# Currying

Currying is the transformation of a function with multiple arguments into a sequence of single-argument functions. That means converting a function like this f(a, b, c, ...) into a function like this f(a)(b)(c)... . Currying is helpful when you have to frequently call a function with a fixed argument.

Normal Function :

function isGreaterThan(a, b) {

return b > a;

}

isGreaterThan(2, 5)

Currying function :

function isGreaterThan(a) {

return function(b) {

return b > a;

}

}

isGreaterThan(2)(5);

const isGreaterThan = a => b => b > a;

# object Update

1. In JavaScript we can copy object and update the object without mutable the value of principal object with object.assign() method.

Syntax: Object.assign(target, sources);

Code:

const obj = { a: 1 };

const copy = Object.assign({}, obj);

console.log(copy);

1. but now we use spread operator for update the object.

# redux with JavaScript

1. **component:** First we create a html screen where work with state.

Code:

<html lang="en">

<head>

</head>

<body>

    <div class="container">

        <p>Home</p>

        <h1 id="counter"></h1>

            <button id="inc"> increment </button>

            <button id="dec"> decrement </button>

</div>

    <script src="react.js"></script>

</body>

</html>

1. **initial the value:** For redux use we first create initial state. This state we pass to redux for use state change and UI interaction.

const initialState = {

    value: 0

}

1. **create action/reducer function:**

function counterReducer(state = initialState, action) {

  if (action.type == "increment") {

    return {

      ...state,

      value: state.value + 1,

    };

  } else if (action.type == "decrement") {

    return {

      ...state,

      value: state.value - 1,

    };

  } else {

    return state;

  }

}

1. **store create:** we use createStore() method for create store. Here we save state and action.

const store = Redux.createStore(counterReducer);

1. **dispatch method:**

inc.addEventListener("click", () => {

  store.dispatch({

    type: "increment",

  });

});

dec.addEventListener("click", () => {

store.dispatch({

    type: "decrement",

});

});

1. **Subscribe:**

store.subscribe(rendar);

1. **update UI :**

const rendar = () => {

  const state = store.getState();

  console.log(state);

  counter.innerText = state.value.toString();

};

# React-Redux

**Normal React-Redux:**

1. **install:** we should install “redux” then “react- redux”

npm i redux

npm I react-redux

1. **value initiate:**

const initialState = {

    value: 0

}

1. **action create:**

function counterReducer (state = initialState, action) {

switch(action.type){

case “increment”:

    return {

      ...state,

      value: state.value + 1,

    };

case “decrement”:

    return {

      ...state,

      value: state.value - 1,

    };

  }

default:

    return state;

}

}

1. **store create:** in react-redux we import createStore from redux. Then pass reducer functions.

Import {createStore} from “redux”

const store = createStore(counterReducer);

1. **provider:** now we provide store in component by wrapping the whole app. Then pass the store.

Import {Provider} from “redux”;

Import store from “. /app”;

<Provider store={store}>

<Counter />

</Provider>

1. **subscribe / connect:** we subscribe with redux store with connect() function. In connect() we pass out component which is interaction with store.

import { connect } from "react-redux"

function Counter() {

    return (

    <div>

        <h1> {count} </h1>

        <div>

            <button onClick={inc} > increment </button>

            <button onClick={dec} > decremnt </button>

        </div>

    </div>

  )

}

export default connect (mapStateToProps, mapDispatchToProps)(Counter);

1. **Argument:** then we make mapStateProps and mapDispatchToProps function and pass as argument in connect() function.

**react-Redux hook:**

useSelector():

# redux toolkit

1. **install redux toolkit:** we install redux toolkit by command in terminal.

Npm install @reduxjs/toolkit

1. **createSliec:** A function that accepts an initial state, an object of reducer functions, and a "slice name", and automatically generates action creators and action types that correspond to the reducers and state.

**Syntax:**

function createSlice ({name: string, initialState: any, reducers: Object})

* 1. initialState: The initial state value for this slice of state.
  2. Name: A string name for this slice of state. Generated action type constants will use this as a prefix.
  3. Reducers: An object containing Redux "case reducer" functions.

Code:

import { createSlice } from '@reduxjs/toolkit'  
const counterSlice = createSlice({  
 name: 'counter',  
 initialState: 0,  
 reducers: {  
 increment: (state) => state + 1,  
 },  
})

* 1. **Return Value**[**​**](https://redux-toolkit.js.org/api/createslice#return-value)**:** createSlice will return an object that looks like:

{  
 name : string,  
 reducer : ReducerFunction,  
 actions : Record<string, ActionCreator>,  
 caseReducers: Record<string, CaseReducer>.  
 getInitialState: () => State  
}

1. Redux Store: