Tip 12: Update Information with Object Spread

In this tip, you'll learn how the object spread operator gives you all the advantages of Object.assign() with reduced syntax.

We'll cover the following

You saw in the previous tip how you can use `Object.assign()` to make copies of objects and how you can overwrite object values with new values from another object. It's a great tool that has a lot of value. But, wow—it's ugly.

Object spread operator

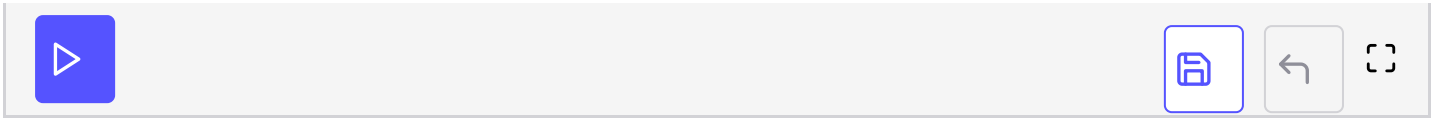
The spread operator was such a popular addition in ES6 that similar syntax is being introduced for objects. The object spread operator is not officially part of the spec, but it's so widely used that it will likely be adopted in the future. You can check out the proposal on github.

Using Proposed Syntax

How does the object spread operator work? Well, it's simple. It works like the array spread operator—the key-values are returned as if in a list. You can easily add information by placing it either before or after the spread. And like the array spread, you must spread it out into something.

```
const book = {
    title: 'Reasons and Persons',
    author: 'Derek Parfit',
};
```

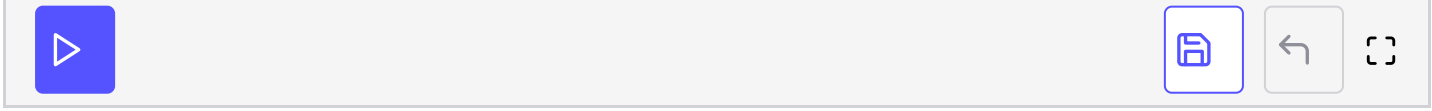
```
const update = { ...book, year: 1984 };
console.log(update);
```



But it's different from the array spread in that if you add a value with the same key, it will use whatever value is declared last.

In this way, it's like `Object.assign()` with much less typing.

```
const book = {
    title: 'Reasons and Persons',
    author: 'Derek Parfit',
};
const update = { ...book, title: 'Reasons & Persons' };
console.log(update);
```



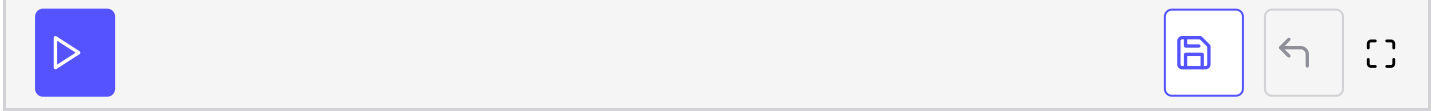
That's it! It takes the best existing features and combines them. You will not be surprised to learn that the JavaScript community embraces it enthusiastically.

Now that you have some great new syntax, try rewriting the functions from the previous tip. I'll give you the original and then the updated version. But try it yourself. It's very simple.

Updating using `Object.assign()`

Here's the way to add or update information with `Object.assign()`:

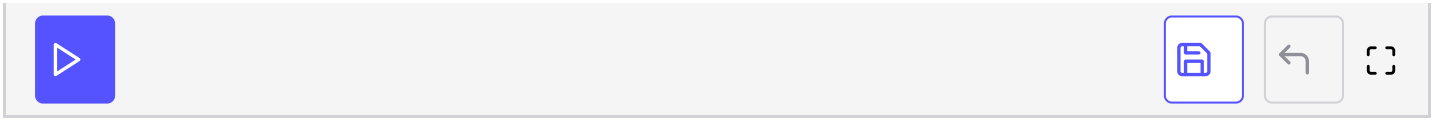
```
const defaults = {
    author: '',
    title: '',
    year: 2017,
    rating: null,
};
const book = {
    author: 'Joe Morgan',
    title: 'Simplifying JavaScript',
};
const updated = Object.assign({}, defaults, book);
console.log(updated);
```



Updating using the object spread operator

And here it is with the object spread operator:

```
const defaults = {
    author: '',
    title: '',
    year: 2017,
    rating: null,
};
const book = {
    author: 'Joe Morgan',
    title: 'ES6 Tips',
};
const bookWithDefaults = { ...defaults, ...book };
console.log(bookWithDefaults);
```



Fixing deep copying issues

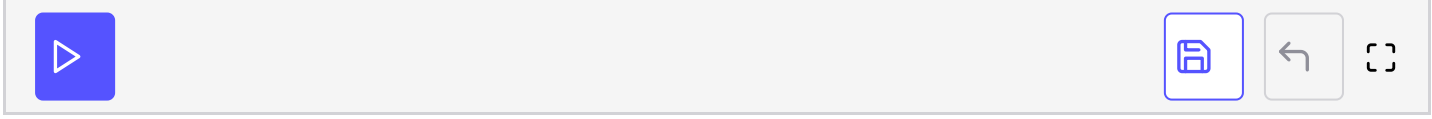
You'll have the same deep merge problems that you have with `Object.assign()`: you don't copy nested objects—you only copy a reference creating a potential problem with mutations.

Fortunately, the fix is less painful on the eyes. Here's the original.

```
const defaultEmployee = {
    name: {
        first: '',
        last: '',
    },
    years: 0,
};

const employee = Object.assign(
    {},
    defaultEmployee,
    {
        name: Object.assign({}, defaultEmployee.name),
    },
);

console.log(employee);
```



Now, before you the look at the answer, really try this out. It's straightforward, but still a little more complex. Got it? Okay. Here's the same update with the object spread operator.

```
const defaultEmployee = {
    name: {
        first: '',
        last: '',
    },
    years: 0,
};


const employee = {
    ...defaultEmployee,
    name: {
        ...defaultEmployee.name,
    },
};

employee.name.first = 'joe';
console.log("Employee name:");
console.log(employee.name.first);
console.log("Default employee name:");
console.log(defaultEmployee.name.first);
```

Advantages

The advantages are clear. The code is more readable. You're signaling your intention to create an object in a clear way. You don't have to worry about mutations because you don't need to remember to start with an empty object.

The object spread is fantastic—it's great for your code and gives you an opportunity to integrate experimental features in your code base.

What will be the output of the code below?

```
let person1 = { name: 'Joey', age: '21' };
let person2 = { name: 'Rachel', age: '22' };

let persons = [ person1, person2 ];

let setJob = function( persons, index, job ) {
    persons[ index ].job = job;
    return persons;
}

setJob( [...persons], 0, 'Artist' );
console.log( persons);
```

Retake Quiz

That’s all there is for existing collections. In the next tip, you will finally get to try out some completely new collections that should improve your code communication. First up, the `Map` object.