Lodash tutorial

last modified January 10, 2023

Lodash tutorial covers the Lodash JavaScript library. In this introductory Lodash tutorial, we cover Lodash functions in multiple examples.

Lodash

Lodash is a JavaScript library which provides utility functions for common programming tasks. It uses functional programming paradigm. Lodash was inspired by Underscore.js.

Lodash helps programmers write more concise and easier to maintain JavaScript code. Lodash contains tools to simplify programming with strings, numbers, arrays, functions and objects.

By convention, Lodash module is mapped to the underscore character.

Lodash installation

First, we install the Lodash library.

```
$ npm init -y
$ npm i lodash
```

The Lodash library is installed locally with npm.

Lodash version

In the first example, we determine the version of the Lodash library.

main.js

```
const _ = require("lodash");

const ver = _.VERSION;
console.log(ver);
```

The example prints the version of the Lodash library.

```
const _ = require("lodash");
```

By convention, the Lodash library is mapped to the underscore character.

```
const ver = _.VERSION;
console.log(ver);
```

The version is stored in the VERSION variable.

```
$ node main.js
4.17.21
```

We use Lodash version 4.17.21.

Lodash first and last array elements

The _.first/_.head functions return the first array element; the _.last function returns the last array element.

main.js

```
const _ = require("lodash");

let words = ['sky', 'wood', 'forest', 'falcon',
    'pear', 'ocean', 'universe'];

let fel = _.first(words);
let lel = _.last(words);

console.log(`First element: ${fel}`);
console.log(`Last element: ${lel}`);
```

The example outputs the first and last elements of an array of words.

```
$ node main.js
First element: sky
Last element: universe
```

Lodash nth array elements

The _.nth funcion gets the element at index n of an array. If n is negative, the nth element from the end is returned.

main.js

```
const _ = require("lodash");

let nums = [1, 2, 3, 4, 5, 6, 7, 8];

console.log(_.nth(nums, 3));
console.log(_.nth(nums, -3));
```

In the example, we get the fourth element from the beginning and end. The indexing starts from zero.

```
$ node main.js
4
6
```

Lodash chunking array

The _.chunk function creates an array of elements split into groups the length of the specified size.

main.js

```
const _ = require("lodash");

let nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];

let c1 = _.chunk(nums, 2);
console.log(c1);

let c2 = _.chunk(nums, 3);
console.log(c2);
```

The example chunks the nums array into an array of two and three element subarrays.

```
$ node main.js
[ [ 1, 2 ], [ 3, 4 ], [ 5, 6 ], [ 7, 8 ], [ 9 ] ]
[ [ 1, 2, 3 ], [ 4, 5, 6 ], [ 7, 8, 9 ] ]
```

Getting array slice

The _.slice method gets a slice from an array. It takes two indexes: the starting and ending index, where the starting index is inclusive and the ending is exclusive.

main.js

```
const _ = require("lodash");

let nums = [1, 2, 3, 4, 5, 6, 7, 8, 9];

let c1 = _.slice(nums, 2, 6);
console.log(c1);

let c2 = _.slice(nums, 0, 8);
console.log(c2);
```

The example creates two slices from the nums array.

```
$ node main.js
[ 3, 4, 5, 6 ]
[ 1, 2, 3, 4, 5, 6, 7, 8 ]
```

Lodash random number

The _.random function produces random values between the inclusive lower and upper bounds.

main.js

```
const _ = require("lodash");

let r = _.random(10);
console.log(r);

r = _.random(5, 10);
console.log(r);
```

The example prints two random values.

```
let r = _.random(10);
```

We produce a random value between 0 and 10.

```
r = _.random(5, 10);
```

Here we produce a random value between 5 and 10.

Lodash random array element

With the _.sample function, we can pick a random element from an array.

main.js

```
const _ = require("lodash");

let words = ['sky', 'wood', 'forest', 'falcon',
    'pear', 'ocean', 'universe'];

let word = _.sample(words);
console.log(word);
```

The example picks a random word from an array with _.sample.

```
$ node main.js
falcon
```

Lodash shuffling array elements

The _.shuffle function shuffles a collection.

main.js

```
const _ = require("lodash");

let words = ['sky', 'wood', 'forest', 'falcon',
    'pear', 'ocean', 'universe'];

console.log(_.shuffle(words));
console.log(_.shuffle(words));
console.log(_.shuffle(words));
console.log(words);
```

The example creates three new randomly reorganized arrays from an initial array of words.

```
$ node main.js
[ 'sky', 'ocean', 'universe', 'falcon', 'pear', 'wood', 'forest' ]
[ 'wood', 'ocean', 'falcon', 'forest', 'sky', 'universe', 'pear' ]
[ 'forest', 'ocean', 'sky', 'wood', 'falcon', 'universe', 'pear' ]
[ 'sky', 'wood', 'forest', 'falcon', 'pear', 'ocean', 'universe' ]
```

This is the output. The original array is not modified; the _.shuffle function creates a new array.

Lodash _.times function

The _.times executes the function n times.

main.js

```
const _ = require("lodash");

_.times(4, () => {

    console.log("brave");
})
```

In the example, we execute the innter function four times. The function prints a word to the console.

```
$ node main.js
brave
brave
brave
brave
```

Lodash _.delay function

The _.delay function delays the execution of a function for the specified amount of milliseconds.

main.js

```
const _ = require("lodash");

function message()
{
    console.log("Some message");
}

_.delay(message, 150);
console.log("Some other message");
```

The example outputs two messages. The first one is delayed for 150 ms.

```
$ node main.js
Some other message
Some message
```

Lodash determine data type

Lodash contains functions which determine the data type of a value.

main.js

```
const _ = require("lodash");
```

```
let vals = [1, 2, 'good', [1, 2], {name: 'Peter', age: 32}];

vals.forEach( (e) => {

    if (_.isNumber(e)) {
        console.log(`${e} is a number`);
    }

    if (_.isString(e)) {
        console.log(JSON.stringify(e) + ' is a string');
    }

    if (_.isArray(e)) {
        console.log(JSON.stringify(e) + ' is an array');
    }

    if (_.isObject(e)) {
        console.log(JSON.stringify(e) + ' is an object');
    }

});
```

In the example, we determine the data types of the elements of an array.

```
let vals = [1, 2, 'good', [1, 2], {name: 'Peter', age: 32}];
```

We have an array of values, including numbers, a string, an array, and an object.

```
if (_.isNumber(e)) {
    console.log(`${e} is a number`);
}
```

The _.isNumber function checks if a value is a number.

```
if (_.isString(e)) {
    console.log(JSON.stringify(e) + ' is a string');
}
```

The _.isString function checks if a value is a string.

```
if (_.isArray(e)) {
    console.log(JSON.stringify(e) + ' is an array');
}
```

The _.isArray function checks if a value is an array.

```
if (_.isObject(e)) {
    console.log(JSON.stringify(e) + ' is an object');
}
```

The _.isObject function checks if a value is an object.

```
$ node main.js
1 is a number
2 is a number
"good" is a string
[1,2] is an array
[1,2] is an object
{"name":"Peter","age":32} is an object
```

Lodash _.range function

The Lodash _.range function creates an array of numbers. The function accepts the start, end, and step parameters.

main.js

```
const _ = require("lodash");

let vals = _.range(10);
console.log(vals);

let vals2 = _.range(0, 15);
console.log(vals2);

let vals3 = _.range(0, 15, 5);
console.log(vals3);
```

In the code example, we create three ranges of values.

```
let vals = _.range(10);
```

This line creates an array of values 0..9. The end value is mandatory, the start and step are optional. The end is non-inclusive; therefore, value 10 is not included.

```
let vals2 = _.range(0, 15);
```

Here we specify start and step parameters. We create an array of values 0..14.

```
let vals3 = _.range(0, 15, 5);
```

Finally, we provide all three parameters. An array of 0, 5, and 10 values is created.

```
$ node main.js
[
  0, 1, 2, 3, 4,
  5, 6, 7, 8, 9
]
[
   0,  1,  2, 3,  4,  5,
   6,  7,  8, 9, 10, 11,
  12, 13, 14
]
[ 0, 5, 10 ]
```

Lodash maximum and minimum

Lodash allows to compute the maximum and minimum values of an array.

main.js

```
const _ = require("lodash");

let vals = [-3, 4, 0, 12, 43, 9, -12];

let min = _.min(vals);
console.log(min);

let max = _.max(vals);
console.log(max);

max = _.max(_.range(5, 25));
console.log(max);

let obs = [{n: 12}, {n: -4}, {n: 4}, {n: -11}];

min = _.minBy(obs, 'n');
console.log(min);

max = _.maxBy(obs, 'n');
console.log(max);
```

The example computes the minimum and maximum values of an array.

```
let min = _.min(vals);
console.log(min);
```

The _.min function returns the minimum of the array.

```
let max = _.max(vals);
console.log(max);
```

The ._max function returns the maximum of the array.

```
min = _.minBy(obs, 'n');
console.log(min);

max = _.maxBy(obs, 'n');
console.log(max);
```

To compute the minimum and maximum of object properties, we use the _.minBy and .maxBy functions.

```
$ node main.js
-12
43
24
{ n: -11 }
{ n: 12 }
```

Lodash _.sum function

The _.sum function calculates the sum of array values.

main.js

```
const _ = require("lodash");

let vals = [-2, 0, 3, 7, -5, 1, 2];

let sum = _.sum(vals);
console.log(sum);
```

In the code example, we compute and print the sum of array values.

```
$ node main.js
6
```

Lodash _.curry

Currying is a transformation of a function with multiple arguments into a sequence of nested functions with a single argument. The currying allows to perform function specialization and composition. Learn more about currying in the JavaScript currying tutorial.

The _.curry function turns a normal function into a curried one.

main.js

```
const _ = require("lodash");

function multiply(a, b, c) {

    return a * b * c;
}

let curried = _.curry(multiply);

let ret = curried(2)(3)(4);
console.log(ret);
```

In the example, we turn the multiply function into a curried version.

```
$ node main.js
24
```

Lodash collection filter

The _.filter function returns an array of elements for which the predicate function returns true.

main.js

```
const _ = require("lodash");

let nums = [4, -5, 3, 2, -1, 7, -6, 8, 9];

let pos_nums = _.filter(nums, (e) => e > 0);
console.log(pos_nums);
```

In the code example, we filter out positive values of an array.

```
let pos_nums = _.filter(nums, (e) => e > 0);
```

A predicate is a fuction that returns a boolean value. In our case we have an anonymous function that returns true for values greater than 0.

```
$ node main.js
[ 4, 3, 2, 7, 8, 9 ]
```

Lodash collection find

The _.find function iterates over elements of a collection and returns the first element for which the predicate returns true. Likewise, the _.findLast returns the last element.

main.js

```
const _ = require("lodash");

let users = [
  { name: 'John', age: 25 },
  { name: 'Lenny', age: 51 },
  { name: 'Andrew', age: 43 },
  { name: 'Peter', age: 81 },
  { name: 'Anna', age: 43 },
  { name: 'Albert', age: 76 },
  { name: 'Adam', age: 47 },
  { name: 'Robert', age: 72 }
];

let u1 = _.find(users, {name: 'Adam'});
console.log(u1);

let u2 = _.find(users, (u) => { return u.age > 60 });
console.log(u2);

let u3 = _.findLast(users, (u) => { return u.age > 60 });
console.log(u3);
```

In the example, we find the user whose name is Adam. Then we find the first and the last user who is older than sixty.

```
$ node main.js
{ name: 'Adam', age: 47 }
{ name: 'Peter', age: 81 }
{ name: 'Robert', age: 72 }
```

Lodash collection pull

The _.pull function removes all given values from the array.

main.js

```
const _ = require("lodash");

let nums = [1, 2, 3, 1, 2, 2, 4, 5, 7, 8];

_.pull(nums, 1, 2);
console.log(nums);
```

In the example, we have an array of numbers. We remove all numbers 1 and 2 from the array.

```
$ node main.js
[ 3, 4, 5, 7, 8 ]
```

Lodash collection take

The _.take function creates a slice of an array with n elements taken from the beginning. The _.takeRight function creates a slice of an array with n elements taken from the end.

main.js

```
const _ = require("lodash");

let nums = [1, 2, 3, 4, 5, 6];

let nums2 = _.take(nums);
let nums3 = _.take(nums, 2);
let nums4 = _.takeRight(nums, 3)

console.log(nums2);
console.log(nums3);
console.log(nums4);
```

The functions take some values from the array of integers and create new arrays. The second optional parameter specifies the number of values to take.

```
$ node main.js
[ 1 ]
[ 1, 2 ]
[ 4, 5, 6 ]
```

Lodash collection takeWhile

The _.takeWhile function creates a slice of an array with elements taken from the beginning. Elements are taken until the given predicate return false. In a similar fashion, the _.takeRightWhile function takes elements from the end.

main.js

```
const _ = require("lodash");

let nums = [1, -2, 3, 4, -5, 6, 7, -8, -9]

let nums2 = _.takeWhile(nums, (n) => { return n < 0 });
```

```
let nums3 = _.takeRightWhile(nums, (n) => { return n < 0 });

console.log(nums2);
console.log(nums3);
```

In the example, the predicate returns true for negative values.

```
$ node main.js
[]
[ -8, -9 ]
```

Lodash collection partition

The partition operation splits the original collection into a pair of arrays. The first array contains elements for which the specified predicate yields true, while the second list contains elements for which the predicate yields false.

main.js

```
const _ = require("lodash");

let nums = [4, -5, 3, 2, -1, 7, -6, 8, 9];
let [nums2, nums3] = _.partition(nums, (e) => e < 0);

console.log(nums2);
console.log(nums3);
```

With the _.partition function we split the array into two arrays; the first contains positive values, the second one negative valaues.

```
let [nums2, nums3] = _.partition(nums, (e) => e < 0);
```

The _.partition function creates two arrays according to the predicate function. With the array destructuring operation, we assign the created arrays to two variables: nums2 and nums3.

```
$ node main.js
[ -5, -1, -6 ]
[ 4, 3, 2, 7, 8, 9 ]
```

Lodash collection reduce

Reduction is a terminal operation that aggregates list values into a single value. The _.reduce function applies a function against an accumulator and each element in the array (from left to

right) to reduce it to a single value. The function that is being applied is called *reducer* function.

Note: reduction operations are really powerful. They can be used to calculate sums, products, averages, maximum and minimum values, sort, reverse, flatten arrays and many more.

For a more in-depth coverage of reductions, read the JavaScript reduce tutorial.

main.js

```
const _ = require("lodash");

let nums = [4, 5, 3, 2, 1, 7, 6, 8, 9];

let sum = _.reduce(nums, (total, next) => { return total + next });
console.log(sum);

let colours = ["red", "green", "white", "blue", "black"];

let res = _.reduceRight(colours, (next, total) => { return `${total}-${next}` });
console.log(res);
```

In the example, we use the reduce operation on a list of integers and strings.

```
let sum = _.reduce(nums, (total, next) => { return total + next });
console.log(sum);
```

We calculate the sum of values. The total is the accumulator, the next is the next value in the list.

```
let res = _.reduceRight(colours, (next, total) => { return `${total}-${next}` });
```

The _.reduceRight accumulates a value starting with last element and applying the operation from right to left to each element and the current accumulator value.

```
$ node main.js
45
red-green-white-blue-black
```

Lodash reduce - count occurrences

The reducer function can be used to count the occurrence of the elements in the array.

main.js

```
const _ = require("lodash");


let words = ['sky', 'forest', 'wood', 'sky', 'rock', 'cloud',
    'sky', 'forest', 'rock', 'sky'];

let tally = _.reduce(words, (total, next) => {

  total[next] = (total[next] || 0) + 1 ;

  return total;
}, {});


console.log(tally);
```

The thearray of words, everal words are included multiple times. The initial value is an empty object. The reducer function either creates a new property or increases the value of the property.

```
$ node main.js
{ sky: 4, forest: 2, wood: 1, rock: 2, cloud: 1 }
```

Lodash reduce - group objects by property

The following example groups objects in an array by a property.

main.js

```
const _ = require("lodash");

let users = [
  { name: 'John', age: 25, occupation: 'gardener' },
  { name: 'Lenny', age: 51, occupation: 'programmer' },
  { name: 'Andrew', age: 43, occupation: 'teacher' },
  { name: 'Peter', age: 52, occupation: 'gardener' },
  { name: 'Anna', age: 43, occupation: 'teacher' },
  { name: 'Albert', age: 46, occupation: 'programmer' },
  { name: 'Adam', age: 47, occupation: 'teacher' },
  { name: 'Robert', age: 32, occupation: 'driver' }
];

let grouped = _.reduce(users, (result, user) => {

    (result[user.occupation] || (result[user.occupation] = [])).push(user);
    return result;
}, {});

console.log(grouped);
```

We have an array of users. We group the users by their occupation. The initial value is an empty object. The resulting object has the occupations as properties; each property contains a list of users with the corresponding occupation.

```
let grouped = _.reduce(users, (result, user) => {

    (result[user.occupation] || (result[user.occupation] = [])).push(user);
    return result;
}, {});
```

The reducer either creates a new property with an empty array and pushes the first user or adds a new user object to already created array.

```
$ node main.js
{
  gardener: [
    { name: 'John', age: 25, occupation: 'gardener' },
    { name: 'Peter', age: 52, occupation: 'gardener' }
  ],
  programmer: [
    { name: 'Lenny', age: 51, occupation: 'programmer' },
    { name: 'Albert', age: 46, occupation: 'programmer' }
  ],
  teacher: [
    { name: 'Andrew', age: 43, occupation: 'teacher' },
    { name: 'Anna', age: 43, occupation: 'teacher' },
    { name: 'Adam', age: 47, occupation: 'teacher' }
  ],
  driver: [ { name: 'Robert', age: 32, occupation: 'driver' } ]
}
```

Lodash string case

Locash library contains several functions that work with the case of words.

main.js

```
const _ = require("lodash");

let words = ["sky", "Sun", "Blue Island"];

console.log(_.map(words, _.camelCase));
console.log(_.map(words, _.capitalize));
console.log(_.map(words, _.kebabCase));
console.log(_.map(words, _.lowerCase));
console.log(_.map(words, _.upperCase));
```

In the example, we modify the case of words with _.camelCase, _.capitalize, ._kebabCase, _lowerCase, and _.upperCase.

```
$ node main.js
[ 'sky', 'sun', 'blueIsland' ]
[ 'Sky', 'Sun', 'Blue island' ]
[ 'sky', 'sun', 'blue-island' ]
[ 'sky', 'sun', 'blue island' ]
[ 'SKY', 'SUN', 'BLUE ISLAND' ]
```

Lodash string _.startsWith and _.endsWith

The _.startsWith function determines if the string starts with the specified string. The _.endsWith function determines if the string ends with the specified string.

main.js

```
const _ = require("lodash");

let words = ["tank", "boy", "tourist", "ten",
        "pen", "car", "marble", "sonnet", "pleasant",
        "ink", "atom"]

console.log("Starting with 't'");
words.forEach( e => {

    if (_.startsWith(e, 't')) {

        console.log(e);
    }
});

console.log("Ending with 'k'");
words.forEach( e => {

    if (_.endsWith(e, 'k')) {

        console.log(e);
    }
});
```

In the example, we print the words that start with 't' and end with 'k'.

```
$ nodejs string_startend.js
Starting with 't'
tank
tourist
ten
Ending with 'k'
```

```
tank
ink
```

Lodash string padding

Strings can be padded with a character if they are shorter than a specified number.

main.js

```
const _ = require("lodash");

let nums = [657, 122, 3245, 345, 99, 18];

nums.forEach( e => {

    console.log(_.padStart(e.toString(), 20, '.'));
});
```

The example pads numbers with a dot character using _.padStart.

```
$ node main.js
..................657
..................122
.................3245
..................345
...................99
...................18
```

Lodash string trim

A string can be trimmed with _.trim, _.trimStart, and _.trimEnd functions. The default character to trim is a whitespace. We can provide our own characters to be trimmed.

main.js

```
const _ = require("lodash");

let word = '\tfalcon\t';

let trimmed = _.trim(word);
console.log(trimmed + 'owl');

let trimmed2 = _.trimStart(word);
console.log(trimmed2 + 'owl');

let trimmed3 = _.trimEnd(word);
console.log(trimmed3 + 'owl');
```

In the example, we trim whitespace characters from a word.

```
let trimmed = _.trim(word);
```

The _.trim function removes the whitespace characters from both sides.

```
let trimmed2 = _.trimStart(word);
```

The _.trimStart function removes the whitespace characters from the start of the string.

```
let trimmed3 = _.trimEnd(word);
```

The _.trimEnd function removes the whitespace characters from the end of the string.

```
$ node main.js
falconowl
falcon  owl
        falconowl
```

main.js

```
const _ = require("lodash")

let words = ['_falcon', '-owl-', '_-sky-_'];
let trimmed = _.map(words, (word) => { return _.trim(word, '-_')});

console.log(trimmed);
```

In this example, we remove two custom characters from an array of strings. We also use the _.map function.

```
$ node main.js
[ 'falcon', 'owl', 'sky' ]
```

Lodash object keys and values

The _.keys function returns an array of the property names of the JavaScript object and the _.values function returns an array of their values.

main.js

```
const _ = require("lodash");

let p = {age: 24, name: "Rebecca", occupation: "teacher"};
```

```
let keys = _.keys(p);
console.log(keys);

let values = _.values(p);
console.log(values);
```

In the example, we print the keys and values of a person object.

```
$ node main.js
[ 'age', 'name', 'occupation' ]
[ 24, 'Rebecca', 'teacher' ]
```

Lodash object picking

The _.pick function creates an object composed of the picked object properties.

main.js

```
const _ = require("lodash");

console.log(_.pick({ name: 'John', age: 25 }, 'name'));
console.log(_.pick({ name: 'John', age: 25 }, 'age'));
```

We pick properties from a simple object.

```
$ node main.js
{ name: 'John' }
{ age: 25 }
```

Lodash object at

The _.at function returns the value at the given object path.

main.js

```
const _ = require("lodash");

let users = [
  { id: 1, name: 'John', about: { 'age': 25, 'colours': ['red', 'green'], } },
  { id: 2, name: 'Lenny', about: { 'age': 51, 'colours': ['blue'], } },
  { id: 3, name: 'Andy', about: { 'age': 43, 'colours': ['orange', 'steelblue'], } },
  { id: 4, name: 'Peter', about: { 'age': 52, 'colours': ['black'], } },
  { id: 5, name: 'Anna', about: { 'age': 43, 'colours': ['purple'], } },
];


let name = _.at(users[2], 'name');
```

```
console.log(name);

let colour = _.at(users[0], 'about.colours[0]');
console.log(colour);
```

We have a list of objects. We get the values by specifying the path to the given property.

```
$ node main.js
[ 'Andrew' ]
[ 'red' ]
```

Lodash object get and set

The _.set function sets the value at the path of the object. If a portion of the path does not exist, it is created. The _.get function gets the value at the path of object; if the value does not exist, we can provide a default one.

main.js

```
const _ = require("lodash");

let data = { user: { name: "John Doe", age: 34, occupation: "gardener"} };

_.set(data, "user.age", 36);
console.log(data);

console.log(_.get(data, "user.name", "unknown"));
console.log(_.get(data, "user.marital_status", "unknown"));
```

In the example, we set a new age for the user, get his name and marital status. The marital status property does not exist, so we get the default unknown.

```
$ node main.js
{ user: { name: 'John Doe', age: 36, occupation: 'gardener' } }
John Doe
unknown
```

Lodash iterate object properties

The _.forIn function can be used to iterate over object properties.

main.js

```
const _ = require("lodash");

let p = {age: 24, name: "Rebecca", occupation: "teacher"};
```

```
_.forIn(p, (value, key) => {

    console.log(`${key}: ${value}`);
})
```

In the example, we iterate over properties of a person object using _.forIn.

```
$ node main.js
age: 24
name: Rebecca
occupation: teacher
```

In this article, we have introduced the Lodash JavaScript library.

List all JavaScript tutorials.