
MDN Web Docs
moz://a

Destructuring assignment

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

```
let a, b, rest;
[a, b] = [10, 20];

console.log(a);
// expected output: 10

console.log(b);
// expected output: 20

[a, b, ...rest] = [10, 20, 30, 40, 50];

console.log(rest);
// expected output: Array [30,40,50]

```

Run ›

Reset

Syntax

```
let a, b, rest;
[a, b] = [10, 20];
console.log(a); // 10
console.log(b); // 20

[a, b, ...rest] = [10, 20, 30, 40, 50];
```

```
console.log(a); // 10
console.log(b); // 20

// Stage 4(finished) proposal
({a, b, ...rest} = {a: 10, b: 20, c: 30, d: 40});
console.log(a); // 10
console.log(b); // 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.

```
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.

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

Similarly, you can destructure arrays on the left-hand side of the assignment

```
const [firstElement, secondElement] = list;
// is equivalent to:
// const firstElement = list[0];
// const secondElement = list[1];
```

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

Examples

Array destructuring

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

```
let a, b;

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

In an array destructuring from an array of length *N* specified on the right-hand side of the assignment, if the number of variables specified on the left-hand side of the assignment is greater than *N*, only the first *N* variables are assigned values. The values of the remaining variables will be undefined.

```
const foo = ['one', 'two'];

const [red, yellow, green, blue] = foo;
console.log(red); // "one"
console.log(yellow); // "two"
console.log(green); // undefined
console.log(blue);  //undefined
```

Default values

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

```
let a, b;

[a=5, b=7] = [1];
console.log(a); // 1
console.log(b); // 7
```

Swapping variables

```
console.log(arr); // [1,3,2]
```

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.

```
function f() {
  return [1, 2];
}

let a, b;
[a, b] = f();
console.log(a); // 1
console.log(b); // 2
```

Ignoring some returned values

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

```
function f() {
  return [1, 2, 3];
}

const [a, , b] = f();
```

side of a rest element:

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

// SyntaxError: rest element may not have a trailing comma
// 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.

```
function parseProtocol(url) {
  const parsedURL = /^(\w+)\:\/\/([^\/]+)\/(.*)$/.exec(url);
  if (!parsedURL) {
    return false;
  }
  console.log(parsedURL);
  // ["https://developer.mozilla.org/en-US/docs/Web/JavaScript",
      "https", "developer.mozilla.org", "en-US/docs/Web/JavaScript"]
```

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

> **Note:** 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.

```
const o = {p: 42, q: true};
```

Unpacking fields from objects passed as a function parameter

```
const user = {
  id: 42,
  displayName: 'jdoe',
  fullName: {
    firstName: 'John',
    lastName: 'Doe'
  }
};

function userId({id}) {
  return id;
}
```

```
const metadata = {
  title: 'Scratchpad',
  translations: [
    {
      locale: 'de',
      localization_tags: [],
      last_edit: '2014-04-14T08:43:37',
      url: '/de/docs/Tools/Scratchpad',
      title: 'JavaScript-Umgebung'
```

```
  console.log('Name: ' + n + ', Father: ' + f);
}

// "Name: Mike Smith, Father: Harry Smith"
```