Array & Object Destructuring

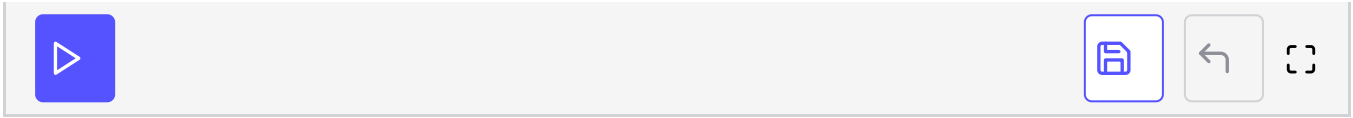
Array and object destructuring can truly change the way we write a lot of our code. They greatly clean up our code by reducing the number of lines we need to write when working with objects and arrays. We'll cover their nuances and show how they work in every situation.

> Array and object destructuring can truly change the way we write a lot of our code. Let's dive right in and explain by example.
>
> Before, to get values out of an array or object, we'd have to use direct assignment.

```
var arr = [1, 2, 3];
var one = arr[0];
var two = arr[1];
var three = arr[2];
console.log(one, two, three); // -> 1 2 3

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

var obj = {
    key1: 'val1',
    key2: 'val2'
};
var key1 = obj.key1;
var key2 = obj.key2;
console.log(key1, key2); // -> val1 val2
```



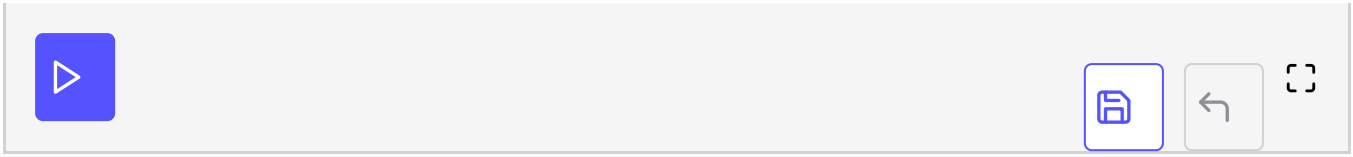
> Now we have a new tool.

```
const arr = [1, 2, 3];
const [one, two, three] = arr;
console.log(one, two, three); // -> 1 2 3

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

const obj = {
    key1: 'val1',
    key2: 'val2'
};
const { key1, key2 } = obj;
```

```
console.log(key1, key2); // -> val1 val2
```



We've just witnessed a new way to assign values to variables. It's as simple as following the pattern shown. It's called destructuring and can be thought of as breaking down the object/array into its individual components.

Line 2 above pulls the numbers `1`, `2`, and `3` out of the array and gives them to the variables `one`, `two`, and `three`, respectively. Any variable names can be used. These are constant variables, as we used the keyword `const` in the destructuring assignment. For array destructuring, the order of the variables maps over to the order of items in the array.

Object destructuring is a little different. Line 11 looks for the properties `key1` and `key2` on `obj`. It finds them and places their values into the variables.

It's a new pattern for variable assignment. It makes it a little easier to strip items and properties off of arrays and objects.

The keyword we use when destructuring matters. If we use `const`, the variables we declare are all constant. If we use `let` or `var` (although you should never use `var`), it'll follow those scoping rules.

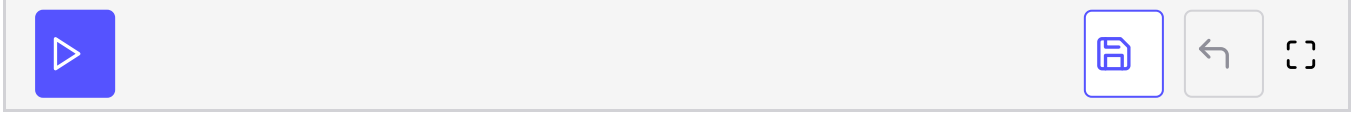
Assigning Object Properties to Renamed Variables

With object destructuring, we can assign object properties to any variable name. We just use a colon and place the variable name we want afterward.

```
const obj = {
    key1: 'val1',
    key2: 'val2'
};

const {
  key1: key1changed,
  key2: key2changed
} = obj;

console.log(key1changed, key2changed); // -> val1 val2
```

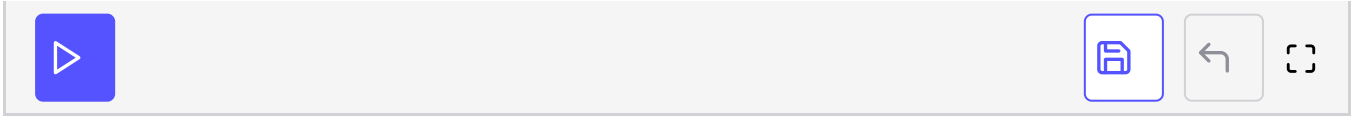


Even if the original property name isn't a proper variable, we can destructure it.

```
const obj = {
    '#': 'hashValue',
    '&': 'ampersandValue'
};

const {
    '#': hash,
    '&': ampersand
} = obj;

console.log(hash, ampersand); // -> hashValue ampersandValue
```

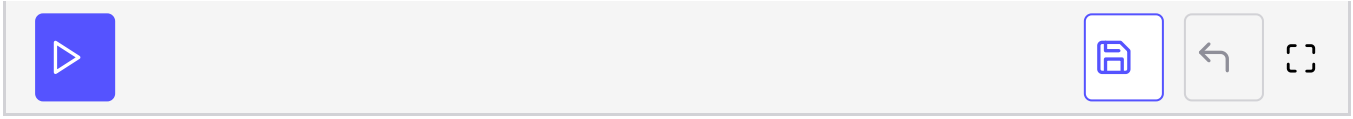


Default Values with Destructuring

Similar to default values in functions, we can use default values when destructuring.

```
const arr = [1, 2];
const [one = 'one', two = 'two', three = 'three'] = arr;
console.log(one, two, three); // -> 1 2 'three'

const obj = { key1: 'val1' };
const {
    key1 = 'val1Default',
    key2 = 'val2Default'
} = obj;
console.log(key1, key2); // -> val1 val2Default
```



Destructuring in Function Parameters

Destructuring in function parameters is an especially powerful tool. It's considered bad practice to have a function accept more than three or four parameters. It makes it difficult to keep track of the order and a mistake can be difficult to debug.

Instead, if a function needs lots of variables, we can pass in an object. Inside

the function, to work with the individual properties, we'd pick them off one by

one. This is how it would be done pre-ES2015.

```
function fn(obj) {
    // We want to strip off properties to use individually.
    var firstName = obj.firstName;
    var lastName = obj.lastName;
    var age = obj.age;
    var occupation = obj.occupation;
}

var firstName = 'Sophia';
var lastName = 'Lundgren';
var age = 25;
var occupation = 'Software Engineer';

fn({
    firstName: firstName,
    lastName: lastName,
    age: age,
    occupation: occupation
});
```

Whew! That's a lot of code. Here's what destructuring in combination with object shorthand does for us.

```
function fn({ firstName, lastName, age, occupation }) {
    // We already have all the variables we need!
}

const firstName = 'Sophia';
const lastName = 'Lundgren';
const age = 25;
const occupation = 'Software Engineer';

fn({ firstName, lastName, age, occupation });
```

That looks a lot better.

What this allows us to do is pass in variables in any order. In line 10 above, we could pass the variables in any order. We're technically passing in an object so the order of assignment doesn't matter.

This results in a much easier way to pass several arguments into a function. Less code, faster to write, easier to read.

Destructuring & Arrow Functions

Of course, destructuring is also allowed in arrow functions. There are some additional rules to follow, however.

This line is a function that will take in an object and return the `property` property on that object.

```
const fn = obj => obj.property;
```

We can write the same thing using object destructuring.

```
const fn = ({ property }) => property;
```

Here, we're stripping off `property` from whatever object is passed in to the function and making it available to use. We have to wrap the destructured item in parentheses. If we don't, the engine will think we're trying to create an object and we'll get a syntax error.

At this moment, array and object destructuring may seem more trouble than it's worth. I advise you to stick with and use it when possible. You'll find yourself using it more and more often and you'll find that your code is better off because of it.

That's it.