Source: https://reactjs.org/tutorial/tutorial.html

General Notes

`class` vs. `className`

- `class` is a HTML attribute
- `.className` allows to get/set that attribute in React code

To trigger a function when something happens to an element (e.g. user clicks on it), don't forget to pass a function!

- RIGHT: `onClick={() => doSomething()}`
- WRONG: `onClick={doSomething()}`

→ In the wrong case, the function is called everytime the component with that element re-renders (note: the element might be the component itself)

Naming convention:

- If it's a prop that's supposed to something when an event happens, name it `on[Event]` (e.g. `onClick`)
- If it's a method that handles an event, name it `handle[Event]` (e.g. `handleClick`)

React.Component

Passed attributes are stored in `this.props.`*xxx*
The component's own attributes are stored in `this.state.`*xxx*

To reference a class that extends React.Component use *<XXX />*, e.g. `<Square />`

Whenever `this.setState(...)` is used, the component and all its children are re-rendered

When to lift state up

a) We want to collect data from multiple children
b) We want children to communicate with each other

→ The parent passes the uplifted state variable as a prop to its children

How to lift state up

1. Put the state variable in the parent's constructor
2. In the parent, pass an additional prop for the relevant event listener, that returns a function of the parent

```
renderSquare(i) {
  return (
  <Square
    value={this.state.squares[i]}
    onClick={() => this.handleClick(i)}
  />;
  )
}
```

3. In the child, let the event listener return a function, that calls the passed prop

```
class Square extends React.Component {
  render() {
    return (
      <button
        className="square"
        onClick={() => this.props.onClick()}
      >
        {this.props.value}
      </button>
    );
  }
}
```

A component that is fully controlled by its parent is called a **controlled component**

If a React component only contains a render() method, then convert it to a function component!

```
function Square(props) {
  return (
    <button className="square" onClick={props.onClick}>
      {props.value}
    </button>
  );
}
```

→ It's no longer a class
→ The `render` function is gone
→ We don't have to return a function in event handlers
→ If we pass a function as a prop we call it as an attribute (idk why???)

→ Instead of `this.props` we write `props`

Immutability

2 possible ways to change data

a) Mutate the data

```
var player = {score: 1, name: 'Jeff'};
player.score = 2;
```

b) Replace the data with a new copy

```
var player = {score: 1, name: 'Jeff'};

var newPlayer = Object.assign({}, player, {score: 2});
// Now player is unchanged, but newPlayer is {score: 2, name: 'Jeff'}

// Or if you are using object spread syntax proposal, you can write:
// var newPlayer = {...player, score: 2};
```

Why you should always replace instead of mutating

- It simplifies historization (e.g. to detect changes, or undo)
- Change detection simplifies controlled re-rendering