Object Destructuring & Spread Operators

This lesson teaches the concepts of Object Destructuring and Spread Operators and how we can use them in React.

What is Destructuring?

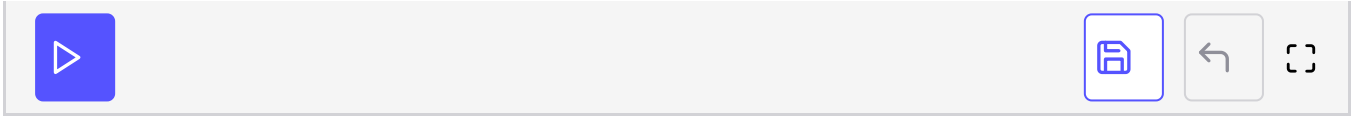
Another language feature introduced in JavaScript is called *Object Destructuring*. It's often the case that you have to access plenty of properties from your state or props in your component. Rather than assigning them to a variable one by one, you can use *destructuring* assignment in JavaScript. See the following example for better understanding:

Assume there is an object named `student` and you want to extract its properties, one way to do this is:

```
const student = {
  ID: '21',
  name: 'Jhon',
  GPA: '3.0',
};

const id = student.ID;
const name = student.name;
const GPA = student.GPA;

console.log(id);
console.log(name);
console.log(GPA);
```



But as you can see, there's a lot of repetitive code in this example. So, to make things simpler, we can rewrite this code as:

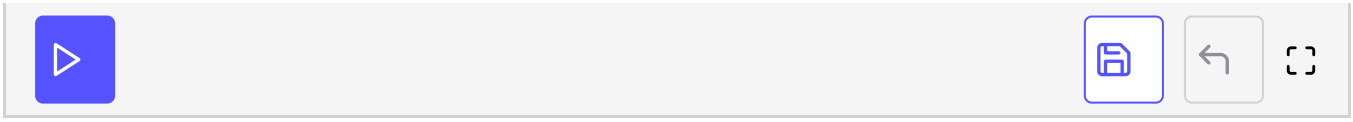
```
const student = {
  ID: '21',
  name: 'Jhon',
  GPA: '3.0',
};
```

```
const {ID, name, GPA} = student;
```

You can list the properties you want inside the curly brackets and even rename them using aliases like this:

```
const student = {
  ID: '21',
  name: 'Jhon',
  GPA: '3.0',
};

const {name:n} = student;
console.log(n);
```



Another example of destructuring could be:

```
// no destructuring
const users = this.state.users;
const counter = this.state.counter;

// destructuring
const { users, counter } = this.state;
```

That's especially beneficial for functional stateless components because they always receive the props object in their function signature. Often you will not use the props but its content, so you can destructure the content already in the function signature.

```
// no destructuring
function Greeting(props) {
  return <h1>{props.greeting}</h1>;
}

// destructuring
function Greeting({ greeting }) {
  return <h1>{greeting}</h1>;
}
```

The destructuring works for JavaScript arrays too. Another great feature is the rest destructuring. It is often used for splitting out a part of an object, but keeping the remaining properties in another object.

```
// rest destructuring
```

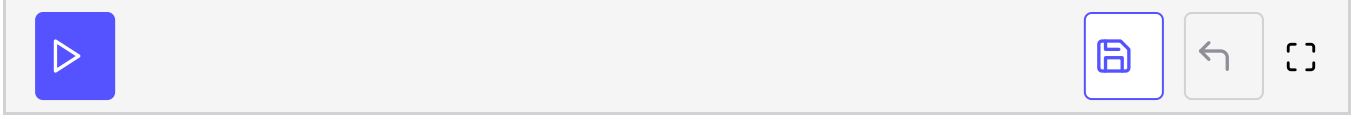
```
// rest destructuring
const { users, ...rest } = this.state;
```

Afterward, the users can be used to be rendered, for instance in a React component, whereas the remaining state is used somewhere else. That's where the JavaScript spread operator comes into play to forward the rest object to the next component. In the next section, you will see this operator in action.

Spread Operator

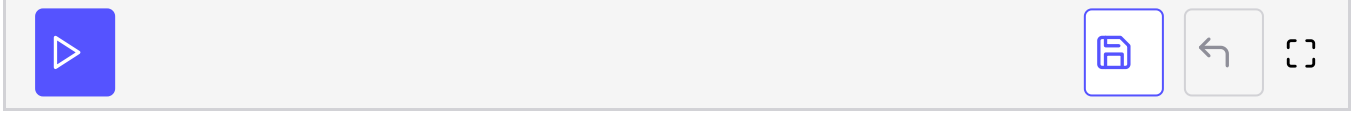
Another JavaScript feature that we use quite a lot in React is the Spread Operator. Spread Operator literally spreads the contents of an array into its elements which makes operations like concatenation etc. easier. Let's say we want to concatenate two arrays, we either do this by using `concat` function, like this:

```
a = [1,2,3];
b = [4,5,6];
c = a.concat(b);
console.log("c: " + c);
```



Or we can do the same thing using Spread Operator:

```
a = [1,2,3];
b = [4,5,6];
c = [...a, ...b]; //spread operator
console.log("c: " + c);
```



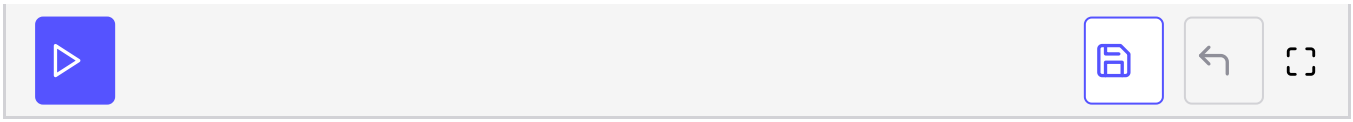
Why is Spread Operator so handy in React?

- You can easily add some elements in the middle of the two arrays: `[...a, 'something', ...b];`
- It's simpler to use and also gives you a visual idea of what your array looks like.
- You can clone arrays using this operator: `clone = [...a];`

- In React, you can combine two objects using Spread Operator and add extra properties to that object too. Let's say we have two objects `a` and `b`, we can combine these two objects and create a new one with all the properties present in both objects. See the example below:

```
const person = { name: "Jhon"};
const student = { ID: "21", GPA: "3.0"};

const new_object = { ...person, ...student, semester: '3'};
console.log(new_object);
```



In addition to this, we can also clone objects in React using Spread Operator.