Component Declarations

In this lesson, we will discuss the three type of class components that we are using in our application i.e. Functional Stateless Components, ES6 Class components, and React.createClass

Now we have four ES6 class components, but our application can still be improved using functional stateless components as an alternative for ES6 class components. Before you refactor your components, let's introduce the different types.

- **Functional Stateless Components** are functions that take input and return an output. The inputs are the props, and the output is a component instance in plain JSX. So far, it is quite similar to an ES6 class component. However, functional stateless components are functions (functional) and they have no local state (stateless). You cannot access or update the state with `this.state` or `this.setState()` because there is no `this` object. Additionally, they have no lifecycle methods except for the `render()` method which will be applied implicitly in functional stateless components. You didn't learn about lifecycle methods yet, but you already used two: `constructor()` and `render()`. The constructor runs only once in the lifetime of a component, whereas the `render()` class method runs once at the beginning and every time the component updates. Keep in mind that functional stateless components have no lifecycle methods when we arrive at lifecycle methods chapter later.
- **ES6 Class Components** extend from the React component. The `extend` hooks all the lifecycle methods, available in the React component API, to the component. This is how we were able to use the `render()` class method. You can also store and manipulate state in ES6 class components using `this.state` and `this.setState()`.
- **React.createClass** was used in older versions of React and is still used in JavaScript ES5 React applications. But Facebook declared it as deprecated

in favor of JavaScript ES6. They even added a deprecation warning in version 15.5, so we will not use it in the course.

When deciding when to use functional stateless components over ES6 class components, a good rule of thumb is to use functional stateless components when you don't need local state or component lifecycle methods. Usually, we implement components as functional stateless components, but once access to the state or lifecycle methods is required, we have to refactor it to an ES6 class component. We started the other way around in our application for the sake of learning.

Returning to the application, we see the App component uses local state, so it has to stay as an ES6 class component. The other three ES6 class components are stateless, so they don't need access to `this.state` or `this.setState()`, and they have no lifecycle methods. We're going to refactor the Search component to a stateless functional component. The Table and Button component refactoring will become your exercise.

```
function Search(props) {
  const { value, onChange, children } = props;
  return (
    <form>
      {children} <input
        type="text"
        value={value}
        onChange={onChange}
      />
    </form>
  );
}
```

The `props` are accessible in the function signature, and the return value is JSX; but we can do more with the code in a functional stateless component using ES6 destructuring. The best practice is to use it in the function signature to destructure the `props`:

```
function Search({ value, onChange, children }) {
  return (
    <form>
      {children} <input
        type="text"
        value={value}
      />
    </form>

  );
}
```

Remember that ES6 arrow functions allow you to keep your functions concise and let you remove the block body of the function. In a concise body, an implicit return is attached, letting us remove the `return` statement. Since the functional stateless component is a function, it can be made more concise as well:

```
const Search = ({ value, onChange, children }) =>
  <form>
    {children} <input
      type="text"
      value={value}
      onChange={onChange}
    />
  </form>
```

The last step is especially useful to enforce props as input and JSX as output. Still, you could *do something* in between by using a block body in your ES6 arrow function:

```
const Search = ({ value, onChange, children }) => {

  // do something

  return (
    <form>
      {children} <input
        type="text"
        value={value}
        onChange={onChange}
      />
    </form>
  );
}
```

Now you have one lightweight functional stateless component. When we need access to the local component state or lifecycle methods, we can refactor it to an ES6 class component. When using block bodies, programmers tend to make their functions more complex, but leaving the block body out lets you focus on the input and output. JavaScript ES6 in React components makes components more readable and elegant.

Exercises:

- Refactor the Table and Button component to stateless functional components

Further Reading

- Read about ES6 class components and functional stateless components