Destructuring an Object

This lesson goes over destructuring an object.

WE'LL COVER THE FOLLOWING

Extract members with destructuring

Destructuring an object lets you create new variables from an existing object.

The syntax starts by using the curly bracket and the name of the destination variable. You can use many variables separated by commas. The name of the variable must be the name of the members you want to extract from the object.

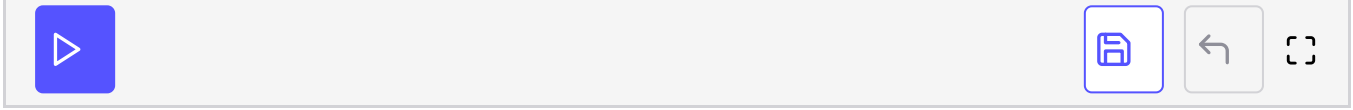
Once the member selection is made, you need to close the curly bracket, use the equal sign and write the name of the variable you want to destruct. In this scenario, like any variable, the use of `let` or `const` is used before starting.

```
const objToDesctruct1 = { destr1: 1, destr2: "2", destr3: true };
let { destr1, destr2 } = objToDesctruct1; // Destructuration
console.log(destr1);
```

Note that assigning a value to the destructured variable does not affect the original variable. At **line 4** the value is multiplied by `100` and the value does

not change at **line 5**.

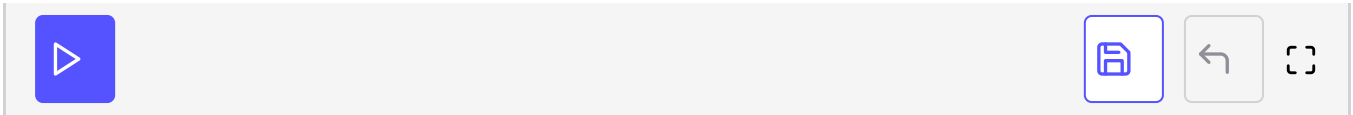
```
const objToDesctruct1 = { destr1: 1, destr2: "2", destr3: true };
let { destr1, destr2 } = objToDesctruct1;
console.log(destr1);
objToDesctruct1.destr1 *= 100;
console.log(destr1); // Remains untouched
```



Destructuring more than once

Re-deconstructuring requires avoiding defining the variable again. However, to use the destructuring, you will have to surround the call with parentheses around the whole instruction.

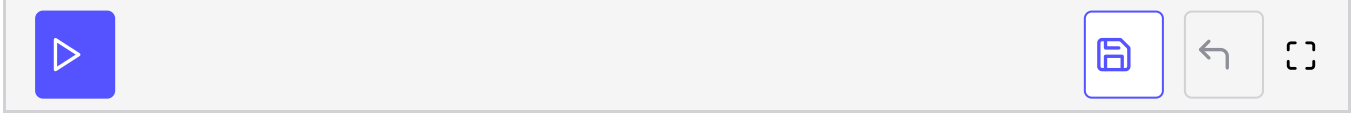
```
const objToDesctruct1 = { destr1: 1, destr2: "2", destr3: true };
let { destr1, destr2 } = objToDesctruct1;
console.log(destr1);
objToDesctruct1.destr1 *= 100;
console.log(destr1);
({ destr1, destr2 } = objToDesctruct1); // Notice parentheses
console.log(destr1);
```



Destructuring remaining variables

The second feature goes along with destructuring into variables. You can set specific variables from an object and indicate that all other members not assigned transfers into a new object. The transfer is done using the spread operator on the last variable of the left side of the assignment.

```
const objToDesctruct2 = { destr1: 1, destr2: "2", destr3: true };
let { destr1, ...remainingObjDestr1 } = objToDesctruct2;
console.log(remainingObjDestr1); // Does not have destr1
```



Renaming members while deconstructing

A third feature is that you can name the new variables with something other

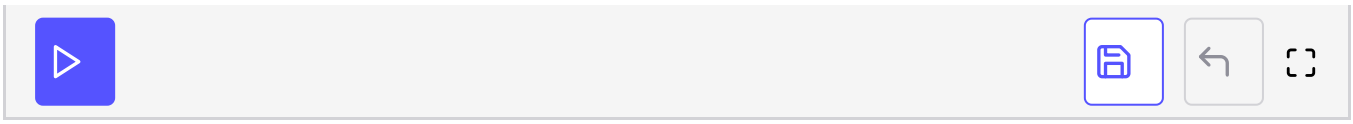
than the name provided by the member of the original object while

destructuring.

The same syntax of the first feature is used for renames, but with a small twist. The syntax changes by adding a colon after the variable and then the new name. Usually, after the colon, you can specify the name, but since you cannot change the name when destructuring, the new name is instead set.

At **line 2** of the following code, the member `destr1` is renamed to `newName`.

```
const objToDesctruct1 = { destr1: 1, destr2: "2", destr3: true };
let { destr1: newName } = objToDesctruct1;
console.log(newName);
```

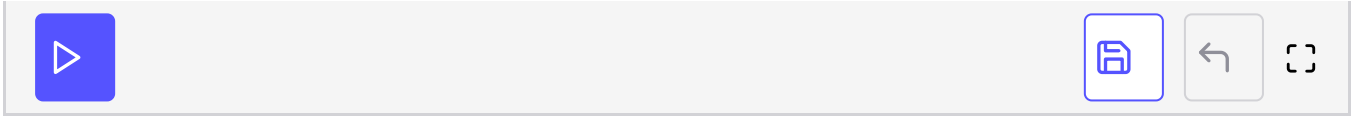


Deconstructuring and default values

Finally, when destructuring, you may stumble into a member that is optional. In this case, you want to ensure that you have a default value. To do so, after the name of the variable, use the equal sign and set the desired default value. TypeScript won't use the default value unless the value of the parameter is `undefined`.

At **line 22** the string has a default value to `"default"`.

```
const objToDesctruct3 = { destr1: 1, destr2: undefined, destr3: true };
let { destr2: newName2 = "default" } = objToDesctruct3;
console.log("Using default if undefined: ", newName2); // default
```



Destructuring an object is slightly more complex than an array because of the number of options. Nevertheless, destructuring an array is often used in languages like `React` where a portion of properties are needed. It allows for quick extraction of a specific set of members to be used directly in a function instead of always relying on the object name followed by a dot and the name of the property.