**Additional resources**

Here is a list of additional resources for Rendering Lists in React:

* [Map()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map) allows you to create new arrays populated with the results of calling a transformation function on every element.
* [Rendering lists on official React docs website](https://beta.reactjs.org/learn/rendering-lists#rendering-data-from-arrays) dives deeper into how to display multiple similar components from a collection of data, providing examples of both filtering and transformations.
* [React keys on official docs](https://beta.reactjs.org/learn/rendering-lists#where-to-get-your-key) offers a comprehensive set of memotecnic rules to reinforce how to use keys properly.

Here is a list of additional resources for Module 1, Lesson 3 (Forms in React):

* [Forms from the official React docs](https://reactjs.org/docs/forms.html) illustrate some examples of how React deals with certain form fields compared to traditional HTML tags, like the text area, select and file input tags. It also showcases how to handle multiple inputs by leveraging **event.target.name** and the implications of using null as a value in a controlled input.
* [Formik](https://formik.org/) is the most popular open source form library for React. It saves you lots of time when building forms and offers a declarative, intuitive and adoptable paradigm.
* [Yup](https://github.com/jquense/yup) is an open-source library that integrates perfectly with Formik. It allows you to set all your form validation rules declaratively.
* [React-hook-form](https://github.com/react-hook-form/react-hook-form) is another popular library to easily manage your form state and validation rules.

Here is a list of additional resources for React Context.

* [React.memo from the official React docs](https://reactjs.org/docs/react-api.html#reactmemo), an API that can be used in conjunction with Context Providers to prevent unnecessary re-renders in top-level components in the tree.
* [useMemo from the official React docs](https://reactjs.org/docs/hooks-reference.html#usememo), a hook to guarantee referential equality on objects across rendering passes.

Below is a list of additional resources as you continue to explore React hooks and custom hooks.

In particular, to complement your learning in the “Getting started with hooks” lesson, you can work through the following:

* The article on [destructuring](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Destructuring_assignment) assignment describes how the destructuring assignment, which allows you to get values out of the array that gets returned when the **useState** hook is invoked, works in more detail.
* [The read props inside the child component](https://beta.reactjs.org/learn/passing-props-to-a-component#step-2-read-props-inside-the-child-component) link on the Beta version of React docs discusses how to use destructuring assignment to get values out of the props object.
* The [useState reference on official React docs website](https://beta.reactjs.org/apis/react/useState#usestate) helps you understand how to work with this hook and some of the caveats involved.
* The [useEffect reference on official React docs website](https://beta.reactjs.org/apis/react/useEffect#useeffect) helps you understand the syntax of this hook and goes into some depth to explain how to use and troubleshoot the **useEffect** hook.

Below is a list of additional resources as you continue to explore React hooks and custom hooks.

In particular, to complement your learning on the 'Rules of hooks and fetching data with hooks' lesson, you can work through the following:

* The [Rules of Hooks reading on Reactjs.org](https://reactjs.org/docs/hooks-rules.html) website gives you an overview of how to work with the hooks as recommended by the React Core team at Meta.
* The [Fetching data with Effects](https://beta.reactjs.org/apis/react/useEffect#fetching-data-with-effects) article on React docs discusses fetching data using a few different approaches, including using async / await syntax.
* [How to use promises](https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Promises) is a resource that describes the "behind-the-scenes" of how data fetching works in greater depth.
* [async function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function) is a resource on MDN that discusses the use of the async and await keywords as a more recent way to handle API requests in JavaScript.

Below is a list of additional readings as you continue to explore 'React hooks and custom hooks'.

In particular, to complement your learning in the 'Advanced hooks' lesson, you can work through the following:

* The [useReducer hook reference](https://react.dev/reference/react/useReducer) in the React docs discusses the basics of **useReducer**, along with specifying initial state and lazy initialization.
* The React docs also has a reference on [using the useRef hook](https://react.dev/reference/react/useRef) which is a great example of various options that are available when working with the **useRef** hook.
* The [Reusing Logic with Custom Hooks](https://beta.reactjs.org/learn/reusing-logic-with-custom-hooks) reference in the React docs discusses the dynamics of custom hooks and provides a few practical examples to complement the theory behind them.

Here is a list of additional resources for JSX Deep Dive:

* [Chakra-UI](https://chakra-ui.com/) is an open-source component library that embraces all the concepts explained during this lesson, being a nice option if you would like to start your project with a set of atomic components that have been carefully designed with flexibility in mind, so that they can be customized as per your theme requirements.
* [Compound components with hooks](https://kentcdodds.com/blog/compound-components-with-react-hooks) is an article that illustrates how a combination of component composition, context and hooks can lead to a clean and concise component design.
* [React Fragments](https://reactjs.org/docs/fragments.html) from the official React docs illustrates how to group a list of React children without adding extra nodes to the DOM.

Here is a list of additional resources as you continue to explore Reusing Behavior:

* [Downshift](https://github.com/downshift-js/downshift) is a popular open-source library that implements an autocomplete, combo box or select experience using the **render** prop pattern.
* [Render props](https://reactjs.org/docs/render-props.html) from the official React docs.
* [Higher Order Components](https://reactjs.org/docs/higher-order-components.html) from the official React docs.
* [Forwarding Refs](https://reactjs.org/docs/forwarding-refs.html#forwarding-refs-in-higher-order-components) from the official React docs showcases in detail how to forward refs in higher-order components, so that they are passed through properly.

Here is a list of additional resources as you continue to explore Integration tests with React Testing Library:

* [React testing library](https://testing-library.com/docs/react-testing-library/intro/) official documentation.
* [Jest](https://jestjs.io/docs/getting-started) official documentation.
* [Continuous delivery](https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment) is a great article from Atlassian that illustrates the differences between Continuous integration, delivery and deployment, and how they all tie together.
* [Practical test pyramid](https://martinfowler.com/articles/practical-test-pyramid.html) is an extensive article that dives into the importance of test automation, showing you which kind of tests you should be looking for in the different levels of the pyramid and providing practical examples on how those can be implemented.