Lists and Keys

First, let's review how you transform lists in JavaScript.

Given the code below, we use the `map()` function to take an array of `numbers` and double their values. We assign the new array returned by `map()` to the variable `doubled` and log it:

```
const numbers = [1, 2, 3, 4, 5];
const doubled = numbers.map((number) => number * 2);
console.log(doubled);
```

This code logs `[2, 4, 6, 8, 10]` to the console.

In React, transforming arrays into lists of elements is nearly identical.

Rendering Multiple Components

You can build collections of elements and include them in JSX using curly braces `{}`.

Below, we loop through the `numbers` array using the JavaScript `map()` function. We return a `<li>` element for each item. Finally, we assign the resulting array of elements to `listItems`:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

We include the entire `listItems` array inside a `<ul>` element, and render it to the DOM:

```
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

Try it on CodePen

This code displays a bullet list of numbers between 1 and 5.

Basic List Component

Usually you would render lists inside a component.

We can refactor the previous example into a component that accepts an array of `numbers` and outputs a list of elements.

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
```

```
    <li>{number}</li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

When you run this code, you'll be given a warning that a key should be provided for list items. A "key" is a special string attribute you need to include when creating lists of elements. We'll discuss why it's important in the next section.

Let's assign a `key` to our list items inside `numbers.map()` and fix the missing key issue.

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>
      {number}
    </li>
  );
  return (
    <ul>{listItems}</ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Try it on CodePen

Keys

Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity:

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
  </li>
);
```

The best way to pick a key is to use a string that uniquely identifies a list item among its siblings. Most often you would use IDs from your data as keys:

```
const todoItems = todos.map((todo) =>
  <li key={todo.id}>
    {todo.text}
  </li>
);
```

When you don't have stable IDs for rendered items, you may use the item index as a key as a last resort:

```
const todoItems = todos.map((todo, index) =>
  // Only do this if items have no stable IDs
  <li key={index}>
    {todo.text}
  </li>
);
```

We don't recommend using indexes for keys if the order of items may change. This can negatively impact performance and may cause issues with component state. Check out Robin Pokorny's article for an in-depth explanation on the negative impacts of using an index as a key. If you choose not to assign an explicit key to list items then React will default to using indexes as keys.

Here is an in-depth explanation about why keys are necessary if you're interested in learning more.

Extracting Components with Keys

Keys only make sense in the context of the surrounding array.

For example, if you extract a `ListItem` component, you should keep the key on the `<ListItem />` elements in the array rather than on the `<li>` element in the `ListItem` itself.

Example: Incorrect Key Usage

```
function ListItem(props) {
  const value = props.value;
  return (
    // Wrong! There is no need to specify the key here:
    <li key={value.toString()}>
      {value}
    </li>
  );
}

function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    // Wrong! The key should have been specified here:
    <ListItem value={number} />
  );
  return (
    <ul>
      {listItems}
    </ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Example: Correct Key Usage

```
function ListItem(props) {
  // Correct! There is no need to specify the key here:
```

```
  return <li>{props.value}</li>;
}

function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    // Correct! Key should be specified inside the array.
    <ListItem key={number.toString()} value={number} />
  );
  return (
    <ul>
      {listItems}
    </ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Try it on CodePen

A good rule of thumb is that elements inside the `map()` call need keys.

Keys Must Only Be Unique Among Siblings

Keys used within arrays should be unique among their siblings. However they don't need to be globally unique. We can use the same keys when we produce two different arrays:

```
function Blog(props) {
  const sidebar = (
    <ul>
      {props.posts.map((post) =>
        <li key={post.id}>
          {post.title}
        </li>
      )}
    </ul>
  );
  const content = props.posts.map((post) =>
    <div key={post.id}>
      <h3>{post.title}</h3>
      <p>{post.content}</p>
    </div>
  );
  return (
    <div>
      {sidebar}
      <hr />
      {content}
    </div>
  );
}

const posts = [
  {id: 1, title: 'Hello World', content: 'Welcome to learning React!'},
  {id: 2, title: 'Installation', content: 'You can install React from npm.'}
];
ReactDOM.render(
  <Blog posts={posts} />,
  document.getElementById('root')
);
```

Try it on CodePen

Keys serve as a hint to React but they don't get passed to your components. If you need the same value in your component, pass it explicitly as a prop with a different name:

```
const content = posts.map((post) =>
  <Post
    key={post.id}
    id={post.id}
    title={post.title} />
);
```

With the example above, the `Post` component can read `props.id`, but not `props.key`.

Embedding map() in JSX

In the examples above we declared a separate `listItems` variable and included it in JSX:

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <ListItem key={number.toString()}
              value={number} />
  );
  return (
    <ul>
      {listItems}
    </ul>
  );
}
```

JSX allows embedding any expression in curly braces so we could inline the `map()` result:

```
function NumberList(props) {
  const numbers = props.numbers;
  return (
    <ul>
      {numbers.map((number) =>
        <ListItem key={number.toString()}
                  value={number} />
      )}
    </ul>
  );
}
```

Try it on CodePen

Sometimes this results in clearer code, but this style can also be abused. Like in JavaScript, it is up to you to decide whether it is worth extracting a variable for readability. Keep in mind that if the `map()` body is too nested, it might be a good time to extract a component.

Edit this page

Previous article Next article

Conditional Rendering Forms