the shortcut

React Tutorial
Lesson 2

Today's lesson will cover

- React Props
- React State using State Hook
- Callback handlers
- Lifting state
- React controlled components
- Break Props Handling
- React Side-Effects
- React Custom Hooks

Intro to React

React Props

- Using so called props, we can pass variables as information from one component to another component
- Whether you declare a component as a function or a class, it must never modify its own props
- React is pretty flexible but it has a single strict rule: All React components must act like pure functions with respect to their props.

React State using the State Hook

- Hooks are a new addition in React 16.8. They let you use state and other React features without writing a class.
- React Props are used to pass information down the component tree; **React state** is used to make applications interactive
- First, there is a utility function called useState that we take from React for managing state. The useState function is called a hook. There is more than one **React hook** -- related to state management but also other things in React
- **What is a Hook?** A Hook is a special function that lets you “hook into” React features. For example, `useState` is a Hook that lets you add React state to function components. React's `useState` hook takes an *initial state* as an argument
- **When would I use a Hook?** If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. **Now you can use a Hook inside the existing function component.**

Callback handlers in JSX

- There is no way to pass information as JavaScript data types up the component tree, since props are naturally only passed downwards. However, we can introduce a **callback handle**r as a function: A callback function gets introduced (A), is used elsewhere (B), but "calls back" to the place it was introduced (C).
- Consider the concept of the callback handler: We pass a function from one component (App) to another component (Search); we call it in the second component (Search); but have the actual implementation of the function call in the first component (App). This way, we can communicate up the component tree. A handler function used in one component becomes a callback handler, which is passed down to components via React props. React props are always passed down as information to the component tree, and callback handlers passed as functions in props can be used to communicate up the component hierarchy.

Lifting State

- Always manage the state at a component where every component that's interested in it is one that either manages the state (using information directly from state) or a component below the managing component (using information from props). If a component below needs to update the state, pass a callback handler down to it. If a component needs to use the state (e.g. displaying it), pass it down as props.

Controlled Components

- **Controlled components** are not necessary React components, but HTML elements
- In HTML, form elements such as `<input>`, `<textarea>`, and `<select>` typically maintain their own state and update it based on user input. In React, mutable state is typically kept in the state property of components, and only updated with `setState()`.
- We can combine the two by making the React state be the “single source of truth”. Then the React component that renders a form also controls what happens in that form on subsequent user input. An input form element whose value is controlled by React in this way is called a “controlled component”.

Break Props Handling

- Props are passed from parent to child down the component tree. Since we use props to transport information from component to component frequently, and sometimes via other components which are in between, it is useful to know a few tricks to make passing props more convenient.

- React props are a JavaScript object, else we couldn't access `props.list` or `props.onSearch` in React components. Since props is an object which just passes information from one component to another component, we can apply a couple JavaScript tricks to it. For example, accessing an object's properties with modern JavaScript object destructuring

React Side Effects

- A "side effect" is anything that affects something outside the scope of the function being executed.
- The *Effect Hook* lets you perform side effects in function components: `useEffect` will run after the render is committed to the screen. Think of effects as an escape hatch from React's purely functional world into the imperative world.
- There are two common kinds of side effects in React components: those that don't require cleanup, and those that do
- Data fetching, setting up a subscription, and manually changing the DOM in React components are all examples of side effects. Whether or not you're used to calling these operations "side effects" (or just "effects"), you've likely performed them in your components before

React Custom Hooks

- Two most popular built-in hooks in React: useState and useEffect
    a. `useState` is used to make your application interactive
    b. `useEffect` is used to opt into the lifecycle of your components.
- **React custom Hooks** is building a hook yourself: Building your own Hooks lets you extract component logic into reusable functions.
- **A custom Hook is a JavaScript function whose name starts with "`use`" and that may call other Hooks.**
- **Custom Hooks are a convention that naturally follows from the design of Hooks, rather than a React feature.**
- Hooks are the bread and butter in React function components so it's really important to understand them

the shortcut

Self Study Assignments

To Dos

- Start working on React or Fullstack JavaScript project, OR complete FCC Frontend Library Certification : Bootstrap, React, Redux, React & Redux, projects; APIs and Microservices Certification : https://www.freecodecamp.org/learn
- Refer to Reactjs official documentation as the best place to learn underlying main concepts of React with examples: https://reactjs.org/docs/hello-world.html