

# Destructuring assignment

[Jump to section ▼](#)

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.



## JavaScript Demo: Expressions - Destructuring assignment

```
1 let a, b, rest;
2 [a, b] = [10, 20];
3
4 console.log(a);
5 // expected output: 10
6
7 console.log(b);
8 // expected output: 20
```

```
9
10 [a, b, ...rest] = [10, 20, 30, 40, 50];
11
12 console.log(rest);
13 // expected output: Array [30,40,50]
14
```

Run ›

Reset

---

# Syntax

```
1  let a, b, rest;
2  [a, b] = [10, 20];
3  console.log(a); // 10
4  console.log(b); // 20
5
6  [a, b, ...rest] = [10, 20, 30, 40, 50];
7  console.log(a); // 10
8  console.log(b); // 20
9  console.log(rest); // [30, 40, 50]
10
11 ({ a, b } = { a: 10, b: 20 });
12 console.log(a); // 10
13 console.log(b); // 20
14
15
16 // Stage 4(finished) proposal
17 ({a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40});
18 console.log(a); // 10
19 console.log(b); // 20
20 console.log(rest); // {c: 30, d: 40}
```

---

## Description

The object and array literal expressions provide an easy way to create *ad hoc* packages of data.

```
1 | const x = [1, 2, 3, 4, 5];
```

The destructuring assignment uses similar syntax, but on the left-hand side of the assignment to define what values to unpack from the sourced variable.

```
1 | const x = [1, 2, 3, 4, 5];  
2 | const [y, z] = x;  
3 | console.log(y); // 1  
4 | console.log(z); // 2
```

This capability is similar to features present in languages such as Perl and Python.

---

## Examples

### Array destructuring

## Basic variable assignment

```
1 | const foo = ['one', 'two', 'three'];
2 |
3 | const [red, yellow, green] = foo;
4 | console.log(red); // "one"
5 | console.log(yellow); // "two"
6 | console.log(green); // "three"
```

## Assignment separate from declaration

A variable can be assigned its value via destructuring separate from the variable's declaration.

```
1 | let a, b;
2 |
3 | [a, b] = [1, 2];
4 | console.log(a); // 1
5 | console.log(b); // 2
```

## Default values

A variable can be assigned a default, in the case that the value unpacked from the array is `undefined`.

```
1 | let a, b;  
2 |  
3 | [a=5, b=7] = [1];  
4 | console.log(a); // 1  
5 | console.log(b); // 7
```

## Swapping variables

Two variables values can be swapped in one destructuring expression.

Without destructuring assignment, swapping two values requires a temporary variable (or, in some low-level languages, the [XOR-swap trick](#)).

```
1 | let a = 1;  
2 | let b = 3;  
3 |  
4 | [a, b] = [b, a];  
5 | console.log(a); // 3  
6 | console.log(b); // 1  
7 |  
8 | const arr = [1,2,3];  
9 | [arr[2], arr[1]] = [arr[1], arr[2]];  
10 | console.log(arr); // [1,3,2]  
11 |
```

## Parsing an array returned from a function

It's always been possible to return an array from a function. Destructuring can make working with an array return value more concise.

In this example, `f()` returns the values `[1, 2]` as its output, which can be parsed in a single line with destructuring.

```
1 function f() {  
2   return [1, 2];  
3 }  
4  
5 let a, b;  
6 [a, b] = f();  
7 console.log(a); // 1  
8 console.log(b); // 2
```

## Ignoring some returned values

You can ignore return values that you're not interested in:

```
1 function f() {  
2   return [1, 2, 3];  
3 }  
4  
5 const [a, , b] = f();
```

```
6 | console.log(a); // 1
7 | console.log(b); // 3
8 |
9 | const [c] = f();
10| console.log(c); // 1
```

You can also ignore all returned values:

```
1 | [, ,] = f();
```

## Assigning the rest of an array to a variable

When destructuring an array, you can unpack and assign the remaining part of it to a variable using the rest pattern:

```
1 | const [a, ...b] = [1, 2, 3];
2 | console.log(a); // 1
3 | console.log(b); // [2, 3]
```

Be aware that a `SyntaxError` will be thrown if a trailing comma is used on the left-hand side with a rest element:

```
1 | const [a, ...b,] = [1, 2, 3];
2 |
```



```
3 // SyntaxError: rest element may not have a trailing comma
4 // Always consider using rest operator as the last element
```



## Unpacking values from a regular expression match

When the regular expression `exec()` method finds a match, it returns an array containing first the entire matched portion of the string and then the portions of the string that matched each parenthesized group in the regular expression. Destructuring assignment allows you to unpack the parts out of this array easily, ignoring the full match if it is not needed.

```
1 function parseProtocol(url) {
2   const parsedURL = /^(\\w+)\\:\\\\(\\[\\^\\/]\\+\\)(\\.*)$/ .exec(url);
3   if (!parsedURL) {
4     return false;
5   }
6   console.log(parsedURL);
7   // ["https://developer.mozilla.org/en-US/Web/JavaScript",
8     "https", "developer.mozilla.org", "en-US/Web/JavaScript"]
9
10  const [, protocol, fullhost, fullpath] = parsedURL;
11  return protocol;
12 }
13
14 console.log(parseProtocol('https://developer.mozilla.org/en-US/Web/JavaScript'));
15 // "https"
```

# Object destructuring

## Basic assignment

```
1 | const user = {  
2 |     id: 42,  
3 |     is_verified: true  
4 | };  
5 |  
6 | const {id, is_verified} = user;  
7 |  
8 | console.log(id); // 42  
9 | console.log(is_verified); // true
```

## Assignment without declaration

A variable can be assigned its value with destructuring separate from its declaration.

```
1 | let a, b;  
2 |  
3 | ({a, b} = {a: 1, b: 2});
```

**Notes:** The parentheses ( ... ) around the assignment statement are required when using object literal destructuring assignment without a declaration.

`{a, b} = {a: 1, b: 2}` is not valid stand-alone syntax, as the `{a, b}` on the left-hand side is considered a block and not an object literal.

However, `({a, b} = {a: 1, b: 2})` is valid, as is `const {a, b} = {a: 1, b: 2}`

Your ( ... ) expression needs to be preceded by a semicolon or it may be used to execute a function on the previous line.

## Assigning to new variable names

A property can be unpacked from an object and assigned to a variable with a different name than the object property.

```
1 | const o = {p: 42, q: true};
2 | const {p: foo, q: bar} = o;
3 |
4 | console.log(foo); // 42
5 | console.log(bar); // true
```

Here, for example, `const {p: foo} = o` takes from the object `o` the property named `p` and assigns it to a local variable named `foo`.

## Default values

A variable can be assigned a default, in the case that the value unpacked from the object is `undefined`.

```
1 | const {a = 10, b = 5} = {a: 3};  
2 |  
3 | console.log(a); // 3  
4 | console.log(b); // 5
```

## Assigning to new variables names and providing default values

A property can be both 1) unpacked from an object and assigned to a variable with a different name and 2) assigned a default value in case the unpacked value is `undefined`.

```
1 | const {a: aa = 10, b: bb = 5} = {a: 3};  
2 |  
3 | console.log(aa); // 3  
4 | console.log(bb); // 5
```

## Unpacking fields from objects passed as function parameter

```
1 | const user = {  
2 |   id: 42,
```

```
3   displayName: 'jdoe',
4   fullName: {
5     firstName: 'John',
6     lastName: 'Doe'
7   }
8 };
9
10 function userId({id}) {
11   return id;
12 }
13
14 function whois({displayName, fullName: {firstName: name}}) {
15   return `${displayName} is ${name}`;
16 }
17
18 console.log(userId(user)); // 42
19 console.log(whois(user)); // "jdoe is John"
```

This unpacks the `id`, `displayName` and `firstName` from the `user` object and prints them.

## Setting a function parameter's default value

```
1   function drawChart({size = 'big', coords = {x: 0, y: 0}, radius = 25} = {}) {
2     console.log(size, coords, radius);
3     // do some chart drawing
4   }
5
```

```
6 | drawChart({
7 |   coords: {x: 18, y: 30},
8 |   radius: 30
9 | });
```

■ In the function signature for **drawChart** above, the destructured left-hand side is assigned to an empty object literal on the right-hand side: `{size = 'big', coords = {x: 0, y: 0}, radius = 25} = {}`. You could have also written the function without the right-hand side assignment. However, if you leave out the right-hand side assignment, the function will look for at least one argument to be supplied when invoked, whereas in its current form, you can simply call **drawChart()** without supplying any parameters. The current design is useful if you want to be able to call the function without supplying any parameters, the other can be useful when you want to ensure an object is passed to the function.

## Nested object and array destructuring

```
1 | const metadata = {
2 |   title: 'Scratchpad',
3 |   translations: [
4 |     {
5 |       locale: 'de',
6 |       localization_tags: [],
7 |       last_edit: '2014-04-14T08:43:37',
8 |       url: '/de/docs/Tools/Scratchpad',
9 |       title: 'JavaScript-Umgebung'
```

```
10     }
11   ],
12   url: '/en-US/docs/Tools/Scratchpad'
13 };
14
15 let {
16   title: englishTitle, // rename
17   translations: [
18     {
19       title: localeTitle, // rename
20     },
21   ],
22 } = metadata;
23
24 console.log(englishTitle); // "Scratchpad"
25 console.log(localeTitle); // "JavaScript-Umgebung"
```

## For of iteration and destructuring

```
1  const people = [
2    {
3      name: 'Mike Smith',
4      family: {
5        mother: 'Jane Smith',
6        father: 'Harry Smith',
7        sister: 'Samantha Smith'
```

```
8     },
9     age: 35
10  },
11  {
12    name: 'Tom Jones',
13    family: {
14      mother: 'Norah Jones',
15      father: 'Richard Jones',
16      brother: 'Howard Jones'
17    },
18    age: 25
19  }
20 ];
21
22 for (const {name: n, family: {father: f}} of people) {
23   console.log('Name: ' + n + ', Father: ' + f);
24 }
25
26 // "Name: Mike Smith, Father: Harry Smith"
27 // "Name: Tom Jones, Father: Richard Jones"
```

## Computed object property names and destructuring

Computed property names, like on [object literals](#), can be used with destructuring.



```
1 | let key = 'z';
2 | let {[key]: foo} = {z: 'bar'};
3 |
4 | console.log(foo); // "bar"
```

## Rest in Object Destructuring

The [Rest/Spread Properties for ECMAScript](#) proposal (stage 4) adds the `rest` syntax to destructuring. Rest properties collect the remaining own enumerable property keys that are not already picked off by the destructuring pattern.

```
1 | let {a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40}
2 | a; // 10
3 | b; // 20
4 | rest; // { c: 30, d: 40 }
```

## Invalid JavaScript identifier as a property name

Destructuring can be used with property names that are not valid JavaScript `identifiers` by providing an alternative identifier that is valid.

```
1 | const foo = { 'fizz-buzz': true };
2 | const { 'fizz-buzz': fizzBuzz } = foo;
3 |
```

4

```
console.log(fizzBuzz); // "true"
```

## Combined Array and Object Destructuring

Array and Object destructuring can be combined. Say you want the third element in the array `props` below, and then you want the `name` property in the object, you can do the following:

```
1  const props = [  
2    { id: 1, name: 'Fizz' },  
3    { id: 2, name: 'Buzz' },  
4    { id: 3, name: 'FizzBuzz' }  
5  ];  
6  
7  const [, , { name }] = props;  
8  
9  console.log(name); // "FizzBuzz"
```

## The prototype chain is looked up when the object is deconstructed

When deconstructing an object, if a property is not accessed in itself, it will continue to look up along the prototype chain.

```
1  let obj = {self: '123'};  
2  obj.__proto__.prot = '456';
```

```
3 | const {self, prot} = obj;  
4 | // self "123"  
5 | // prot "456" (Access to the prototype chain)
```

---

## Specifications

### Specification

ECMAScript (ECMA-262)

The definition of 'Destructuring assignment' in that specification.

---

## Browser compatibility

[Update compatibility data on GitHub](#)

Desktop						Mobile						Node.js
Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	Android webview	Chrome for Android	Firefox for Android	Opera for Android	Safari on iOS	Samsung Internet	Node.js
Destructuring assignment												
49	14	41 ★	No	36	8	49	49	41 ★	36	8	5.0	6.0.0
Computed property names												
49	14	41	No	36	10	49	49	41	36	10	5.0	6.0.0
Rest in arrays												
49	14 🚩	41	No	36	9.1	49	49	41	36	9.3	5.0	6.0.0
Rest in objects 🧪												
60	79	55	No	47	11.1	60	60	55	44	11.3	8.0	8.3.0

What are we missing?



Full support



No support



Experimental. Expect behavior to change in the future.



See implementation notes.



User must explicitly enable this feature.

---

## See also

- [Assignment operators](#)
- ["ES6 in Depth: Destructuring" on hacks.mozilla.org](#)

---

 **Last modified:** Jun 2, 2020, by [MDN contributors](#)

## Related Topics

*JavaScript*

### Tutorials:

- ▶ [Complete beginners](#)

► JavaScript Guide

► Intermediate

► Advanced

## References:

► Built-in objects

▼ Expressions & operators

Addition (+)

Addition assignment (+=)

Assignment (=)

Bitwise AND (&)

Bitwise AND assignment (&=)

Bitwise NOT (~)

Bitwise OR (|)

Bitwise OR assignment (|=)

Bitwise XOR (^)

Bitwise XOR assignment (^=)

Comma operator (,)

Conditional (ternary) operator

Decrement (--)

Destructuring assignment

Division (/)

Division assignment (/=)

Equality (==)

Exponentiation (\*\*)

Exponentiation assignment (\*\*=)

Function expression

Greater than (>)

Greater than or equal (>=)

Grouping operator ( )

Increment (++)

Inequality (!=)

Left shift (<<)

Left shift assignment (<<=)

Less than (<)

Less than or equal (<=)

Logical AND (&&)

Logical NOT (!)

Logical OR (||)

Multiplication (\*)

Multiplication assignment (\*=)

Nullish coalescing operator (??)

Object initializer

Object instances

Operator precedence

Optional chaining (?.)

 Pipeline operator

Property accessors

Remainder (%)

Remainder assignment (%=)

Right shift (>>)

Right shift assignment (>>=)

Spread syntax (...)

Strict equality (===)

Strict inequality (!==)

Subtraction (-)

Subtraction assignment (--)

Unary negation (-)

Unary plus (+)

Unsigned right shift (>>>)

Unsigned right shift assignment (>>>=)

async function expression

await

class expression

delete operator



function\* expression

in operator

instanceof

new operator

new.target

super

this

typeof

void operator

yield

yield\*

► Statements & declarations

► Functions

► Classes

► Errors

► Misc



# Learn the best of web development

Get the latest and greatest from MDN delivered straight to your inbox.

[Sign up now](#)



[Web Technologies](#)

[Learn Web Development](#)

[About MDN](#)

[Feedback](#)



[MDN](#)  

[About](#)

[MDN Web Docs Store](#)

[Contact Us](#)

[Firefox](#)

[Mozilla](#)  

---

© 2005-2020 Mozilla and individual contributors. Content is available under these licenses.

[Terms](#) [Privacy](#) [Cookies](#)