Redux cheat sheet

action types

```
const ADD_TODO = 'ADD_TODO'
const REMOVE_TODO = 'REMOVE_TODO'
const UPDATE_TODO = 'UPDATE_TODO'
```

action creators

```
const addTodo = (text) => ({
    type: ADD_TODO,
    text
})
const removeTodo = (id) => ({
    type: REMOVE_TODO,
    id
})
const updateTodo = (id, text) => ({
    type: UPDATE_TODO
    id,
    text
})
```

reducers

```
const initialState = {
    todos: []
}

function todosReducer(state = initialState,
action) {
    switch (action.type) {
        case UPDATE_TODO:
            const newState =
deepClone(state)
            const todo = newState.todos.find(
                todo => todo.id === action.id
            )
            todo.text = action.text
            return newState
    }
}

function deepClone(obj) {
    return JSON.parse(JSON.stringify(obj))
}
```

store

```
import {
    createStore,
    combineReducers
} from 'redux'
import todos from './todosReducer'
import counter from './counterReducer'

const rootReducer = combineReducers({
    todos,
    counter
})

const store = createStore(rootReducer)

export default store
```

react-redux provider

```
import React from 'react'
import { render } from 'react-dom'
import { Provider } from 'react-redux'
import { createStore } from 'redux'
import todoApp from './reducers'
```

react-redux connect

```
import { connect } from 'react-redux'

YourComponent = connect(
    mapStateToProps,
    mapDispatchToProps
```

```
import App from './components/App'

const store = createStore(todoApp)

render(
    <Provider store={store}>
        <App />
    </Provider>,
    document.getElementById('root')
)
```

redux middleware

```
const logger = store => next => action => {
    console.log('dispatching', action)
    let result = next(action)
    console.log('next state', store.getState())
    return result
}

// ----------

import {
    createStore,
    applyMiddleware
} from 'redux'

const store = createStore(
    rootReducer,
    applyMiddleware(logger)
)
```

```
)(YourComponent)

export default YourComponent
```

advanced reducers

```
// Use this as a helper function
export const createReducer = (initialState,
actionsMap) => (
    state = initialState,
    action
) => {
    const reduceFn = actionsMap[action.type]
    if (reduceFn) {
        return reduceFn(state, action)
    }
    return state
}

// In your reducer file
const actionMap = {
    [actionTypes.increment]: (state, action) => {
        return {
            ...state,
            count: state.count + 1
        }
    },
    [actionTypes.decrement]: (state, action) => {
        return {
            ...state,
            count: state.count - 1
        }
    },
    [actionTypes.setCount]: (state, action) => {
        return {
            ...state,
            count: action.count
        }
    }
}

const initialState = {
    count: 0
}
```

```
const reducer = createReducer(initialState,
actionsMap)
```