How to iterate over a JavaScript object?

Asked 11 years, 11 months ago Modified 28 days ago Viewed 853k times

669

I have an object in JavaScript:

```
{
    abc: '...',
    bca: '...',
    zzz: '...',
    xxx: '...',
    ccc: '...',
    // ...
}
```

I want to use a `for` loop to get its properties. And I want to iterate it in parts (not all object properties at once).

With a simple array I can do it with a standard `for` loop:

```
for (i = 0; i < 100; i++) { ... } // first part
for (i = 100; i < 300; i++) { ... } // second
for (i = 300; i < arr.length; i++) { ... } // last
```

But how to do it with objects?

javascript loops object iteration javascript-objects

Share

Improve this question

Follow

edited Sep 13, 2017 at 19:56

Sebastian Simon

19.4k ● 8 ● 60 ● 84

asked Jan 17, 2013 at 12:36

nkuhta

11.1k ● 12 ● 44 ● 55

35 Bear in mind that object properties are not stored in order. When you iterate over an object there is no guarantee to the order in which they will appear. – James Allardice Jan 17, 2013 at 12:40

Does this answer your question? How to loop through a plain JavaScript object with the objects as members? – shmuels Apr 29, 2020 at 16:01

2 @JamesAllardice's comment is from 2013. Since 2015 objects now have a defined iteration order. – AndreKR Sep 11, 2023 at 20:27

1 modern ECMAScript specification: array indices first in ascending order by value, then other string keys in ascending chronological order of property creation. – david.pfx Jan 29 at 12:08

19 Answers

Sorted by: Highest score (default)

1320

For iterating on keys of Arrays, Strings, or Objects, use `for .. in` :

```
for (let key in yourobject) {
  console.log(key, yourobject[key]);
}
```

With ES6, if you need both keys and values simultaneously, do

```
for (let [key, value] of Object.entries(yourobject)) {
    console.log(key, value);
}
```

To avoid logging inherited properties, check with hasOwnProperty :

```
for (let key in yourobject) {
   if (yourobject.hasOwnProperty(key)) {
      console.log(key, yourobject[key]);
   }
}
```

You don't need to check `hasOwnProperty` when iterating on keys if you're using a simple object (for example one you made yourself with `{}`).

This MDN documentation explains more generally how to deal with objects and their properties.

If you want to do it "in chunks", the best is to extract the keys in an array. As the order isn't guaranteed, this is the proper way. In modern browsers, you can use

```
let keys = Object.keys(yourobject);
```

To be more compatible, you'd better do this :

```
let keys = [];
for (let key in yourobject) {
    if (yourobject.hasOwnProperty(key)) keys.push(key);
}
```

Then you can iterate on your properties by index: `yourobject[keys[i]]` :

```
for (let i=300; i < keys.length && i < 600; i++) {
   console.log(keys[i], yourobject[keys[i]]);
}
```

Share

Improve this answer

Follow

edited May 27, 2022 at 9:04

answered Jan 17, 2013 at 12:37

Denys Séguret

382k ● 90 ● 809 ● 775

4 OP wants to perform this in chunks, not all keys in a single loop. – pawel Jan 17, 2013 at 12:39

2 @Cerbrus The OP allready knows how to iterate an array in parts. Using `keys` from the code given should be enough. – Yoshi Jan 17, 2013 at 12:47

2 @Cerbrus Please read before commenting ! What's not clear in *"To be more compatible, you'd better do this"* ? – Denys Séguret Jan 17, 2013 at 12:49

2 @am05mhz As I said, it's useless with most objects. But not for all. Try this: jsbin.com/hirivubuta/1/edit?js,console,output – Denys Séguret Sep 10, 2015 at 6:21

7 9 years developing in JS, and I always doubt myself with `for..of` and `for..in`, and ended up coming here. – choz Sep 23, 2020 at 17:59

101

Here is another iteration solution for modern browsers:

```
Object.keys(obj)
  .filter((k, i) => i >= 100 && i < 300)
  .forEach(k => console.log(obj[k]));
```

Or without the filter function:

```
Object.keys(obj).forEach((k, i) => {
    if (i >= 100 && i < 300) {
        console.log(obj[k]);
    }
});
```

However you must consider that properties in JavaScript object are not sorted, i.e. have no order.

Share

Improve this answer

Follow

edited Apr 15, 2020 at 8:46

Jacob van Lingen

9,517 ● 7 ● 52 ● 83

answered Jan 17, 2013 at 12:40

VisioN

145k ● 33 ● 286 ● 289

If I will break loop, it will start from beginning of object next time, that is not right way. – nkuhta Jan 17, 2013 at 12:43

Using `Object.entries` you do something like this.

54

```
// array like object with random key ordering
const anObj = { 100: 'a', 2: 'b', 7: 'c' };
console.log(Object.entries(anObj)); // [ ['2', 'b'],['7', 'c'],['100', 'a'] ]
```

The Object.entries() method returns an array of a given object's own enumerable property [key, value]

So you can iterate over the Object and have `key` and `value` for each of the object and get something like this.

```
const anObj = { 100: 'a', 2: 'b', 7: 'c' };
Object.entries(anObj).map(obj => {
   const key   = obj[0];
   const value = obj[1];

   // do whatever you want with those values.
});
```

or like this

```
// Or, using array extras
Object.entries(obj).forEach(([key, value]) => {
  console.log(`${key} ${value}`); // "a 5", "b 7", "c 9"
});
```

For a reference have a look at the MDN docs for Object Entries

Share

Improve this answer

Follow

edited Nov 22 at 16:17

answered Jan 30, 2018 at 6:12

Adeel Imran

13.9k ● 8 ● 65 ● 79

Even if you're not a fan of "map", entries() is the best thing they've added in this century.
– tekHedd Apr 13, 2023 at 21:30

22

With the new ES6/ES2015 features, you don't have to use an object anymore to iterate over a hash. You can use a Map. Javascript Maps keep keys in insertion order, meaning you can iterate over them without having to check the hasOwnProperty, which was always really a hack.

Iterate over a map:

```
var myMap = new Map();
myMap.set(0, "zero");
myMap.set(1, "one");
for (var [key, value] of myMap) {
  console.log(key + " = " + value);
```

```
}
// Will show 2 logs; first with "0 = zero" and second with "1 = one"

for (var key of myMap.keys()) {
  console.log(key);
}
// Will show 2 logs; first with "0" and second with "1"

for (var value of myMap.values()) {
  console.log(value);
}
// Will show 2 logs; first with "zero" and second with "one"

for (var [key, value] of myMap.entries()) {
  console.log(key + " = " + value);
}
// Will show 2 logs; first with "0 = zero" and second with "1 = one"
```

or use forEach:

```
myMap.forEach(function(value, key) {
  console.log(key + " = " + value);
}, myMap)
// Will show 2 logs; first with "0 = zero" and second with "1 = one"
```

Share Improve this answer Follow

answered Aug 4, 2016 at 3:42

Paul

1,056 ● 11 ● 18

2 forEach is the prefered one – pungggi Jun 3, 2018 at 9:59

19

If you want the *key and value* when iterating, you can use a for...of loop with Object.entries.

```
const myObj = {a: 1, b: 2}

for (let [key, value] of Object.entries(myObj)) {
    console.log(`key=${key} value=${value}`)
}

// output:
// key=a value=1
// key=b value=2
```

Share

Improve this answer

Follow

edited Sep 28, 2018 at 18:18

answered Sep 28, 2018 at 18:11

Derek Soike

11.6k ● 3 ● 82 ● 76

8

The only reliable way to do this would be to save your object data to 2 arrays, one of keys, and one for the data:

```
var keys = [];
var data = [];
for (var key in obj) {
    if (obj.hasOwnProperty(key)) {
        keys.push(key);
        data.push(obj[key]); // Not necessary, but cleaner, in my opinion. See
the example below.
    }
}
```

You can then iterate over the arrays like you normally would:

```
for(var i = 0; i < 100; i++){
    console.log(keys[i], data[i]);
    //or
    console.log(keys[i], obj[keys[i]]); // harder to read, I think.
}
for(var i = 100; i < 300; i++){
    console.log(keys[i], data[i]);
}
```

I am not using `Object.keys(obj)`, because that's IE 9+.

Share

Improve this answer

Follow

edited Feb 15, 2018 at 13:06

answered Jan 17, 2013 at 12:44

Cerbrus

72.7k • 19 • 136 • 150

5

->if we iterate over a JavaScript object using and find key of array of objects

```
Object.keys(Array).forEach(key => {

 console.log('key',key)

})
```

Share Improve this answer Follow

answered Dec 14, 2017 at 5:56

ashishdudhat

190 • 3 • 9

5

Define object in arguments and avoid selectors & subscripts

There are a number of syntax choices but this one defines the object upfront in the closure's arguments which eliminates the need for selectors or subscripts in the

iterator. `k` is key, `v` is value, `i` is index.

```
const obj = {
    kiwi: true,
    mango: false,
    pineapple: 500
};

Object.entries(obj).forEach(([k, v], i) => {
    console.log(k, v, i);
});

// kiwi true 0
// mango false 1
// pineapple 500 2
```

Share

Improve this answer

Follow

edited Jun 30, 2022 at 20:44

answered May 20, 2022 at 20:07

slushy

12.3k ● 3 ● 44 ● 56

4

> If you have a simple object you can iterate through it using the following code:

```
let myObj = {
  abc: '...',
  bca: '...',
  zzz: '...',
  xxx: '...',
  ccc: '...',
  // ...
};

let objKeys = Object.keys(myObj);

//Now we can use objKeys to iterate over myObj

for (item of objKeys) {
  //this will print out the keys
  console.log('key:', item);

  //this will print out the values
  console.log('value:', myObj[item]);
}
```

Run code snippet

Expand snippet

> If you have a nested object you can iterate through it using the following code:

```
let b = {
  one: {
    a: 1,
    b: 2,
    c: 3
  },
  two: {
    a: 4,
    b: 5,
    c: 6
  },
  three: {
    a: 7,
    b: 8,
    c: 9
  }
};

let myKeys = Object.keys(b);

for (item of myKeys) {
  //print the key
  console.log('Key', item)

  //print the value (which will be another object)
  console.log('Value', b[item])

  //print the nested value
  console.log('Nested value', b[item]['a'])
}
```

Run code snippet Expand snippet

> If you have array of objects you can iterate through it using the following code:

```
let c = [
{
  a: 1,
  b: 2
},
{
  a: 3,
  b: 4
}
];
```

```
for(item of c){
//print the whole object individually
console.log('object', item);

//print the value inside the object
console.log('value', item['a']);
}
```

Run code snippet Expand snippet

Share Improve this answer Follow

answered Jun 26, 2020 at 9:56

Hadi Mir

5,093 ● 2 ● 32 ● 33

2

If you wanted to iterate the whole object at once you could use `for in` loop:

```
for (var i in obj) {
  ...
}
```

But if you want to divide the object into parts in fact you cannot. There's no guarantee that properties in the object are in any specified order. Therefore, I can think of two solutions.

First of them is to "remove" already read properties:

```
var i = 0;
for (var key in obj) {
    console.log(obj[key]);
    delete obj[key];
    if ( ++i > 300) break;
}
```

Another solution I can think of is to use Array of Arrays instead of the object:

```
var obj = [['key1', 'value1'], ['key2', 'value2']];
```

Then, standard `for` loop will work.

Share

Improve this answer

Follow

edited Jul 15, 2014 at 12:15

FranciscoBouza

620 ● 7 ● 20

answered Jan 17, 2013 at 12:48

Michał Miszczyszyn

12.7k ● 2 ● 37 ● 53

2

```
const o = {
  name: "Max",
  location: "London"
};

for (const [key, value] of Object.entries(o)) {
  console.log(`${key}: ${value}`);
}
```

[Try online]

Share Improve this answer Follow

edited Oct 3, 2019 at 17:25

answered Oct 3, 2019 at 17:11

Vishal

20.6k • 23 • 82 • 95

1

I finally came up with a handy utility function with a unified interface to iterate Objects, Strings, Arrays, TypedArrays, Maps, Sets, (any Iterables).

```
const iterate = require('@a-z/iterate-it');
const obj = { a: 1, b: 2, c: 3 };

iterate(obj, (value, key) => console.log(key, value));
// a 1
// b 2
// c 3
```

https://github.com/alrik/iterate-javascript

Share Improve this answer Follow

answered Feb 15, 2018 at 23:10

Alrik Zachert

29 • 5

1

Really a PITA this is not part of standard Javascript.

```
/**
 * Iterates the keys and values of an object.  Object.keys is used to extract
the keys.
 * @param object The object to iterate
 * @param fn (value,key)=>{}
 */
function objectForEach(object, fn) {
    Object.keys(object).forEach(key => {
        fn(object[key],key, object)
    })
}
```

Note: I switched the callback parameters to (value,key) and added a third object to make the API consistent other APIs.

Use it like this

```
const o = {a:1, b:true};
objectForEach(o, (value, key, obj)=>{
    // do something
});
```

Share

Improve this answer

Follow

edited Jul 3, 2018 at 13:30

answered Apr 28, 2018 at 2:14

Steven Spungin

29k • 7 • 95 • 83

1 upvoted just for your statement in the first sentence. Even though it'd be better if the value was first parameter, the index or key second parameter, and the object third parameter, to make it more like the array forEach(). I'd recommend recommending lodash though. – CONTRACT SAYS I'M RIGHT Jun 12, 2018 at 11:14

I do like the idea of the (value, key) order. That is how a library such as Vue does it too. Because the object is the context, it do think it belongs as the first parameter though. That's pretty standard for functional programming. – Steven Spungin Jun 12, 2018 at 15:42

I would agree here, were it not for ECMA-262 defining an array as an object having a forEach(), map(), reduce(), filter(), which all take callbacks receiving the order [value, index, array]. An object in JS can be understood as just another collection; and then these methods become unified in their parameters of [value, key|index, context] (this is what lodash and underscore are doing). In my opinion, this "unified collection" protocol is just stronger. Also, the object *isn't* the context: you can set `this` to whatever you like for the callback, as the callback has its own context. – CONTRACT SAYS I'M RIGHT Jun 12, 2018 at 15:53

Perhaps I should have used the work receiver instead of this. Anyway still a PITA; I would welcome the parameters in any order. – Steven Spungin Jun 12, 2018 at 17:55

Oh, I see that we might have misunderstood each other. I was always commenting about the callback parameters and their order, not about the actual `objectForEach` function. Sorry if that was confusing. – CONTRACT SAYS I'M RIGHT Jun 12, 2018 at 21:25

1

For object iteration we usually use a `for..in` loop. This structure will loop through all **enumerable** properties, including ones who are inherited via prototypal inheritance. For example:

```
let obj = {
  prop1: '1',
  prop2: '2'
}

for(let el in obj) {
  console.log(el);
```

```
  console.log(obj[el]);
}
```

Run code snippet Expand snippet

However, `for..in` will loop over all enumerable elements and this will not able us to split the iteration in chunks. To achieve this we can use the built in `Object.keys()` function to retrieve all the keys of an object in an array. We then can split up the iteration into multiple for loops and access the properties using the keys array. For example:

```
let obj = {
  prop1: '1',
  prop2: '2',
  prop3: '3',
  prop4: '4',
};

const keys = Object.keys(obj);
console.log(keys);


for (let i = 0; i < 2; i++) {
  console.log(obj[keys[i]]);
}


for (let i = 2; i < 4; i++) {
  console.log(obj[keys[i]]);
}
```

Run code snippet Expand snippet

Share Improve this answer Follow

answered Sep 28, 2018 at 8:09

Willem van der Veen

36.4k • 18 • 204 • 176

1

Yes. You can loop through an object using for loop. Here is an example

```
var myObj = {
    abc: 'ABC',
    bca: 'BCA',
    zzz: 'ZZZ',
    xxx: 'XXX',
    ccc: 'CCC',
}
```

```
var k = Object.keys (myObj);
for (var i = 0; i < k.length; i++) {
    console.log (k[i] + ": " + myObj[k[i]]);
}
```

Run code snippet Expand snippet

NOTE: the example mentioned above will only work in IE9+. See Objec.keys browser support here.

Share Improve this answer Follow

answered May 28, 2019 at 8:14

Omprakash Arumugam

1,034 ● 8 ● 10

1

Here is a Hand Made Solution:

```
function iterationForObject() {
    let base = 0,
        Keys= Object.keys(this);
    return {
        next: () => {
            return {
                value: {
                    "key": Keys[base],
                    "value": this[Keys[base]]
                },
                done: !(base++ < Keys.length)
            };
        }
    };
}
Object.prototype[Symbol.iterator] = iterationForObject;
```

And Then You Can Loop Any Object:

```
for ( let keyAndValuePair of (Object Here) ) {
    console.log(`${keyAndValuePair.key} => ${keyAndValuePair.value}`);
}
```

Share Improve this answer Follow

answered Aug 19, 2020 at 12:13

Kareem Medhat

11 ● 1 ● 2

```
<script type="text/javascript">
// method 1
```

1

```
var images = {};
images['name'] = {};
images['family'] = {};
images[1] = {};
images['name'][5] = "Mehdi";
images['family'][8] = "Mohammadpour";
images['family']['ok'] = 123456;
images[1][22] = 2602;
images[1][22] = 2602;
images[1][22] = 2602;
images[1][22] = 2602;
images[1][23] = 2602;

for (const [key1, value1] of Object.entries(images)){
    for (const [key2, value2] of Object.entries(value1)){
        console.log(`${key1} => ${key2}: ${value2}`);
    }
}


console.log("============================");

// method 2
var arr = [];
for(var x = 0; x < 5; x++){
     arr[x] = [];
     for(var y = 0; y < 5; y++){
         arr[x][y] = x*y;
     }
 }

for(var i = 0; i < arr.length; i++) {
    var cube = arr[i];
    for(var j = 0; j < cube.length; j++) {
        console.log("cube[" + i + "][" + j + "] = " + cube[j]);
    }
}

</script>
```

Share Improve this answer Follow

answered Aug 26, 2020 at 6:21

Mehdi Mohammadpour

1,120 ● 1 ● 7 ● 11

0

```
var Dictionary = {
  If: {
    you: {
      can: '',
      make: ''
    },
    sense: ''
  },
  of: {
    the: {
      sentence: {
        it: '',
        worked: ''
```

```
      }
    }
  }
};

function Iterate(obj) {
  for (prop in obj) {
    if (obj.hasOwnProperty(prop) && isNaN(prop)) {
      console.log(prop + ': ' + obj[prop]);
      Iterate(obj[prop]);
    }
  }
}
Iterate(Dictionary);
```

Share Improve this answer Follow

answered Jul 3, 2016 at 22:51

HovyTech

389 • 6 • 19

1 Actually no. This implies that `Object`s are in-order. They're not. `If you can make sense of the sentence it worked` only works because of implementation details. It's not guaranteed to work at all. Also you shouldn't TitleCase your functions & variables. That's for `class`es. – Florian Wendelborn Feb 19, 2017 at 12:46

0

You can try using lodash- A modern JavaScript utility library delivering modularity, performance & extras js to fast object iterate:-

```
var  users  =   {
    'fred':     {
        'user':   'fred',
            'age':  40
    },
    'pebbles':  {
        'user':   'pebbles',
         'age':  1
    }
};
_.mapValues(users,  function(o)  {
    return  o.age;
});
// => { 'fred': 40, 'pebbles': 1 } (iteration order is not guaranteed)
// The `_.property` iteratee shorthand.
console.log(_.mapValues(users,  'age')); // returns age property & value
console.log(_.mapValues(users,  'user')); // returns user property & value
console.log(_.mapValues(users)); // returns all objects
// => { 'fred': 40, 'pebbles': 1 } (iteration order is not guaranteed)
```

```
<script src="https://cdnjs.cloudflare.com/ajax/libs/lodash-
compat/3.10.2/lodash.js"></script>
```

Run code snippet Expand snippet

Share

Improve this answer

Follow

edited Apr 24, 2018 at 11:53

answered Apr 9, 2018 at 6:34

Parth Raval

4,395 3 24 39