravel and Bootstrap (http://vegibit.com/create-form-elements-using-laravel-and-bootstrap/)



(http://vegibit.com/underscore-js-map-function/)

Underscore JS Map Function (http://vegibit.com/underscore-js-map-function/)



(http://vegibit.com/laravel-eloquent-orm-tutorial/)

Laravel Eloquent ORM Tutorial (http://vegibit.com/laravel-eloquent-orm-tutorial/)

(http://vegibit.com/php-string-functions/)

The Ultimate PHP String Functions List (http://vegibit.com/php-string-functions/)

(http://vegibit.com/getting-started-with-testing-in-laravel/)

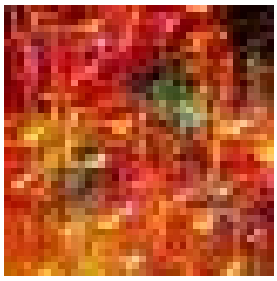
Getting Started With Testing In Laravel

(http://vegibit.com/getting-started-with-testing-in-laravel/)

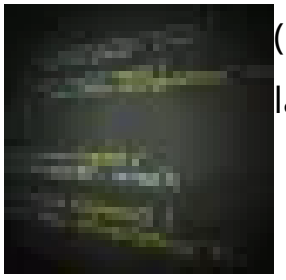
(http://vegibit.com/laravel-cache-tutorial/)

Laravel Cache Tutorial (http://vegibit.com/laravel-cache-tutorial/)

(http://vegibit.com/super-easy-crud-with-gii-and-the-yii2-framework/)

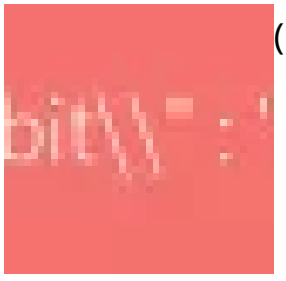


Super Easy CRUD With Gii And The Yii2 Framework (http://vegibit.com/super-easy-crud-with-gii-and-the-yii2-framework/)



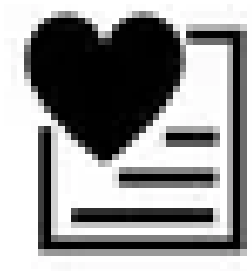
(http://vegibit.com/many-to-many-relationships-in-laravel/)

Many to Many Relationships in Laravel (http://vegibit.com/many-to-many-relationships-in-laravel/)



(http://vegibit.com/laravel-file-structure/)

Laravel File Structure (http://vegibit.com/laravel-file-structure/)



(http://vegibit.com/how-to-write-a-listicle/)

14 Steps To Write A Great Listicle (http://vegibit.com/how-to-write-a-listicle/)

(http://vegibit.com/how-to-install-the-advanced-template-in-yii2/)

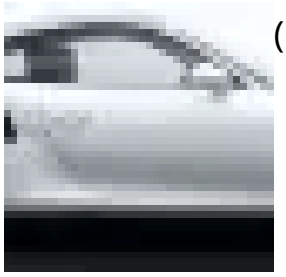
How To Install The Advanced Template In Yii2 (http://vegibit.com/how-to-install-the-advanced-template-in-yii2/)

(http://vegibit.com/most-useful-php-array-functions/)

The 27 Most Useful PHP Array Functions You Need To Know! (http://vegibit.com/most-useful-php-array-functions/)

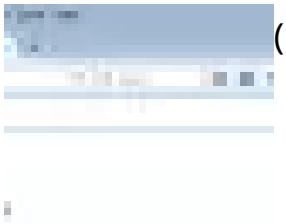
(http://vegibit.com/upgrading-vuejs/)

Upgrading VueJS (http://vegibit.com/upgrading-vuejs/)



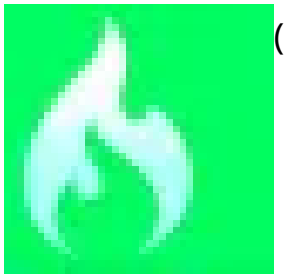
(http://vegibit.com/what-is-a-laravel-interface/)

What is a Laravel Interface? (http://vegibit.com/what-is-a-laravel-interface/)



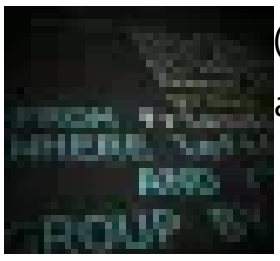
(http://vegibit.com/vue-js-for-interactive-web-interfaces/)

Vue.js for Interactive Web Interfaces (http://vegibit.com/vue-js-for-interactive-web-interfaces/)



(http://vegibit.com/codeigniter-model-tutorial/)

Codeigniter Model Tutorial (http://vegibit.com/codeigniter-model-tutorial/)



(http://vegibit.com/mysql-group-by-having-limit-offset-and-more/)

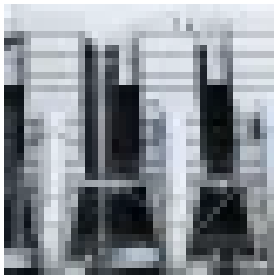
MySQL Group By Having Limit Offset and More! (http://vegibit.com/mysql-group-by-having-limit-offset-

and-more/)

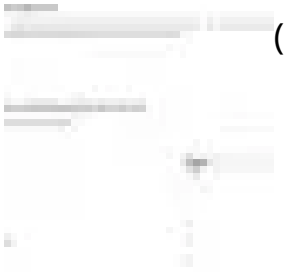
(http://vegibit.com/custom-helper-functions-in-laravel/)

Custom Helper Functions in Laravel (http://vegibit.com/custom-helper-functions-in-laravel/)

(http://vegibit.com/creating-static-and-dynamic-web-pages-in-laravel/)

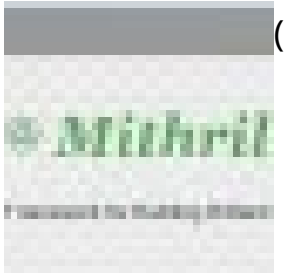


Creating Static And Dynamic Web Pages In Laravel (http://vegibit.com/creating-static-and-dynamic-web-pages-in-laravel/)



(http://vegibit.com/developing-with-vuejs-and-php/)

Developing With VueJS and PHP (http://vegibit.com/developing-with-vuejs-and-php/)



(http://vegibit.com/mithril-javascript-tutorial/)

Mithril JavaScript Tutorial (http://vegibit.com/mithril-javascript-tutorial/)

(http://vegibit.com/easy-php-dates-and-times-with-carbon/)

Easy PHP Dates and Times With Carbon (http://vegibit.com/easy-php-dates-and-times-with-carbon/)

(http://vegibit.com/laravel-collections-tutorial/)

Laravel Collections Tutorial (http://vegibit.com/laravel-collections-tutorial/)