Lists and Keys

(https://github.com/learn-co-curriculum/react-hooks-lists-and-keys) (https://github.com/learn-co-curriculum/react-hooks-lists-and-keys/issues/new)

Learning Goals

- Transform arrays of data into arrays of JSX elements
- Understand the need for the `key` prop and when to use it

Working with Arrays

Let's suppose we had an array of our favorite CSS colors, and we wanted to display them as separate `li` elements in an ordered list. We could do something like this:

```
function ColorList() {
  const colors = [
    "firebrick",
    "rebeccapurple",
    "salmon",
    "darkslategray",
    "hotpink",
  ];

  return (
    <div>
      <h1>Top 5 CSS Colors</h1>
      <ol>
        <li style={{ color: colors[0] }}>{colors[0]}</li>
        <li style={{ color: colors[1] }}>{colors[1]}</li>
        <li style={{ color: colors[2] }}>{colors[2]}</li>
```

```
          <li style={{ color: colors[3] }}>{colors[3]}</li>
          <li style={{ color: colors[4] }}>{colors[4]}</li>
        </ol>
      </div>
    );
  }
```

Try running this code in the browser to confirm it works — we've got a nice looking list going!

But as you can already tell, our code isn't very DRY, and this approach breaks down as soon as we add another color. How can we create these elements *dynamically* instead?

Luckily for us, JSX is still *just JavaScript*, so our usual techniques for working with arrays in regular JavaScript still apply!

Our goal is to take an array of *strings* and transform that into a new array of *JSX elements*. What array method would be appropriate here? Think on it, then scroll down...

...

...

...

...

...

...

...

If you came up with `.map`, awesome! That's the perfect tool for the job here — we've got an array of 5 strings and we want an array of 5 JSX elements instead.

Here's how we could use `.map`:

```
function ColorList() {
  const colors = [
    "firebrick",
    "rebeccapurple",
    "salmon",
    "darkslategray",
    "hotpink",
  ];

  const colorElements = colors.map((color) => {
    return <li style={{ color: color }}>{color}</li>;
  });
  return (
    <div>
      <h1>Top 5 CSS Colors</h1>
      <ol>
        {/* display the array of <li> elements here! */}
        {colorElements}
      </ol>
    </div>
  );
}
```

Our code still works, but now, we can create the `<li>` elements *dynamically* based on the `colors` array. Adding a new string to that array means that our list grows without us having to create a new `<li>` by hand.

The key Prop

If you run our updated demo code in the browser and open up the console, you'll see a big warning message, like this:

```
Warning: Each child in a list should have a unique "key" prop.
  Check the render method of `ColorList`.
```

To fix this error, we must give each `<li>` element a special `key` prop, like so:

```
const colorElements = colors.map((color) => {
  return (
    <li key={color} style={{ color: color }}>
      {color}
    </li>
  );
});
```

This special `key` prop allows React internally to keep track of each element in the array of JSX, so that in case any of the elements are added, updated or deleted, React can optimize performance and keep track internally of those changes.

The key should be a **unique value** associated with each element from the array. Since each element in our array of colors is unique, we can just use each color for the `key` prop.

Any time you are creating an *array* of JSX elements, you *must* use the `key` prop.

Let's see a couple more examples of how this `key` prop can be used.

With Objects

If you have an array of objects, it's often best to use the object's `id` as the key:

```
const users = [
  { id: 1, firstName: "Duane", lastName: "Watson" },
  { id: 2, firstName: "Duane", lastName: "Johnson" },
];

const userHeadings = users.map((user) => {
  return <h1 key={user.id}>{user.firstName}</h1>;
});
```

With Non-Unique Arrays

If you have an array of elements that aren't unique, and you can't use the `id`, you might be tempted to use the index position instead:

```
const fib = [0, 1, 1, 2, 3, 5];

const fibList = fib.map((number, index) => {
  return <div key={index}>{number}</div>;
});
```

While this will make the error message go away, it's not necessarily the best approach — in fact, the React docs recommend **only using the index as a last resort (https://reactjs.org/docs/lists-and-keys.html#keys)**. If you're interested in some alternatives to using the index, **this article (https://medium.com/@robinpokorny/index-as-a-key-is-an-anti-pattern-e0349aece318)** goes over a few suggestions.

With Nested Components

Back to our `ColorList` example. Let's imagine we want to create a separate component to display each color. In the example below, we'd need to use the `key` prop on the `<ColorItem>` components, **not** the `<li>`:

```
// ColorItem component
function ColorItem(props) {
  return <li style={{ color: props.color }}>{props.color}</li>;
}

// ColorList component
function ColorList() {
  const colors = [
    "firebrick",
    "rebeccapurple",
    "salmon",
    "darkslategray",
    "hotpink",
```

```
  ];

  const colorElements = colors.map((color) => {
    return <ColorItem key={color} color={color} />;
  });
  // etc
}
```

One other thing to note about this example: the `key` prop **won't show up with the rest of the props in our functions**. If you add a `console.log(props)` in the `ColorItem` component, you won't see the `key` prop. This is because the `key` prop is meant for use internally by React.

Conclusion

In this lesson, we learned how to make our code more dynamic by using `.map` to transform an array of data into an array of JSX elements instead.

We also learned that, any time you are creating an *array* of JSX elements, you *must* use the `key` prop on each element of that array. The key needs to be some **unique value** for each element of the array. Array indexes should **not** be used as keys.

Resources

- **Lists and Keys (https://reactjs.org/docs/lists-and-keys.html)**
- **Why Using Index as a Key is Probably a Bad Idea (https://medium.com/@vraa/why-using-an-index-as-key-in-react-is-probably-a-bad-idea-7543de68b17c)**