Ternary Operator

The ternary operator is a simplified conditional operator like if / else.

Syntax: condition ? <expression if true> : <expression if false>

Let's revisit the App( ) function above to see how to pass data with props.

First, we need to define a prop on the Welcome Component and assign a value to it:

import Welcome from './Welcome';

function App() {

return (

<div className="App">

<Welcome name="John"/>

<Welcome name="Mary"/>

<Welcome name="Alex"/>

</div>

);

}

Props are custom values and they also make components more dynamic. Since the Welcome component is the child here, we need to define props on its parent (App), so we can pass the values and get the result simply by accessing the prop "name":

function Welcome(props) {

return <h1>Hello, {props.name}</h1>;

}

**Running useEffect**

Using useEffect hook is like having both componentDidMount and componentDidUpdate in one single method, since useEffect runs on every render. It accepts two arguments:

* (mandatory) A function to run on every render
* (optional) An array of state variables to watch for changes. useEffect will be skipped if none of the variables are updated.

Rewriting the above class into function component would look like this:

const Example = props => {

const [name, setName] = useState('Nathan');

useEffect(() => {

console.log(`Hello I'm ${name}`);

});

return (

<div>

<p>{`Hello I'm ${name}`}</p>

<button

onClick={() => {

setName('Gary')

}}>

Change me

</button>

</div>

)

}

The function component above will run the function inside of useEffect function on each render. Now this isn’t optimal because the state won’t be updated after the first click. This is where useEffect second argument come into play.

useEffect(() => {

console.log(`Hello I'm ${name} and I'm a ${role}`);

},

[name]);

The second argument of useEffect function is referred to as the “dependency array”. When the variable included inside the array didn’t change, the function passed as the first argument won’t be executed.

**The componentWillUnmount effect**

If you have code that needs to run when the component will be removed from the DOM tree, you need to specify a componentWillUnmount effect by writing a return statement into the first argument function. Here is an example:

useEffect(() => {

console.log(`useEffect function`);

return () => { console.log("componentWillUnmount effect"); }

}, [name] );

### Running useEffect only once

To run useEffect hook only once like componentDidMount function, you can pass an empty array into the second argument:

useEffect(

() => {

console.log(`useEffect function`);

},

[] );

The empty array indicates that the effect doesn’t have any dependencies to watch for change, and without a trigger, it won’t be run after the component is mounted.