Redux

- React = the data contained in the application; provides the views to display that state
- Redux = the views contained in the application; served to construct the application state
- react-redux is the library that combines React and Redux (glue between these)
  - import { connect } from 'react-redux';

- **Container** (aka "smart component") is a react component that has a direct connection to the state managed by redux
- We want the most parent component that knows about a piece of state to be a container
- Components (aka "dumb components") don't care about when state changes

- **Reducers** - a function that returns a piece of the application's state
  - Application can have many different pieces of state, so it can have many different reducers
  - Reducers produce the value of the state
  - rootReducer combines all reducers, where:
    - {key is piece of state: [value is output of reducers, like a list of objects]}

To call a separate function within JSX: {this.function()}
Key in map just needs to be a unique value

function **mapStateToProps**(state)
- This function is the glue between React and Redux
- The purpose of this is to take the application state as an argument
- Whatever is returned will show up as **props** inside of the Container class
- Whatever is returned (usually an object) will be set equal to this.props
- return {propertyName: state.propertyName (this will be the value returned, like an array of objects)};
- Whenever state changes, the container will instantly **re-render**

export default connect(mapStateToProps)(ContainerName);
- connect takes a function and a component and produces a Container

mapDispatchToProps() allows us to pass through our action creator as a prop
bindActionCreators() function, we can wrap the action creator in our store's dispatch function, so that the action is properly dispatched to the store

Actions and Action Creators
- **Actions** are for changing state
- An action creator is a function that calls an action and returns an object
  - The action has a type hat describes the type of action that was just triggered. The action can also have some data that further describes the data
- The action object is then sent to ALL the reducers in an application.
  - **Switch statement** is in a reducer which will go to a different line depending on the type of action. If the type matches, it will return action.whatever. Reducer doesn't have to react to every action, in which case it will just return the current state.
- The reducer can change to return a different piece of state depending on what the action is. The new state is then sent to the application state, then causes container to re-render.
  - Once all reducers have processed the action (new state or current state), the news assembled state goes back to all containers.
  - Then the function mapStateToProps will re-render with updated state.
- **When the user triggers a new event, this is repeated: call action creator -> returns action -> action flows to reducers -> reducers assemble new state -> new state flows to containers**