ReactJS

Before you start, you should have a basic understanding of

What is HTML

What is CSS

What is DOM

What is ES6

What is Node.js

What is npm

Overview

React JS is a open source JavaScript Library for create Reusable Components. React JS is developed and maintained by Facebook and the main aim of React JS is Better performance web applications. React JS carry out User Interface of the applications which means **"View"** in Model-View-Controller. React JS works template language for your Application.

Why ReactJs

There are plenty of frontend frameworks available for web applications, let's have a look about how React JS have benefits over other Technologies.

Simplicity

- React JS is very simple to use.
- It uses plain JavaScript, so easy to lern and development.
- Well component based Structured Approach.
- React JS introduced special syntax called JSX which makes developers to write HTML with JavaScript.
- JSX is much easier than JavaScript

Very Easy to learn

- If you have basic programming knowledge, you can easily learn Ract.
- Less knowledge in HTML & JavaScript is enough to Start learn React JS.

Why ReactJs

Native Approach

- Not only for web applications, you can use React for Mobile Application also.
- It allows developers to reuse the code.
- We can use the same React code for Web Application, Android and iOS Applications.

Better Performance

- React JS offers Better Performance since it's component based approach.

Well Testability

- React JS applications are very easy to test.

ES6 Javascript Basics
“Developer Must Know”

1. Default Parameters in ES6
2. Template Literals in ES6
3. Multi-line Strings in ES6
4. Destructuring Assignment in ES6
5. Enhanced Object Literals in ES6
6. Arrow Functions in ES6
7. Promises in ES6
8. Classes in ES6

History

First, a bit of history because those who don’t know the history can’t make it.
This is a brief JavaScript timeline:

1. 1995: JavaScript is born as LiveScript
2. 1997: ECMAScript standard is established
3. 1999: ES3 comes out and IE5 is all the rage
4. 2000–2005: XMLHttpRequest, a.k.a. AJAX, gains popularity in app such as Outlook Web Access (2000) and Oddpost (2002), Gmail (2004) and Google Maps (2005).
5. 2009: ES5 comes out (this is what most of us use now) with `forEach`, `Object. Keys`, `Object. Create` (specially for Douglas Crockford), and standard JSON
6. 2015: ES6/ECMAScript2015 comes out; it has mostly syntactic sugar, because people weren’t able to agree on anything more ground breaking (ES7?)

Enough with history, let’s get to the business of coding.

Default Parameters in ES6

```
var link = function (height, color, url) {
      var height = height || 50
      var color = color || 'red'
      var url = url || 'http://tops-int.com' ...
}
```

They were okay until the value was 0 and because 0 is falsy in JavaScript it would default to the hard-coded value instead of becoming the value itself. Of course, who needs 0 as a value (#sarcasmfont), so we just ignored this flaw and used the logic OR anyway... No more! In ES6, we can put the default values right in the signature of the functions:

Template Literals in ES6

Template literals or interpolation in other languages is a way to output variables in the string. So in ES5 we had to break the string like this:

```
var name = 'Your name is ' + first + ' ' + last + '.'
var url = 'http://localhost:3000/api/messages/' + id
```

Luckily, in ES6 we can use a new syntax `${NAME}` inside of the back-ticked string:

```
var name = `Your name is ${first} ${last}.`
var url = `http://localhost:3000/api/messages/${id}`
```

Multi-line Strings in ES6

ES5

```
var roadPoem = 'Then took the other, as just
as fair,\n\t'
+ 'And having perhaps the better claim\n\t'
+ 'Because it was grassy and wanted wear,\n\t'
+ 'Though as for that the passing there\n\t'
+ 'Had worn them really about the same,\n\t'
var fourAgreements = 'You have the right to be
you.\n\
You can only be you when you do your best.'
```

ES6

```
var roadPoem = `Then took the other, as just
as fair,
And having perhaps the better claim Because
it was grassy and wanted wear,
Though as for that the passing there Had
worn them really about the same,`

var fourAgreements = `You have the right to
be you.
You can only be you when you do your best.`
```

Destructuring Assignment in ES6

Destructuring can be a harder concept to grasp, because there's some magic going on... let's say you have simple assignments where keys house and mouse are variables house and mouse:

```
var {house, mouse} = $('body').data() // we'll get house and mouse variables
var {json: jsonMiddleware} = require('body-parser')
var {username, password} = req.body
```

This also works with arrays

```
var [col1, col2] = $('.column'),
[line1, line2, line3, , line5] = file.split('\n')
```

Enhanced Object Literals in ES6

```
var serviceBase = {port: 3000, url: 'azat.co'},
getAccounts = function(){return [1,2,3]}
var accountService = { __proto__: serviceBase, getAccounts,
 toString() {
      return JSON.stringify((super.valueOf()))
 },
 getUrl() {
      return "http://" + this.url + ':' + this.port},
      [ 'valueOf_' + getAccounts().join('_') ]: getAccounts()
 };
 console.log(accountService)
```

Arrow Functions in ES6

```
$('.btn').click((event) =>{
this.sendData()
})
```

```
var logUpperCase = function() {
      this.string = this.string.toUpperCase()
      return () => console.log(this.string)
}
logUpperCase.call({ string: 'es6 rocks' })()
```

Classes in ES6

```
class baseModel {
      constructor(options = {}, data = []) { // class constructor
              this.name = 'Base'
              this.url = 'http://azat.co/api'
              this.data = data
              this.options = options
      }
      getName() { // class method
              console.log(`Class name: ${this.name}`)
      }
}
```

Module1

JavaScript

Basic JavaScript

- - Understanding **var, let** and **Const**
- - JS switch, if, else,
- - JS loop
- - Javascript JSON

Functions

- - Function Declaration in JS
- - Arrow Functions
- - Higher Order Functions
- - Map, Reduce and Filter

Classes

- - Javascript Classes and Objects
- - Class Properties and Methods
- - This keyword and binding in JS
- - Class inheritance
- - Exports and Imports

Array in JS

- - Creating Array
- - Array methods
- - The Spread & Rest operators
- - Destructuring

JS Async

- - Callbacks
- - Promises
- - Async/Await

JavaScript Introduction

- JavaScript is a cross-platform, object-oriented scripting language invented in web browsers to make web pages more dynamic and give feedback to your user. Adding JavaScript to your HTML code allows you to change completely the document appearance, from changing text, to changing colors, or changing the options available in a drop-down list, or switching one image with another when you roll your mouse over it and much more.

- JavaScript is mainly used as a client side scripting language. This means that all the action occurs on the client's side of things. When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it's up to the browser to do something with it.

What can JavaScript do?

Display information based on the time of the day

JavaScript can check the computer's clock and pull the appropriate data based on the clock information.

Detect the visitor's browser

JavaScript can be used to detect the visitor's browser, and load another page specifically designed for that browser.

Control Browsers

JavaScript can open pages in customized windows, where you specify if the browser's buttons, menu line, status line or whatever should be present.

Validate forms data

JavaScript can be used to validate form data at the client-side saving both the precious server resources and time.

What can JavaScript do?

Create Cookies

JavaScript can store information on the visitor's computer and retrieve it automatically next time the user visits your page.

Add interactivity to your website

JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on a button.

Change page contents dynamically

JavaScript can randomly display content without the involvement of server programs .It can read and change the content of an HTML elements or move them around

How to implement JavaScript to an HTML page

You can link to outside files (with the file extension .js), or write blocks of code right into your HTML documents with the <script> tag. So, the <script type="text/javascript"> and </script> tells where the JavaScript starts and ends.

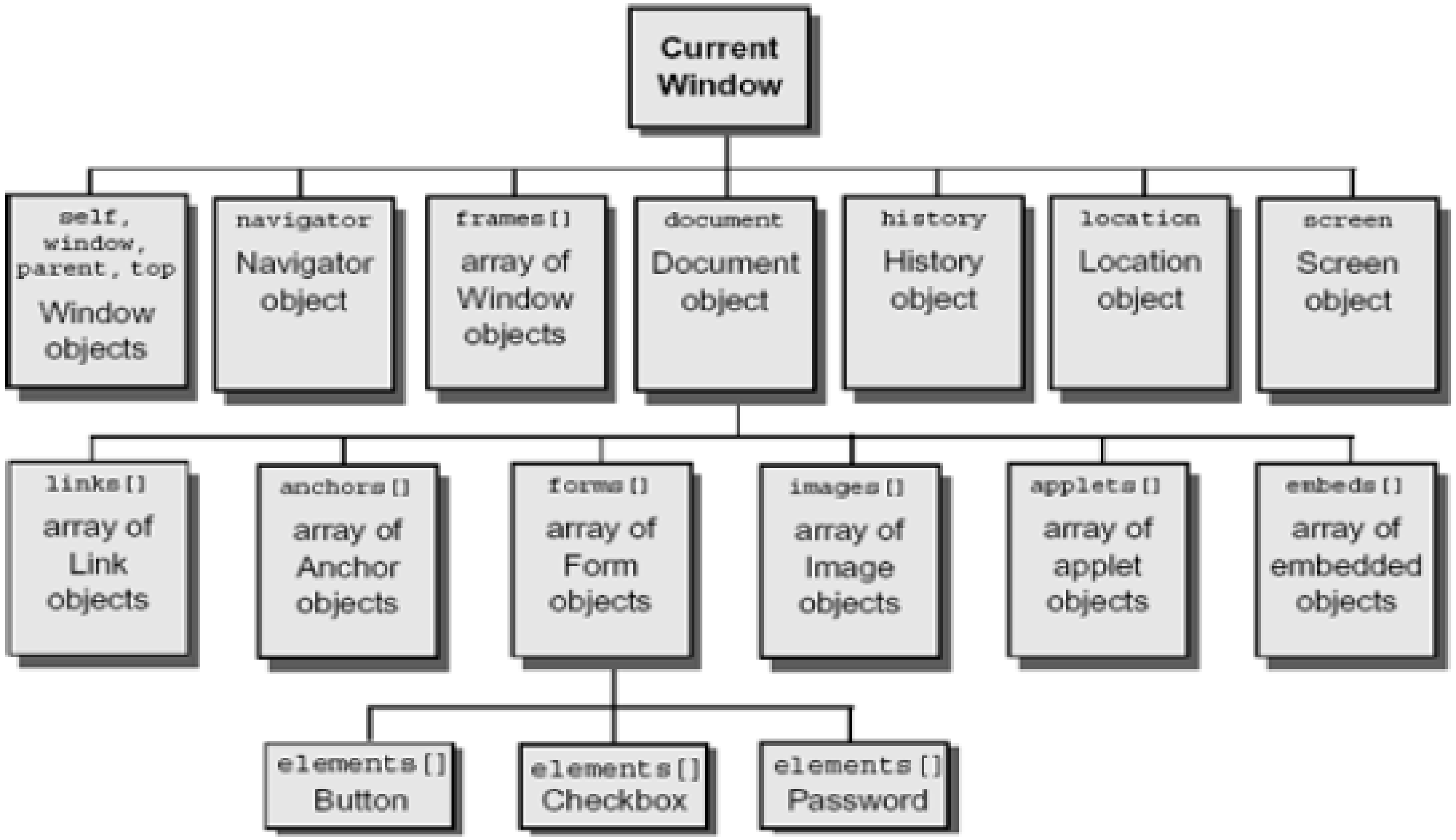
```
<html>
   <head>
   <title>My First Script</title>
   </head>
<body>
   <script type="text/javascript">
   <!--
   //-->
   </script>
</body>
</html>
```

Functions

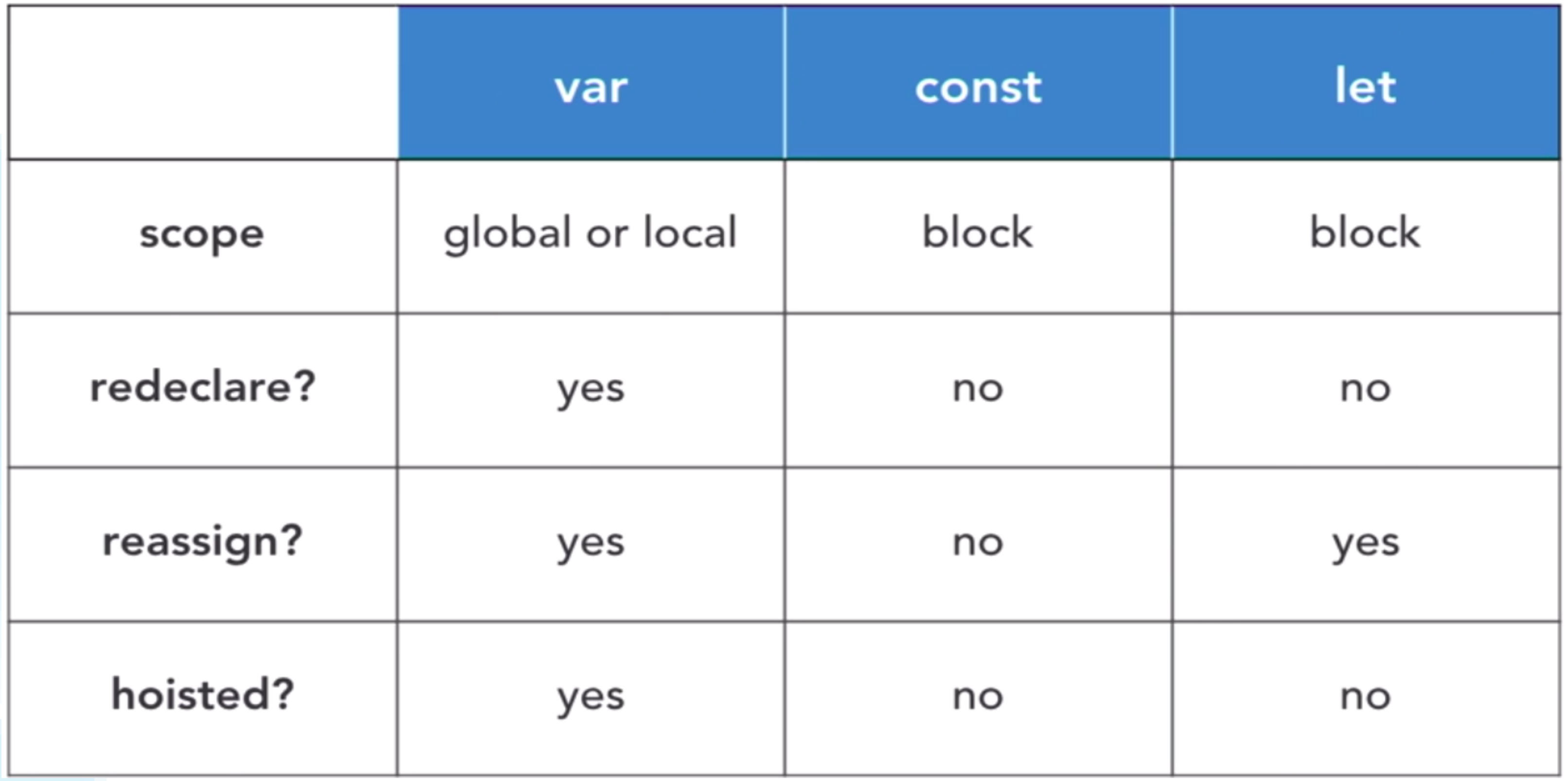
Defining

```
<script type="text/javascript">
   function function_name (argument_1, ... , argument_n){
          statement_1;
          statement_2;
          statement_m;
          return return_value;
   }
</script>
```

```
<html>
<body>
 <script type="text/javascript">
 document.write("Hello World!");
</script>
</body>
</html>
```



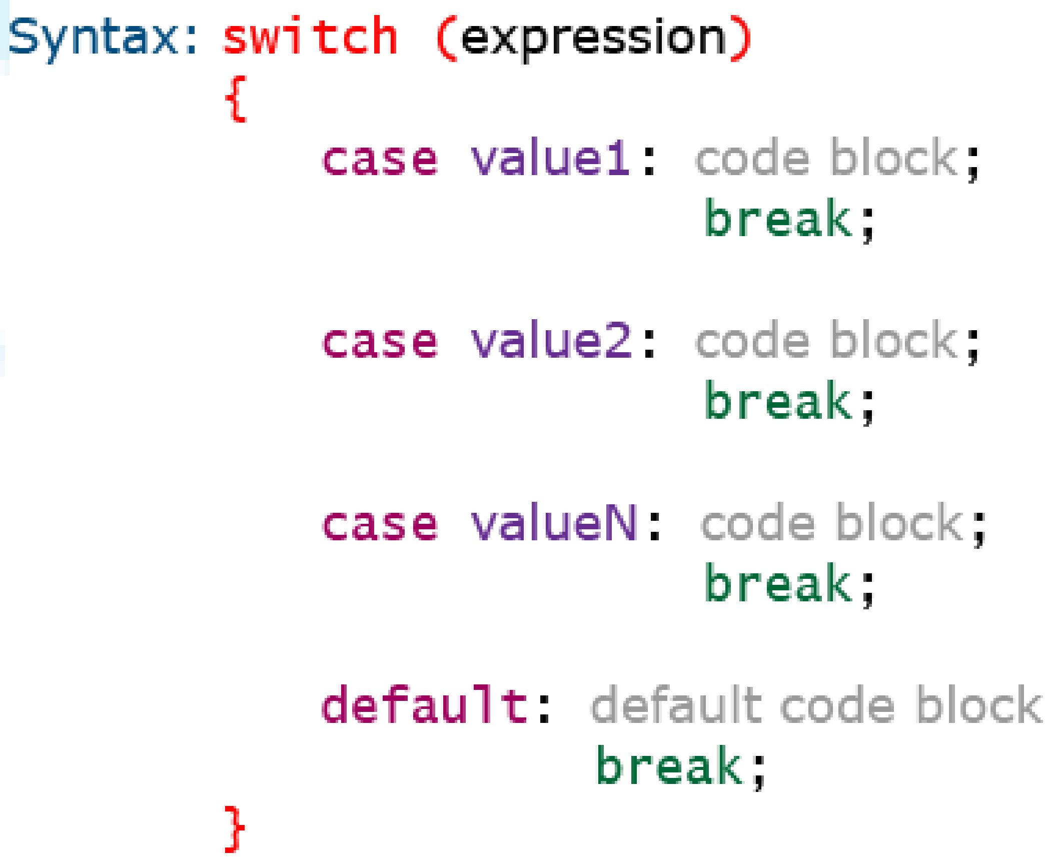
Var, let and Const



	var	const	let
scope	global or local	block	block
redeclare?	yes	no	no
reassign?	yes	no	yes
hoisted?	yes	no	no

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.1.1letVarConst.html

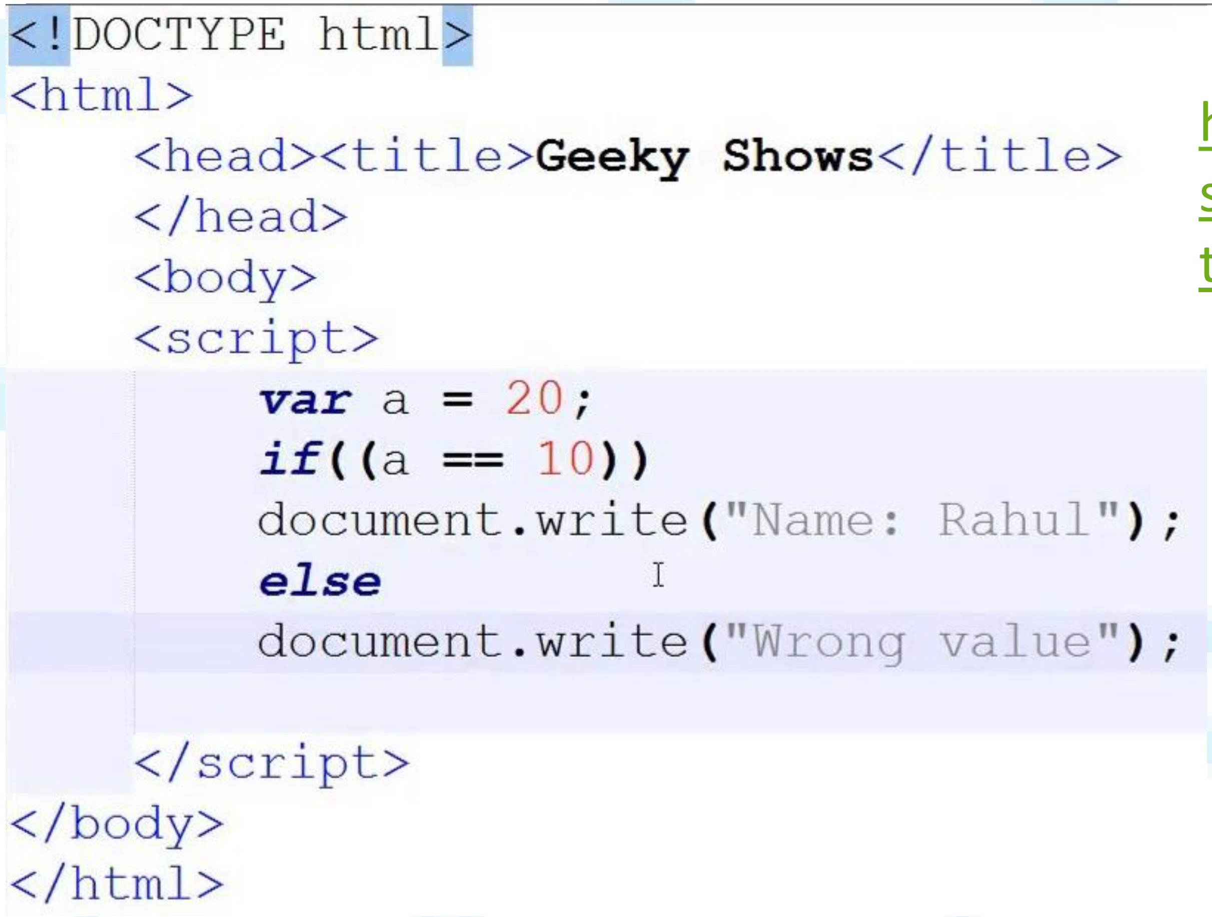
Switch case



```
Syntax: switch (expression)
        {
           case value1:  code block;
                         break;

           case value2:  code block;
                         break;

           case valueN:  code block;
                         break;

           default:  default code block
                     break;
        }
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.1.2.1SwitchCase.html

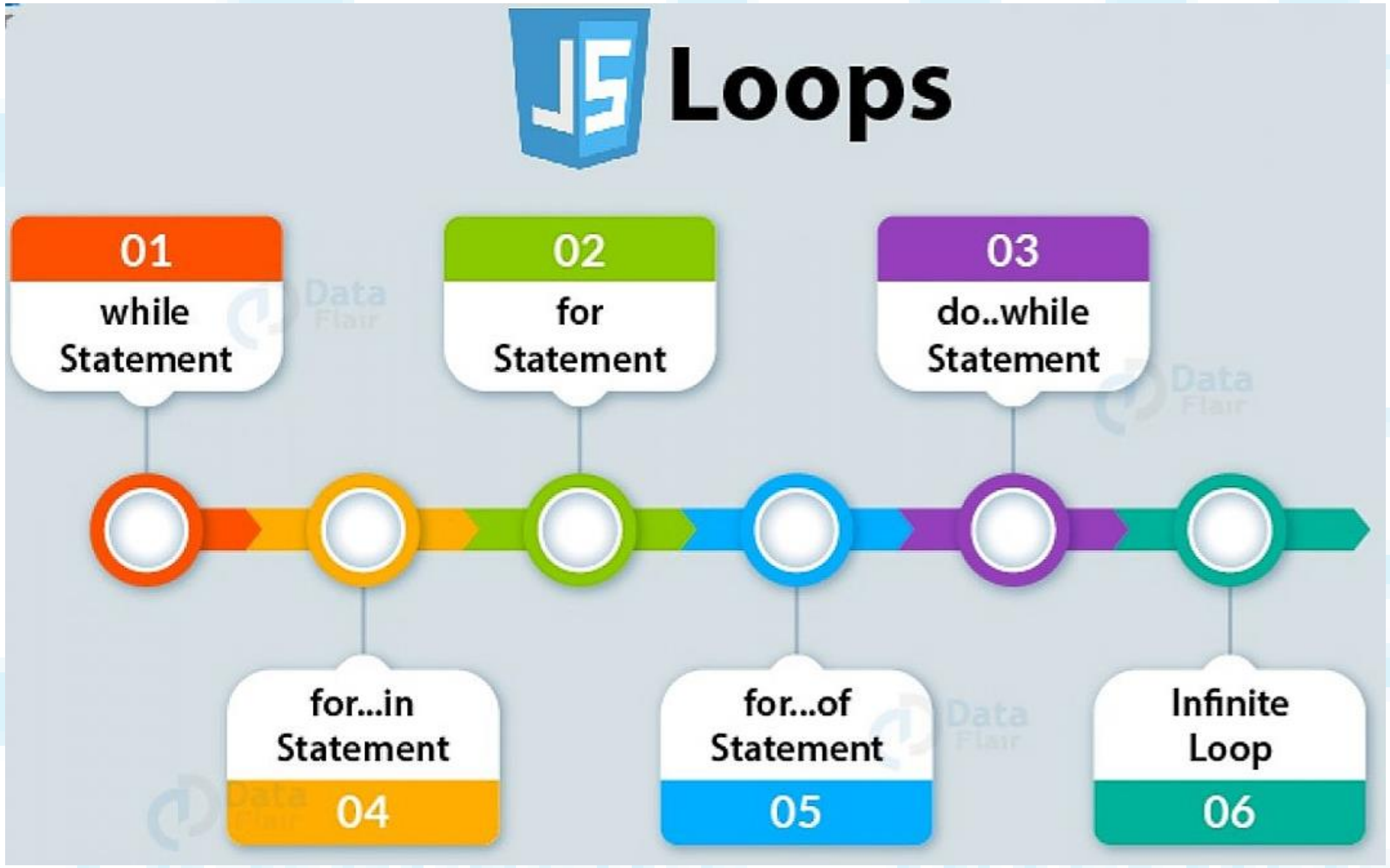
If-Else

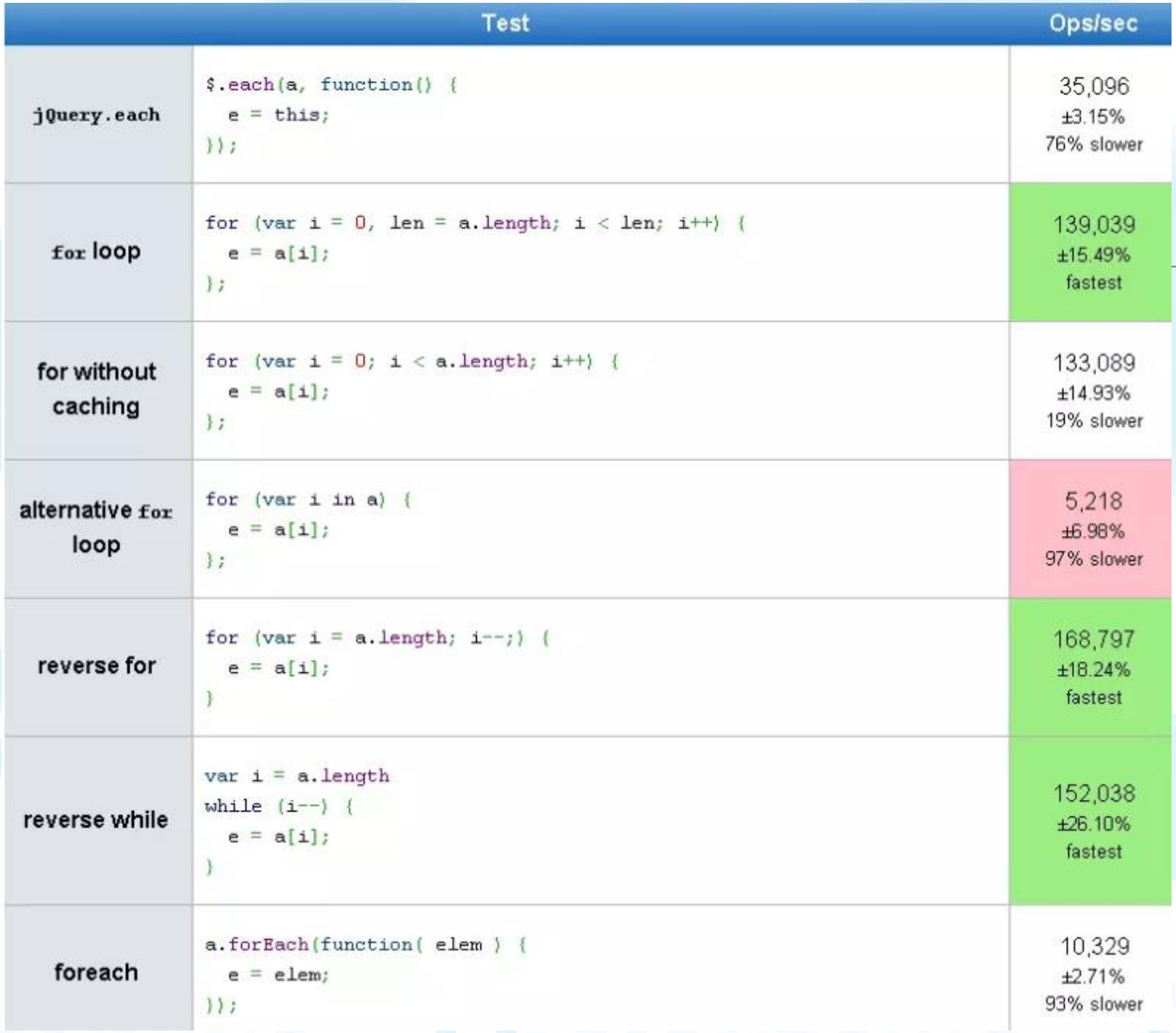


```
<!DOCTYPE html>
<html>
    <head><title>Geeky Shows</title>
    </head>
    <body>
    <script>
        var a = 20;
        if((a == 10))
        document.write("Name: Rahul");
        else
        document.write("Wrong value");

    </script>
</body>
</html>
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.1.2.2ifelse.html

loops


Loops
01
while Statement
02
for Statement
03
do..while Statement
for...in Statement
04
for...of Statement
05
Infinite Loop
06

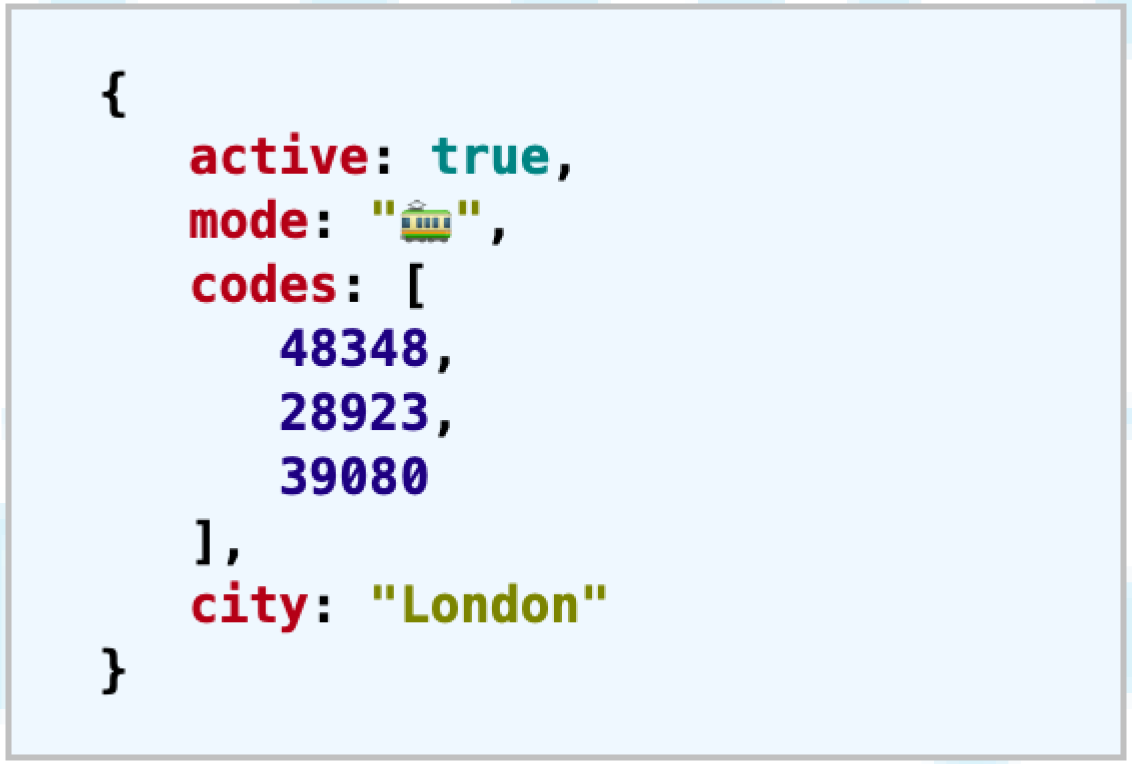


Test		Ops/sec
`jQuery.each`	`$.each(a, function() { e = this; });`	35,096 ±3.15% 76% slower
`for` loop	`for (var i = 0, len = a.length; i < len; i++) { e = a[i]; };`	139,039 ±15.49% fastest
for without caching	`for (var i = 0; i < a.length; i++) { e = a[i]; };`	133,089 ±14.93% 19% slower
alternative `for` loop	`for (var i in a) { e = a[i]; };`	5,218 ±6.98% 97% slower
reverse for	`for (var i = a.length; i--;) { e = a[i]; }`	168,797 ±18.24% fastest
reverse while	`var i = a.length while (i--) { e = a[i]; }`	152,038 ±26.10% fastest
foreach	`a.forEach(function( elem ) { e = elem; });`	10,329 ±2.71% 93% slower

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.1.4foreach.html

JSON

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight data interchange format
- JSON is language independent *
- JSON is "self-describing" and easy to understand

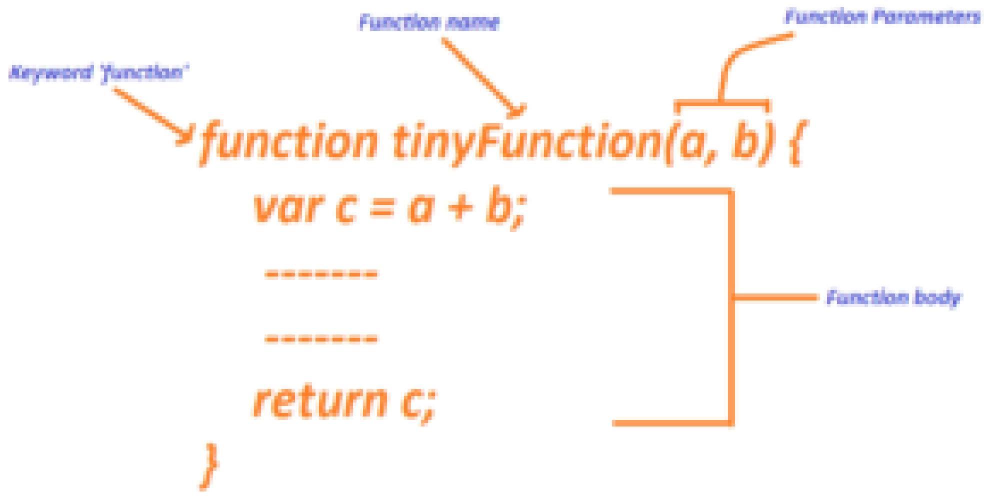


```
{
    active: true,
    mode: "🚋",
    codes: [
        48348,
        28923,
        39080
    ],
    city: "London"
}
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.1.4JsonText.html

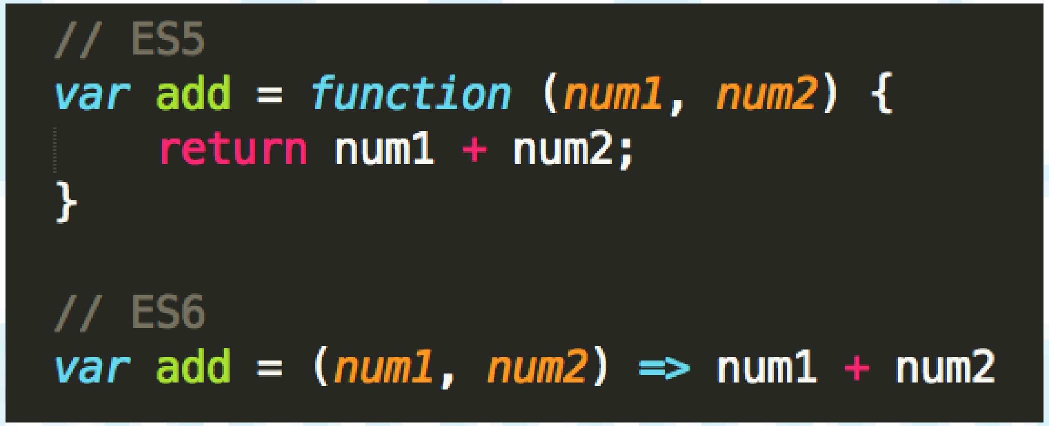
JavaScript Functions

SIMPLE FUNCTION



```
function tinyFunction(a, b) {
    var c = a + b;
    --------
    --------
    return c;
}
```

ARROW FUNCTION



```
// ES5
var add = function (num1, num2) {
    return num1 + num2;
}

// ES6
var add = (num1, num2) => num1 + num2
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.2.1Function.html

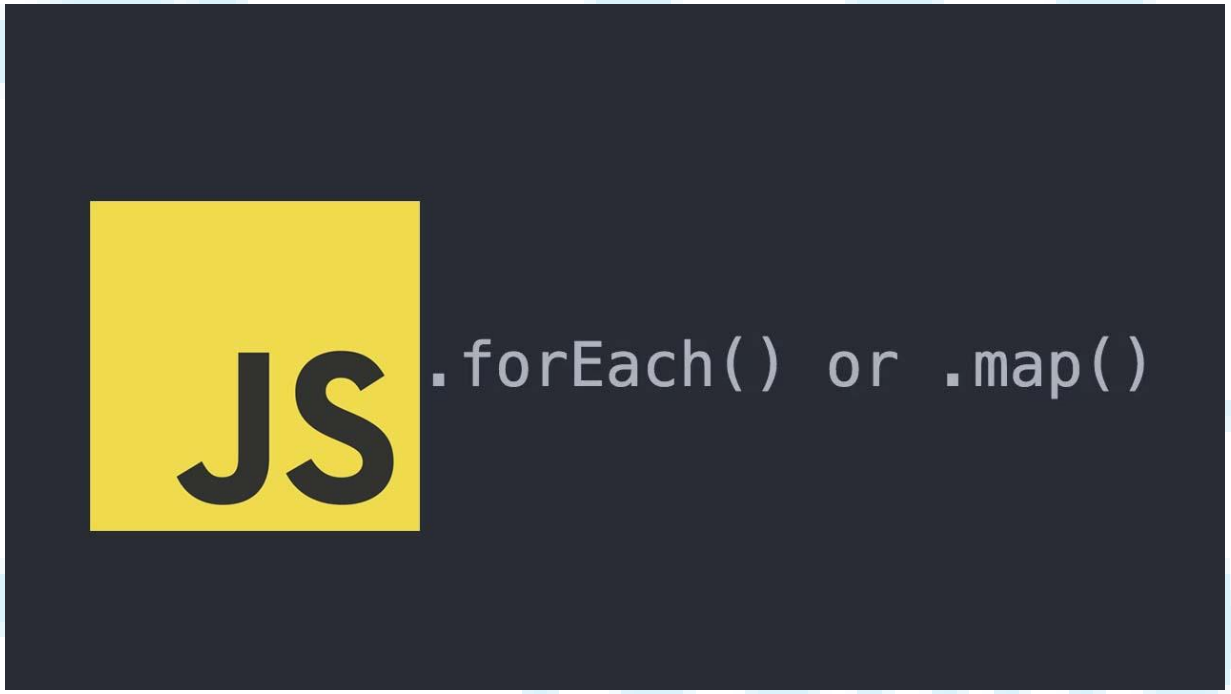
Higher Order Functions

A higher-order function is a function that can take another function as an argument, or that returns a function as a result.

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.2.3.1HighOrderfunction.html

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.2.3.2HighOrderfunction.html

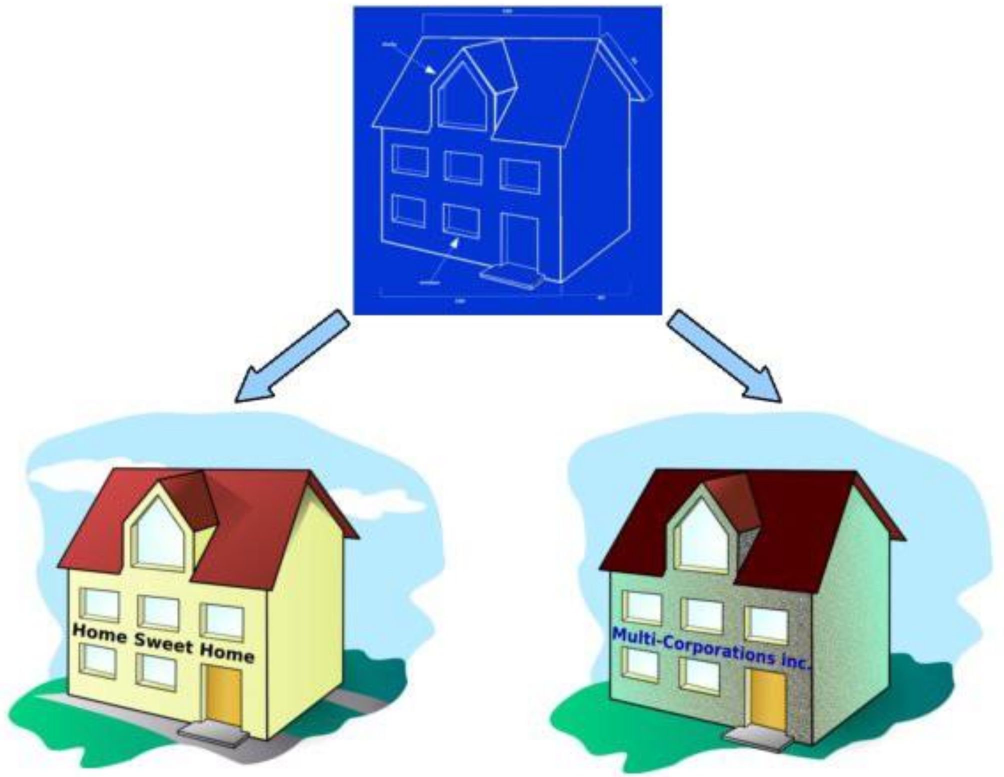
forEach() Or map()



https://github.com/TopsCode/ReactJs/blob/main/Module1/1.2.4.1map.html
https://github.com/TopsCode/ReactJs/blob/main/Module1/1.2.4.2filter.html

Class Object

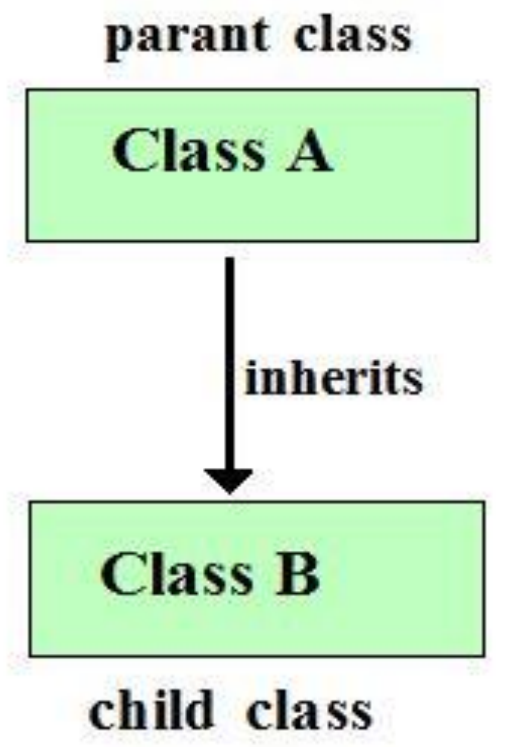
```
Syntax :

class ClassName{
      // class body
}

//creating an object
new name()
```



https://github.com/TopsCode/ReactJs/blob/main/Module1/ 1.3.1Class.html
https://github.com/TopsCode/ReactJs/blob/main/Module1/ 1.3.2Class.html

Inheritance

```
Syntax :

class name extends
otherName {
// class body
}

//creating an object
new name()
```



https://github.com/TopsCode/ReactJs/blob/main/Module1/1.3.4Inheritance.html

Array

An array is a special variable, which can hold more than one value at a time.

```
var faculties = [“Jigar”, “Jay”, “Rahul”];
var  faculties = new Array(“Jigar”, “Jay”, “Rahul”);
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.4.1Array.html

Spread in array literals

Without spread syntax, to create a new array using an existing array as one part of it, the array literal syntax is no longer sufficient and imperative code must be used instead using a combination of `push()`, `splice()`, `concat()`, etc. With spread syntax this becomes much more succinct:

Simple :

```
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5]; // Append all items from arr2 onto arr1
arr1 = arr1.concat(arr2);
```

spread syntax this becomes:

```
let arr1 = [0, 1, 2];
let arr2 = [3, 4, 5];
arr1 = [...arr1, ...arr2];
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.4.2ArraySpread.html

Array Destructuring assignment

The **destructuring assignment** syntax is a JavaScript expression that makes it possible to unpack values from arrays, or properties from objects, into distinct variables.

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.4.4ArrayDestructing.html
https://github.com/TopsCode/ReactJs/blob/main/Module1/1.4.3ArrayRest.html

JavaScript Callbacks

A callback is a function passed as an argument to another function

This technique allows a function to call another function

A callback function can run after another function has finished

When to Use a Callback?

They are simplified to teach you the callback syntax.

Where callbacks really shine are in asynchronous functions, where one function has to wait for another function (like waiting for a file to load).

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.5.1.1JavaScriptCallBack.html
https://github.com/TopsCode/ReactJs/blob/main/Module1/1.5.1.2JavaScriptCallBack.html

Asynchronous JavaScript Waiting for a Timeout

When using the JavaScript function `setTimeout()`, you can specify a callback function to be executed on time-out:

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.5.3.1Async.html
https://github.com/TopsCode/ReactJs/blob/main/Module1/1.5.3.2AsyncAwait.html

JavaScript Promise Object

"Producing code" is code that can take some time

"Consuming code" is code that must wait for the result

A Promise is a JavaScript object that links producing code and consuming code

Promise Object Properties

myPromise.state	myPromise.result
"pending"	undefined
"fulfilled"	a result value
"rejected"	an error object

https://github.com/TopsCode/ReactJs/blob/main/Module1/1.5.2.1Promises.html
https://github.com/TopsCode/ReactJs/blob/main/Module1/1.5.2.2Promises.html

Module2

Components, State, Props

COMPONENTS, STATE, PROPS

JSX

- - Why JSX?
- - Embedding Expressions in JSX
- - Attributes with JSX
- - Children with JSX

Installation

- - Add React to a HTML Website
- - Create New React App
- - Hello World

Components, State, Props

- - Function Component
- - Class Component
- - Props
- - State
- - Class Component Lifecycle

Installation

Text Editor/Source Code Editor – Visual Studio Code, Notepad++, Atom etc

Web Browser – Google Chrome, Firefox

React Developer Tools

Requirements

NPM - It is used to take advantage of a vast ecosystem of third-party packages, and easily install or update them.

Webpack - webpack is a static module bundler for modern JavaScript applications. When webpack processes your application, it internally builds a dependency graph which maps every module your project needs and generates one or more bundles.

Babel - Babel is a toolchain that is mainly used to convert ECMAScript 2015+ code into a backwards compatible version of JavaScript in current and older browsers or environments.

Create React App

Create React App is a tool, built by developers at Facebook that gives you a massive head start when building React apps. It saves you from time consuming setup and configuration. You simply run one command and Create React App sets up the tools you need to start your React project.

Syntax:-

```
npx create-react-app my-app
```

Create React App

Installs the latest version of React, React-DOM and react-scripts.

React, JSX, ES6, TypeScript and Flow syntax support.

Autoprefixed CSS, so you don’t need -webkit- or other prefixes.

Installs the dependencies needed to build project

Zero Setup and Configuration Required

- Webpack – It configuring Webpack to bundle your project assets.
- Babel – It transpile JSX into browser-ready code.

Creates the initial project structure

Development Server

Built-in Testing - Jest testing framework

Create React App provides helpful runtime error messages in the browser

Command Line Client

npm is installed with Node.js

This means that you have to install Node.js to get npm installed on your computer.

Download Node.js from the official Node.js web site: https://nodejs.org

```
C:\>npm install <package>
```

https://nodejs.org/en/

npm

- npm is the world's largest Software Library (Registry)
- npm is also a software Package Manager and Installer

Installation through node

Check Node version

- npm –version

- npm install -g create-react-app
- create-react-app –version

- create-react-app app-name

- cd app-name
- npm start

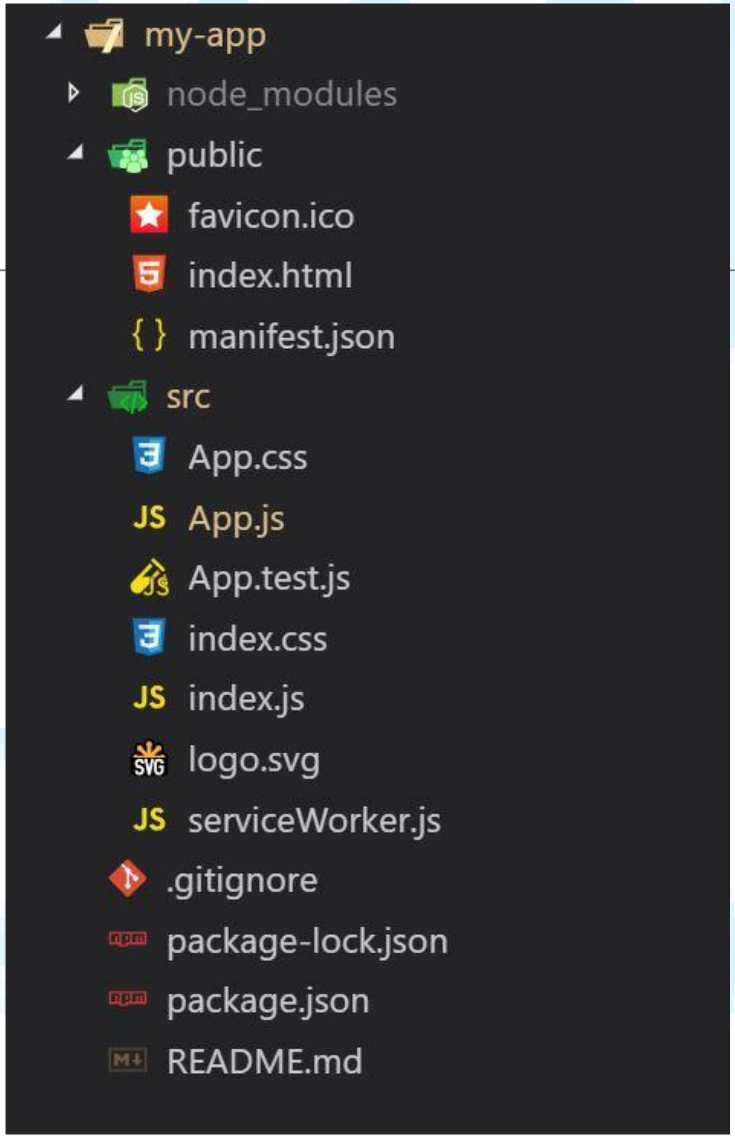
If you've previously installed create-react-app globally via npm install -g create-react-app, we recommend you uninstall the package using npm uninstall -g create-react-app to ensure that npx always uses the latest version.

- New node 12 version

https://stackoverflow.com/questions/59188624/template-not-provided-using-create-react-app

- npx create-react-app my-app
- npm init react-app my-app
- yarn create react-app my-app

Directory Structure



```
my-app
  node_modules
  public
    favicon.ico
    index.html
    manifest.json
  src
    App.css
    App.js
    App.test.js
    index.css
    index.js
    logo.svg
    serviceWorker.js
  .gitignore
  package-lock.json
  package.json
  README.md
```

Directory Structure

my-app – This is your Project Name

node_modules – It contains all packages and dependencies installed.

public -

favicon.ico – It's a favicon for website

index.html – This file holds the HTML template of our app.

manifest.json - manifest.json provides metadata used when your web app is installed on a user's mobile device or desktop.

src

App.css – It's a css file related to App.js but its global.

App.js – It's Parent component of your react app

App.test.js – Its Test environment

index.css - It's a css file related to index.js but its global

index.js – It's the JavaScript entry Point.

logo.svg – React logo file

serviceWorker.js – It can help to access website offline.

For the project to build, these files must exist with exact filenames

Directory Structure

gitignore – It is used when you want to ignore git push.

package-lock.json – Its version control package json file

package,json – All dependencies mentioned in this file.

README.md – It readme file

public Folder

If you put a file into the public folder, it will not be processed by Webpack (Webpack does not read the public folder; it will only read the files inside the src folder). Instead it will be copied into the build folder untouched.

To reference assets in the public folder, you need to use a special variable called PUBLIC_URL.

Only files inside the public folder will be accessible by %PUBLIC_URL% prefix.

```
<link rel="shortcut icon" href="%PUBLIC_URL%/favicon.ico">
```

When you run npm run build, Create React App will substitute %PUBLIC_URL% with a correct absolute path so your project works even if you use client-side routing or host it at a non-root URL.

public Folder

You need a file with a specific name in the build output, such as manifest.webmanifest.

You have thousands of images and need to dynamically reference their paths.

You want to include a small script like pace.js outside of the bundled code.

Some library may be incompatible with Webpack and you have no other option but to include it as a <script> tag.

render() Method

The render() method is the only required method in a class component. It examines this.props and this.state .

It returns one of the following types:

React elements – These are created via JSX(Not required).

For example, <div /> and <App /> are React elements that instruct React to render a DOM node, or another user-defined component, respectively.

Arrays and fragments - It is used to return multiple elements from render.

Portals – It is used to render children into a different DOM subtree.

String and numbers - These are rendered as text nodes in the DOM.

Booleans or null - It renders nothing. (Mostly exists to support return test && <Child /> pattern, where test is boolean.)

Note - The render() function should be pure, meaning that it does not modify component state, it returns the same result each time it's invoked, and it does not directly interact with the browser.

React Element

You can create a react element using React.createElement() method but there is a easy way to create element via JSX.

Using createElement() Method

React.createElement(“h1”, null, “Hello World”);

Using JSX

<h1>Hello World</h1>

React.createElement(type, props, children)

React.createElement(type, props, children) - It creates a React Element with the given arguments.

Syntax:- React.createElement(type, props, children)

type: Type of the html element or component. (example : h1,h2,p,button..etc).

props: The properties object.

Example: {style :{ color:“blue”}} or className or event handlers etc.

children: anything you need to pass between the dom elements.

Ex:-

React.createElement(‘h1’, null, ‘Hello GeekyShows’);

ReactDOM.render(element, DOMnode)

ReactDOM.render(element, DOMnode) - It takes a React Element and render it to a DOM node.

Syntax:- ReactDOM.render(element, DOMnode)

> The first argument is which component or element needs to render in the dom.
>
> The second argument is where to render in the dom.

Ex:-

ReactDOM.render(< App />, document.getElementById(“root”));

ReactDOM.render(element, DOMnode)

Webpack parses through the application starting at src/index.js, following any imported modules, until it has a complete dependency graph.

In order to convert the ES2015+ code that Webpack comes across into a version of JavaScript that will behave consistently across browsers, Webpack passes any JavaScript files it comes across through Babel.

Babel is a transpiler which parses newer and experimental JavaScript syntax, and transforms the code into a version of JavaScript which has better support across browsers.

Files src/index.js and public/index.html, these files can be modified as necessary, but the names and locations shouldn't be altered.

App.test.js is a simple unit test Create React App sets up to test if the App component renders without crashing.

What is Babel?

Babel is a JavaScript compiler that can translate markup or programming languages into JavaScript.

With Babel, you can use the newest features of JavaScript (ES6 - ECMAScript 2015).

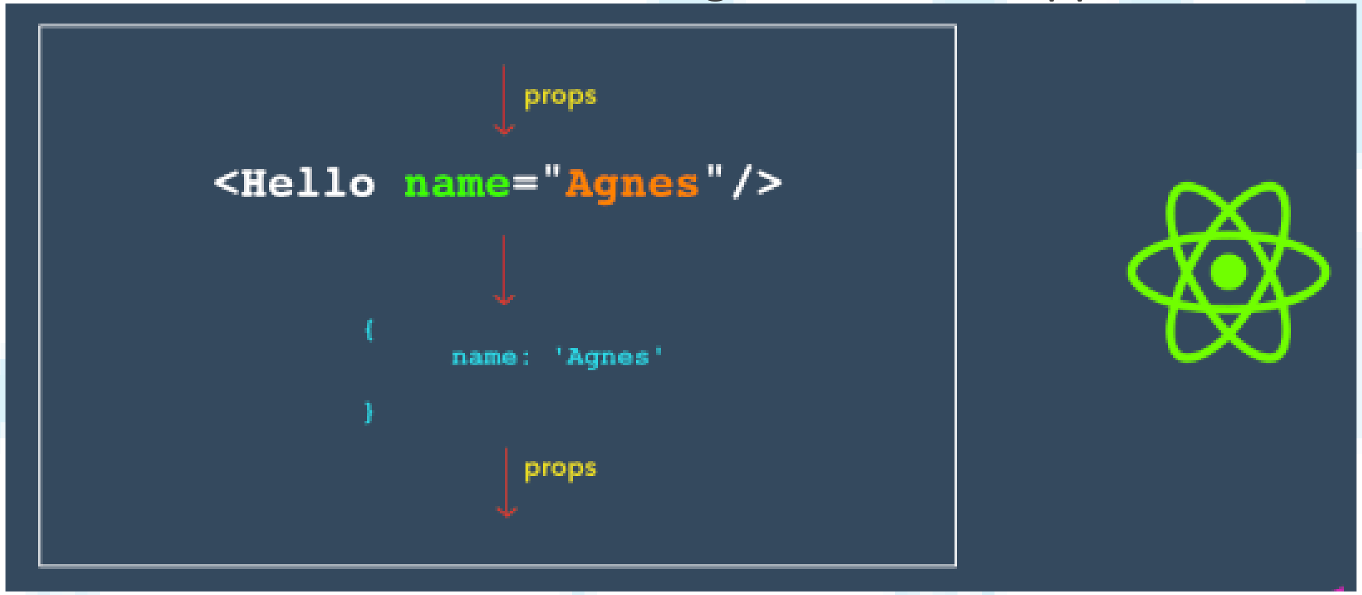
Babel is available for different conversions. React uses Babel to convert JSX into JavaScript.

- Please note that <script type="text/babel"> is needed for using Babel.

Components and Props

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. This page provides an introduction to the idea of components.

Conceptually, components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.



Function and Class Components

FUNCTION COMPONENT

```
function Welcome(props) {
      return <h1>Hello, {props.name}</h1>;
 }
```

CLASS COMPONENT

```
class Welcome extends React.Component {
                render() {
                    return <h1>Hello, {this.props.name}</h1>;
                }
 }
```

Components

Components are the building blocks of any React app.

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation.

Components are like JavaScript functions. They accept arbitrary inputs (called "props") and return React elements describing what should appear on the screen.

Always start component names with a capital letter.

React treats components starting with lowercase letters as DOM tags. For example, <div /> represents an HTML div tag, but <App /> represents a component requires App to be in scope.

Function Components

It is a JavaScript function which accepts a single “props” object argument with data and returns a React Element.

Syntax:-

```
function func_name ( ) {
   return React Element;
}
```

Ex:-

```
function Student( ){
   return <h1>Hello Rahul</h1>
}
```

```
const Student = ( ) =>{
   return <h1>Hello Rahul</h1>
}
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/2.3.1.1.1FunctionComponent.js

Class Component

A class component requires you to extend from React.Component. The class must implement a render() member function which returns a React component to be rendered, similar to a return value of a functional component. In a class-based component, props are accessible via this.props.

Syntax:-

```
class class_name extends Component {
  render( ){
      return React Element
   }
}
```

Ex:-

```
class Student extends Component {
  render( ){
      return <h1>Hello Rahul</h1>
   }
}
```

```
class Student extends Component {
  render( ){
       return <h1>Hello
{this.props.name}</h1>
   }
}
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/2.3.1.1CreateSimpleClassindex.js

https://github.com/TopsCode/ReactJs/blob/main/Module1/2.3.1.4AdvantagesOfComponentindex.js

Rendering a Component

```
ReactDOM.render(<Student />, document.getElementById("root"));

ReactDOM.render(<Student  name="Rahul"/>, document.getElementById("root"));
```

Ex:-

```
function Student(props){
  return <h1>Hello {props.name}</h1>
}
ReactDOM.render(<Student name="Rahul"/>, document.getElementById("root"));
```

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object "props".

Composing Components

Components can refer to other components in their output. This lets us use the same component abstraction for any level of detail. Ex:-

```
function Student(){   return <h1>Hello Rahul</h1> }

function App( ){   return (<div>

            <Student / >

            <Student / >

            <Student / >

            </div> ); }

ReactDOM.render(<App />, document.getElementById(“root”));
```

Functional vs. Class Component

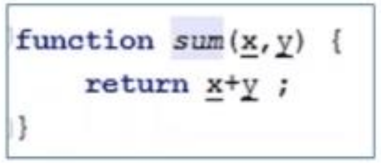
Use functional components if you are writing a presentational component which doesn’t have its own state or needs to access a lifecycle hook. You cannot use setState() in your component because Functional Components are plain JavaScript functions,

Use class Components if you need state or need to access lifecycle hook because all lifecycle hooks are coming from the React.Component which you extend from in class components.

https://github.com/TopsCode/ReactJs/blob/main/Module1/2.3.1.1CreateSimpleClassindex.js

Props

- ✓ Props are read-only components
- ✓ Whether components are declared as function or class, it must never change its *props*
- ✓ Such components are called ‘pure functions’



```
function sum(x,y) {
    return x+y ;
}
```

RULE: ***All React components must act like pure functions with respect to their props.***

Props

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object. We call this object “props”.

```
function Student(props) {
  return ( <div>
              <h1>Hello, {props.name}</h1>;
              <h2>Your Roll: {props.roll}</h2>;
               </div> );
}
```

Props

```
ReactDOM.render(  <Student name=“Rahul" roll=“101” />,  document.getElementById('root') );
```

JavaScript Expression as Props

If you pass no value for a prop, it defaults to true

```
ReactDOM.render(  <Student name={“Rahul“} roll=“101” />,  document.getElementById('root') );
ReactDOM.render(  <Student name=“Rahul" roll={100+1} />,  document.getElementById('root') );
```

Props

```
class Student extends React.Component {
  render() {  return ( <div>
              <h1>Hello, {this.props.name}</h1>;
              <h2>Your Roll: {this.props.roll}</h2>;
               </div> );
     }
}
ReactDOM.render(  <Student name=“Rahul" roll=“101” />,
document.getElementById('root') );
```

https://github.com/TopsCode/ReactJs/blob/main/Module1/2.3.1.3inheritanceWithPropsData.js

Props

Whether you declare a component as a function or a class, it must never modify its own props.

All React components must act like pure functions with respect to their props.

Pure Function

```
function sum(a, b) {
  return a + b;
}
```

Impure Function

```
function withdraw(account, amount) {
  account.total -= amount;
}
```

Props are Read-Only

https://github.com/TopsCode/ReactJs/blob/main/Module1/2.3.1.3inheritanceWithPropsData.js

Type checking With PropTypes

npm install prop-types

To run typechecking on the props for a component, you can assign the special propTypes property.

Ex:-

```
import PropTypes from 'prop-types';
Student.propTypes = {
  name: PropTypes.string
};
```

Note –

When an invalid value is provided for a prop, a warning will be shown in the JavaScript console.

For performance reasons, propTypes is only checked in development mode.

Typechecking With PropTypes

PropTypes exports a range of validators that can be used to make sure the data you receive is valid.

```
optionalArray: PropTypes.array,
optionalBool: PropTypes.bool,
optionalFunc: PropTypes.func,
optionalNumber: PropTypes.number,
optionalObject: PropTypes.object,
optionalString: PropTypes.string,
optionalSymbol: PropTypes.symbol,
```

Required

```
import PropTypes from 'prop-types';
Student.propTypes = {
  name: PropTypes.string.isRequired
};
```

Default Prop Values

You can define default values for your props by assigning to the special defaultProps property.

```
Student.defaultProps = {
  name: ‘Jay Amin'
};
```

Children in JSX

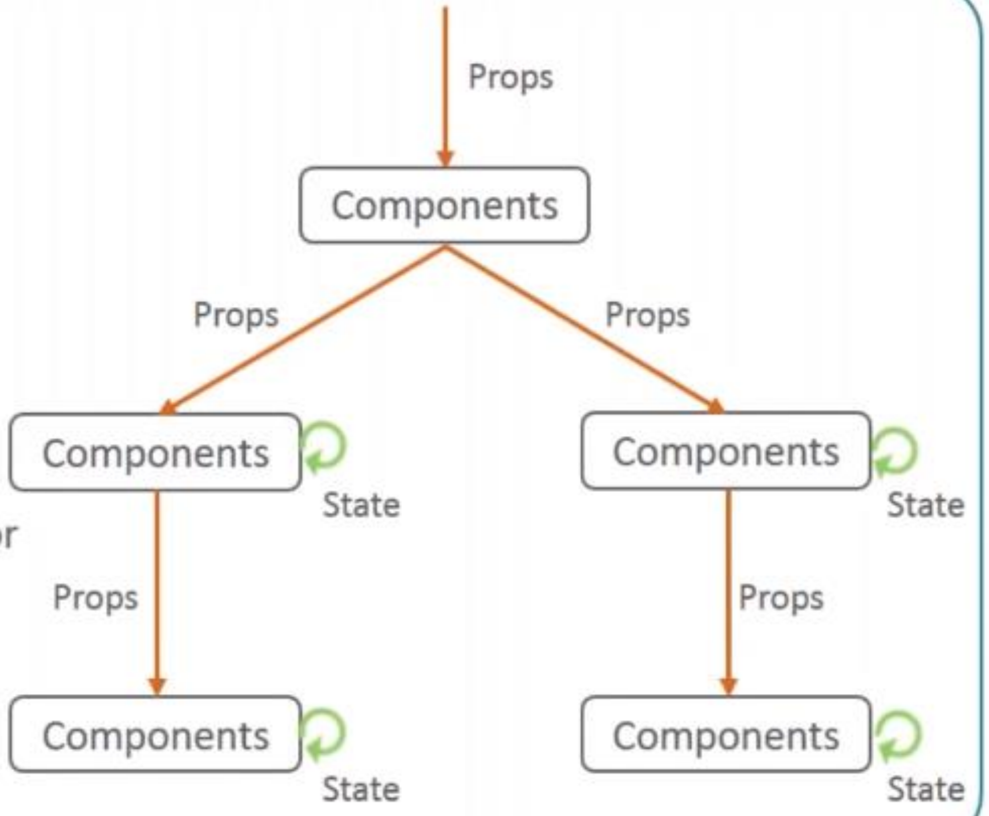
In JSX expressions that contain both an opening tag and a closing tag, the content between those tags is passed as a special prop: props.children.

Ex:- `<Student>I am child</Student>`

```
props.children    // I am child
```

States

- ✓ Heart of react components
- ✓ Must be kept as simple as possible
- ✓ Determines components rendering and behavior
- ✓ Creates dynamic and interactive components



State

State is similar to props, but it is private and fully controlled by the component. We can create state only in class component. It is possible to update the state/Modify the state.

There are two way to initialize state in React Component :-

- Directly inside class
- Inside the constructor

Directly inside class

```
class Student extends Component {
  // States  - Here it is a class property
state = {
    name: “Rahul”,
    prop1: this.props.prop1
  }
  render() {
  }
}
```

Note -
The state property is referred as state.
This is a class instance property.

Inside the Constructor

```
class Student extends Component {
  constructor(props) {
    // It is required to call the parent class
constructor
    super(props);
    this.state = { name: “Jay”,
      prop1: this.props.prop1}
  }
  render() {   }
}
```

- When the component class is created, the constructor is the first method called, so it’s the right place to add state.
- The class instance has already been created in memory, so you can use *this* to set properties on it.
- When we write a constructor, make sure to call the parent class’ constructor by super(props)
- When you call super with props. React will make props available across the component through this.props

Update State

setState ( ) Method is used to update states.

Ex:-

```
this.state = {
    name: “Jai”
}
this.setState({name: “Jay”});
```

Update State

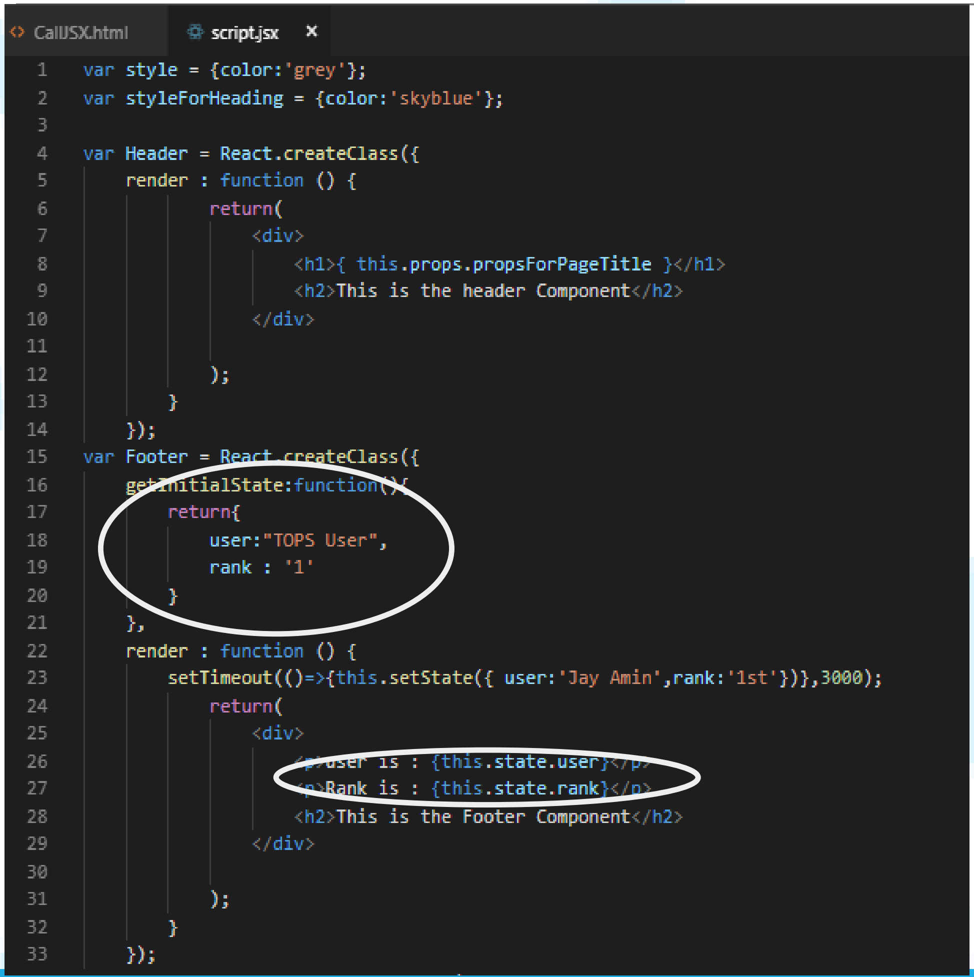
```
this.setState(function(state, props) {
  return {
  };
});
```

It accepts a function rather than an object.

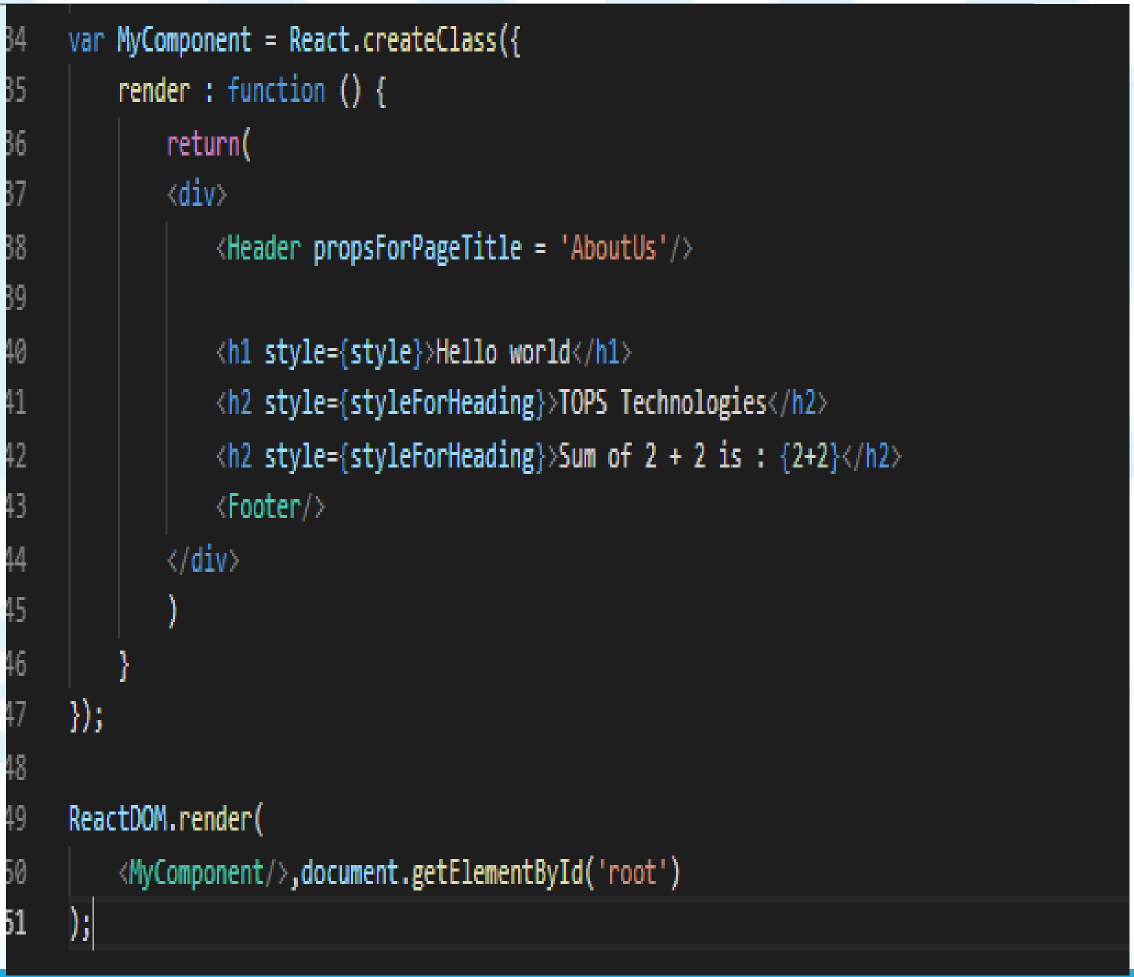
It receives the previous state as the first argument,

The props at the time the update is applied as the second argument.

States



```
var style = {color:'grey'};
var styleForHeading = {color:'skyblue'};

var Header = React.createClass({
    render : function () {
            return(
                <div>
                    <h1>{ this.props.propsForPageTitle }</h1>
                    <h2>This is the header Component</h2>
                </div>

            );
        }
    });
var Footer = React.createClass({
    getInitialState:function(){
        return{
            user:"TOPS User",
            rank : '1'
        }
    },
    render : function () {
        setTimeout(()=>{this.setState({ user:'Jay Amin',rank:'1st'})},3000);
            return(
                <div>
                    <p>User is : {this.state.user}</p>
                    <p>Rank is : {this.state.rank}</p>
                    <h2>This is the Footer Component</h2>
                </div>

            );
        }
    });
```



```
var MyComponent = React.createClass({
    render : function () {
        return(
        <div>
            <Header propsForPageTitle = 'AboutUs'/>

            <h1 style={style}>Hello world</h1>
            <h2 style={styleForHeading}>TOPS Technologies</h2>
            <h2 style={styleForHeading}>Sum of 2 + 2 is : {2+2}</h2>
            <Footer/>
        </div>
        )
    }
});

ReactDOM.render(
    <MyComponent/>,document.getElementById('root')
);
```

Class Component Lifecycle

We have seen so far that React web apps are actually a collection of independent components which run according to the interactions made with them. Every React Component has a lifecycle of its own, lifecycle of a component can be defined as the series of methods that are invoked in different stages of the component's existence. The definition is pretty straightforward but what do we mean by different stages? A React Component can go through four stages of its life as follows.

Class Component Lifecycle

Initialization: This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class.

Mounting: Mounting is the stage of rendering the JSX returned by the render method itself.

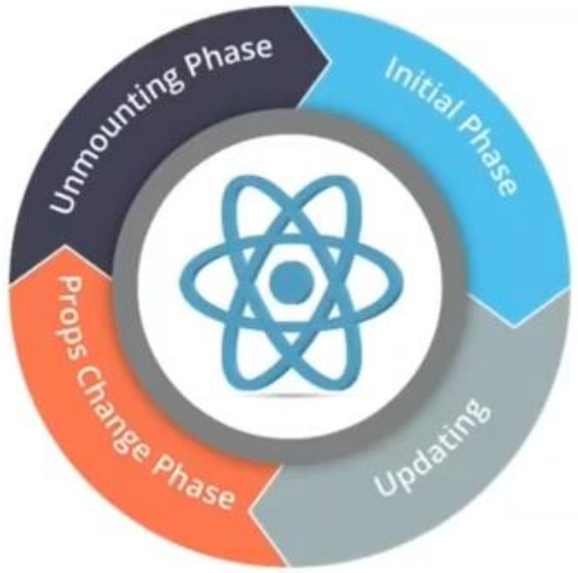
Updating: Updating is the stage when the state of a component is updated and the application is repainted.

Unmounting: As the name suggests Unmounting is the final step of the component lifecycle where the component is removed from the page.

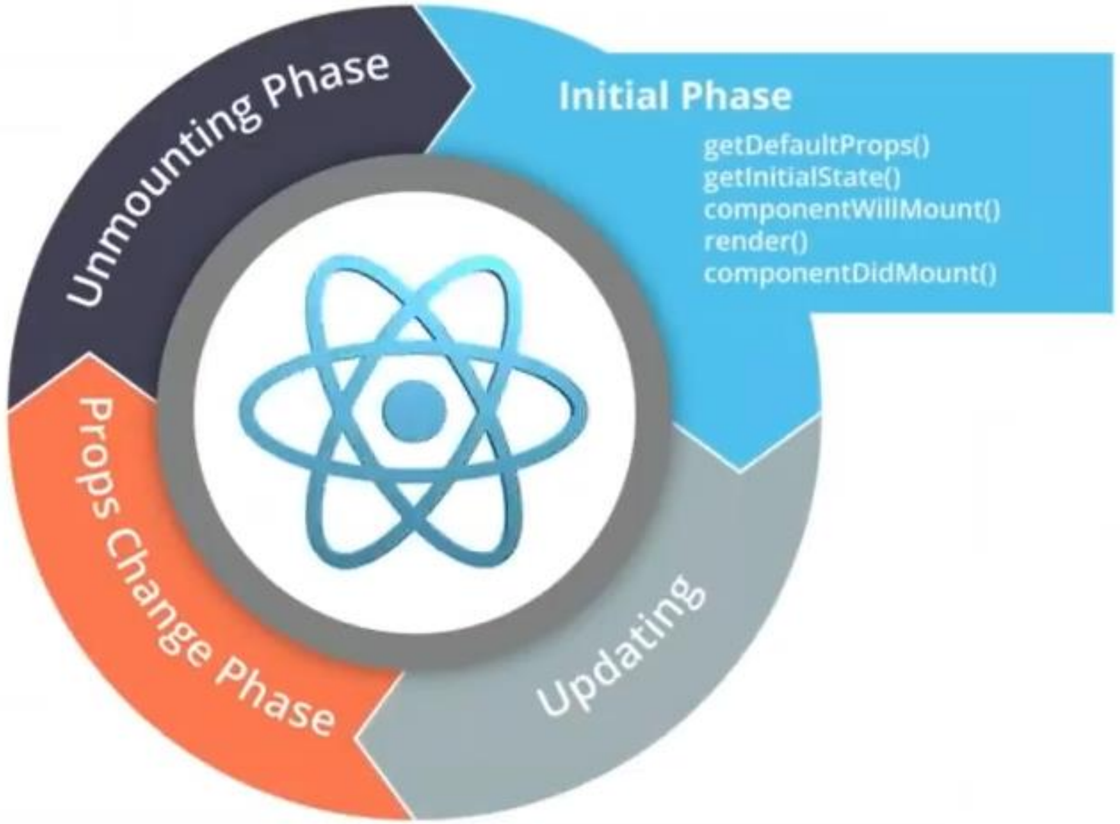
Class Component Lifecycle

React provides various methods which notifies when certain stage of the lifecycle occurs called **Lifecycle methods**

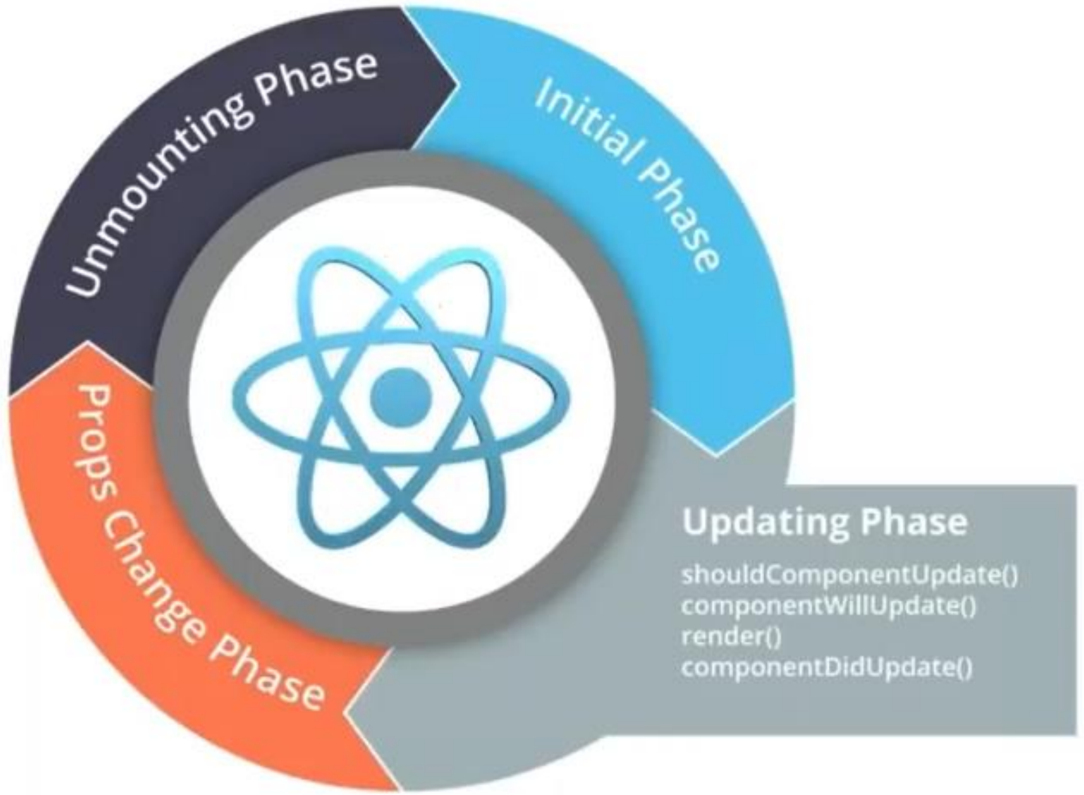
- ✓ These are special event handlers that are called at various points in components life
- ✓ Code can be added to them to perform various tasks
- ✓ They are invoked in a predictable order
- ✓ Entire lifecycle is divided into 4 phases.



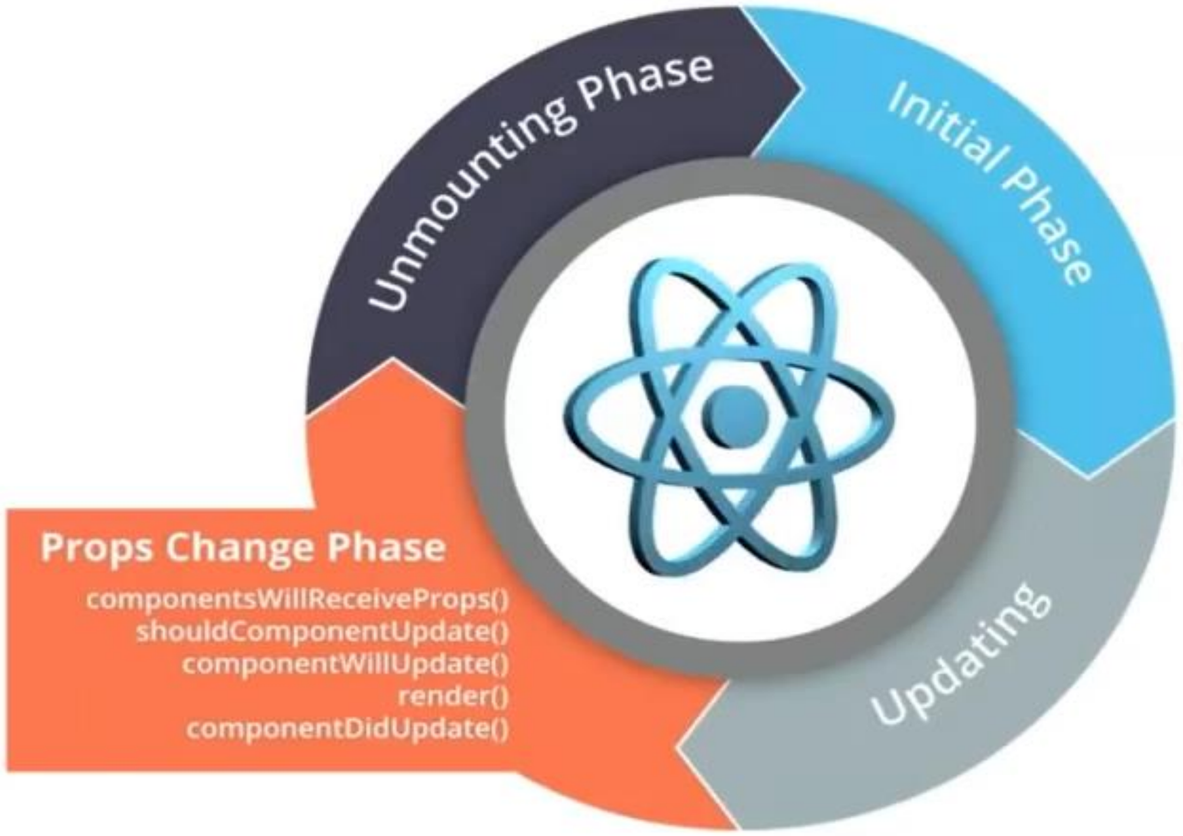
Initial Phase



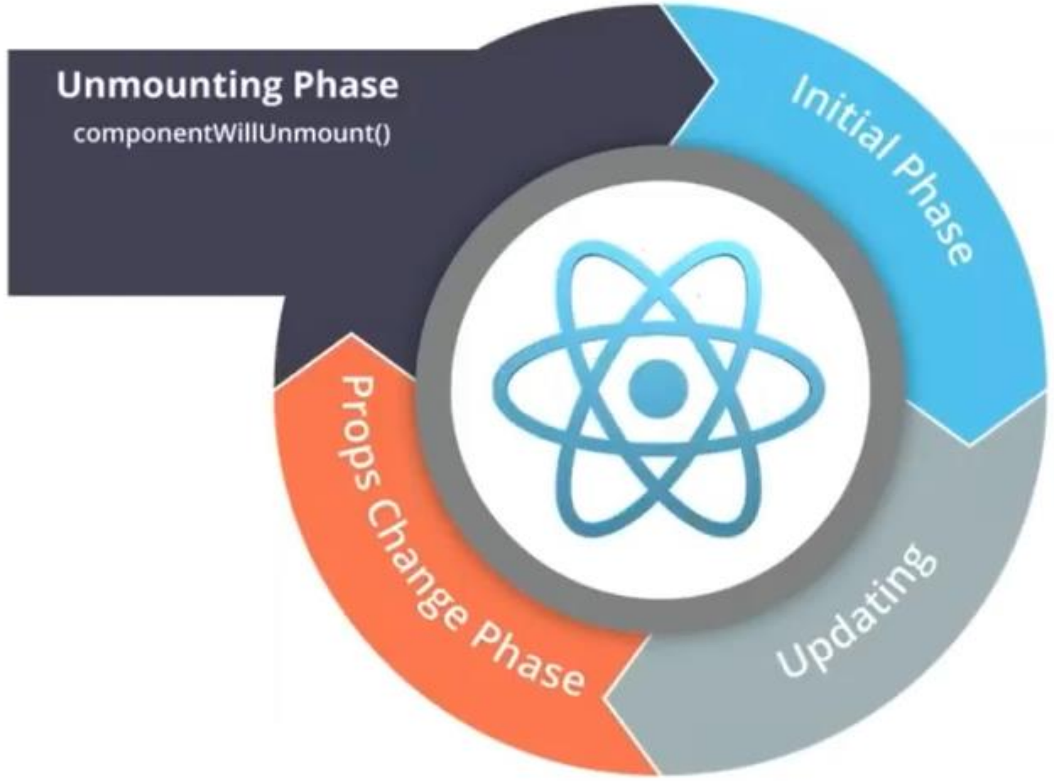
UpdatingPhase

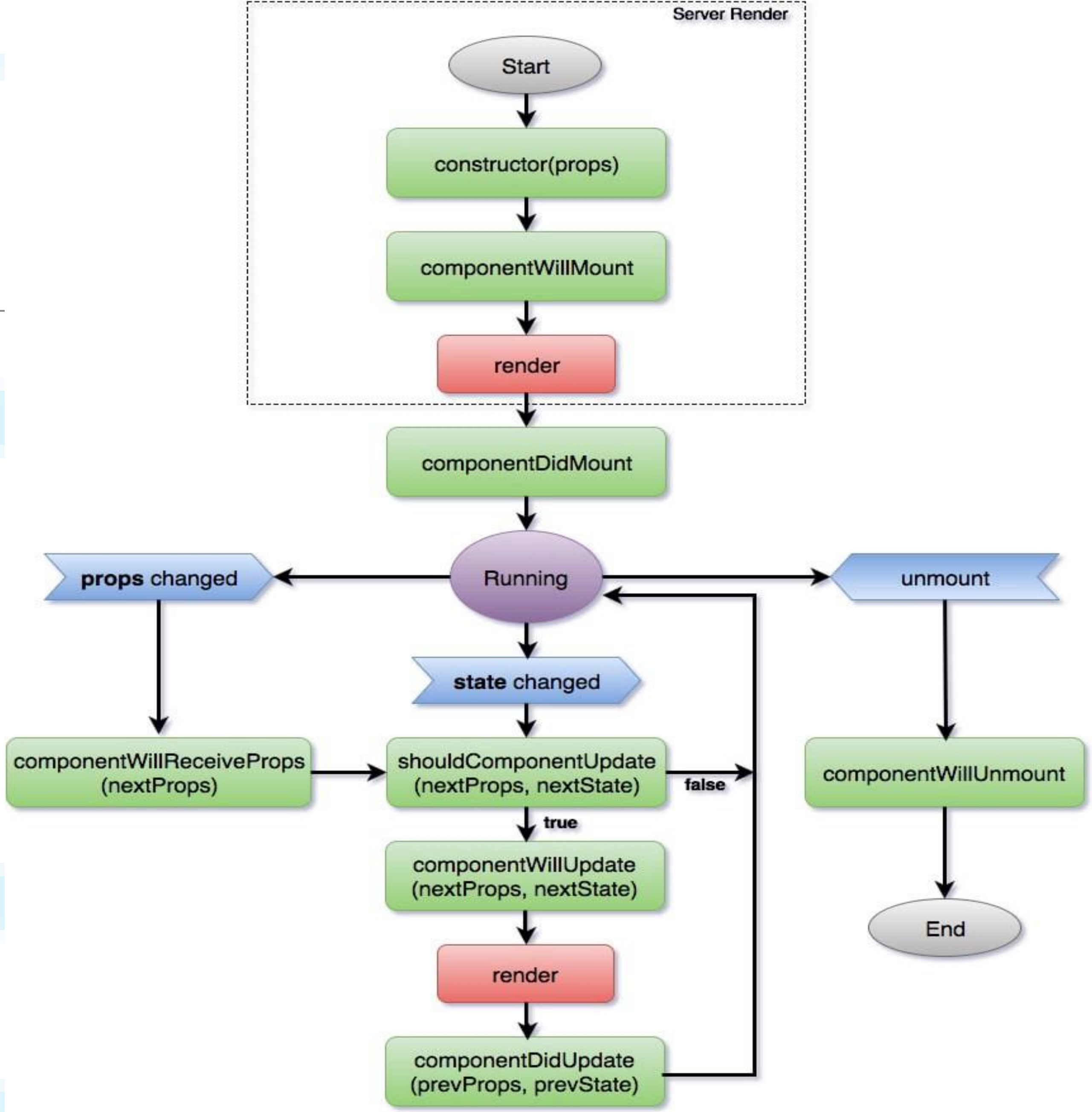


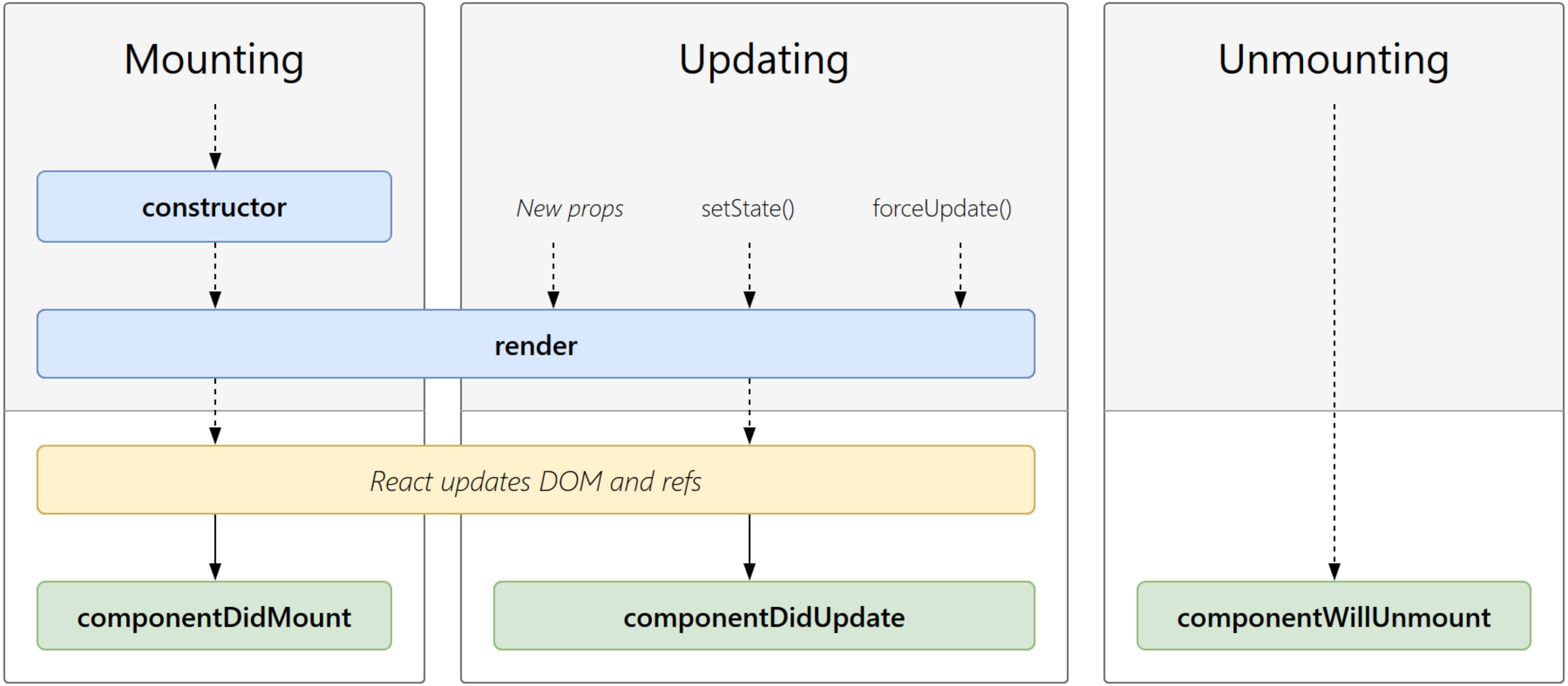
Props Change Phase



Unmourning Phase




Server Render
Start
constructor(props)
componentWillMount
render
componentDidMount
Running
props changed
unmount
state changed
componentWillReceiveProps (nextProps)
shouldComponentUpdate (nextProps, nextState)
false
true
componentWillUnmount
componentWillUpdate (nextProps, nextState)
render
End
componentDidUpdate (prevProps, prevState)


Mounting
Updating
Unmounting
constructor
New props
setState()
forceUpdate()
render
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

Module 3

LIST & HOOKS

List & Hooks

List

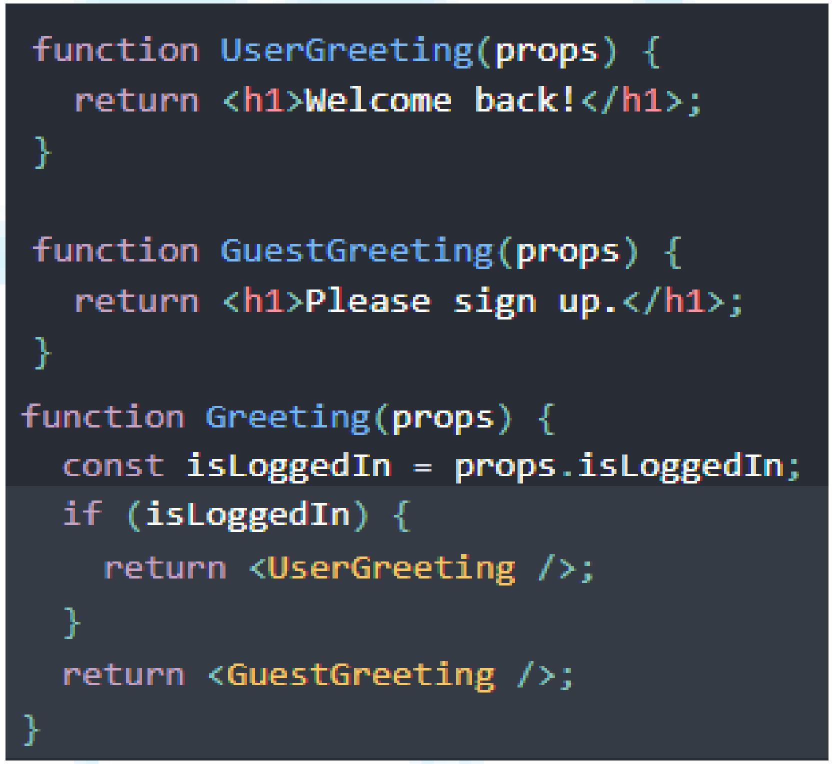
- - Conditional Rendering
- - Lists and Keys
- - Forms
- - Handling Events
- - Lifting State up

Hooks

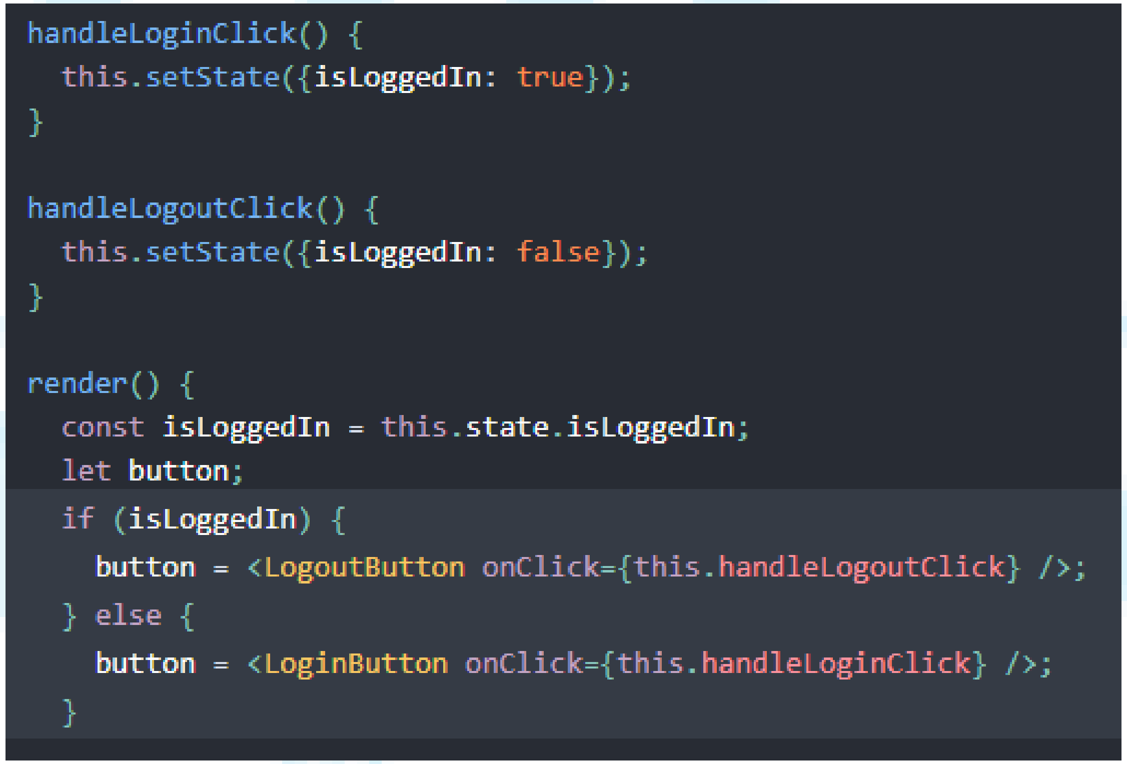
- - Introduction
- - Using the State hook
- - Using the Effect hook
- - Rules of Hook
- - Custom Hook

Conditional Rendering

Conditional rendering in React works the same way conditions work in JavaScript

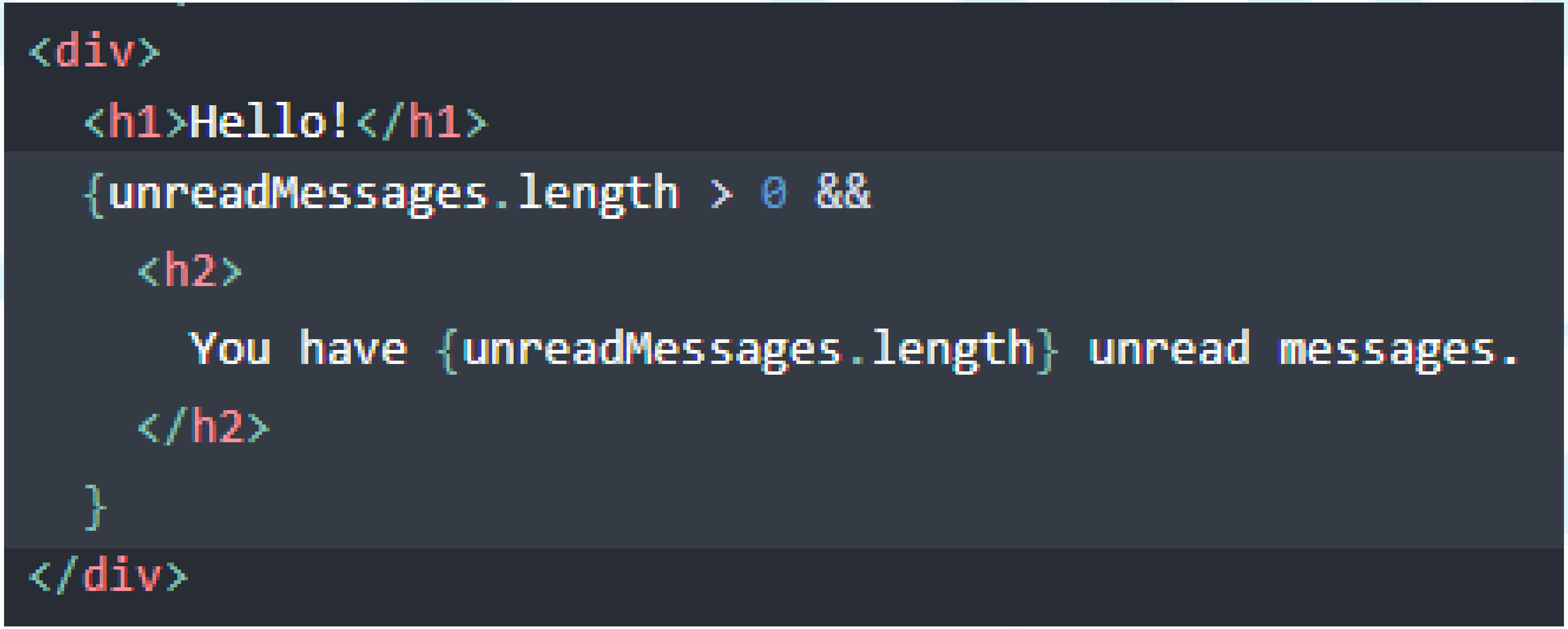


```
function UserGreeting(props) {
  return <h1>Welcome back!</h1>;
}

function GuestGreeting(props) {
  return <h1>Please sign up.</h1>;
}
function Greeting(props) {
  const isLoggedIn = props.isLoggedIn;
  if (isLoggedIn) {
    return <UserGreeting />;
  }
  return <GuestGreeting />;
}
```



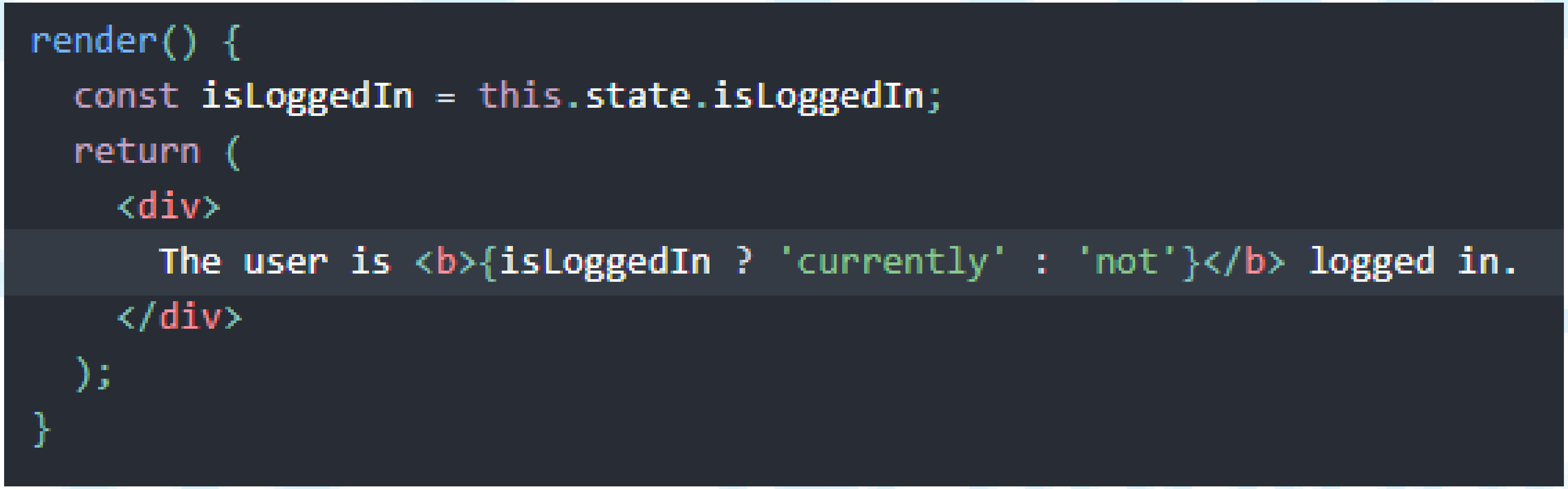
```
handleLoginClick() {
  this.setState({isLoggedIn: true});
}

handleLogoutClick() {
  this.setState({isLoggedIn: false});
}

render() {
  const isLoggedIn = this.state.isLoggedIn;
  let button;
  if (isLoggedIn) {
    button = <LogoutButton onClick={this.handleLogoutClick} />;
  } else {
    button = <LoginButton onClick={this.handleLoginClick} />;
  }
```

Inline if with logical && operator



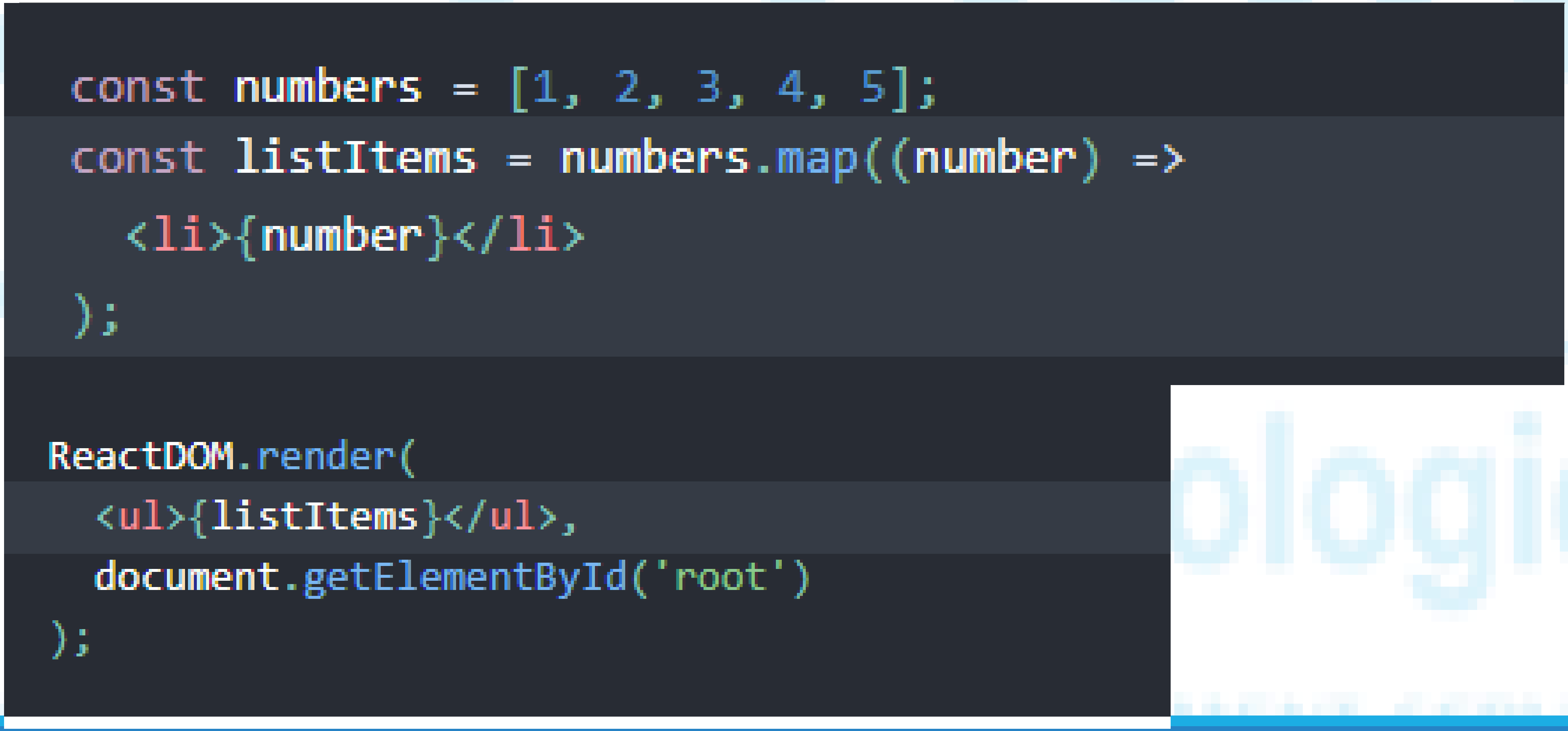
```
<div>
  <h1>Hello!</h1>
  {unreadMessages.length > 0 &&
    <h2>
      You have {unreadMessages.length} unread messages.
    </h2>
  }
</div>
```

Inline if...else with conditional operator



```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

List : Rendering Multiple components



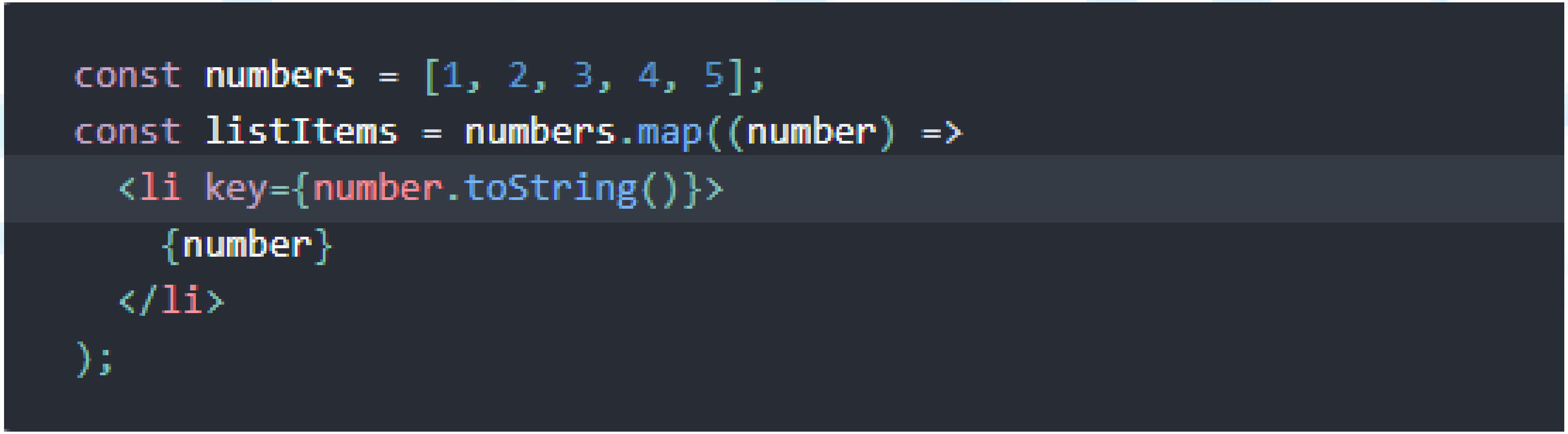
```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li>{number}</li>
);
```

```
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```

Keys

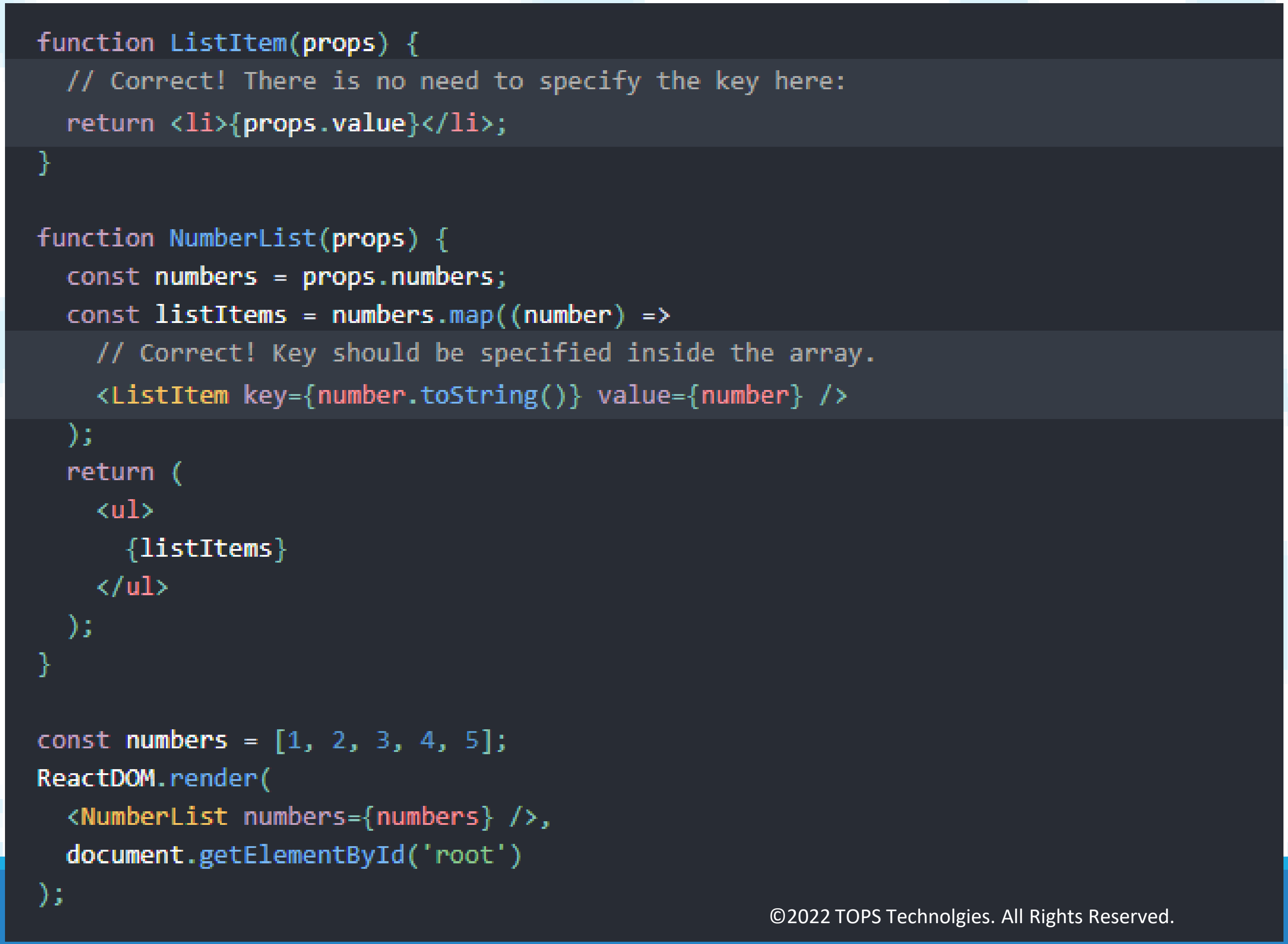
Keys help React identify which items have changed, are added, or are removed.

Keys Must Only Be Unique Among Siblings



```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
  </li>
);
```

Extracting Components with Key



```
function ListItem(props) {
  // Correct! There is no need to specify the key here:
  return <li>{props.value}</li>;
}

function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    // Correct! Key should be specified inside the array.
    <ListItem key={number.toString()} value={number} />
  );
  return (
    <ul>
      {listItems}
    </ul>
  );
}

const numbers = [1, 2, 3, 4, 5];
ReactDOM.render(
  <NumberList numbers={numbers} />,
  document.getElementById('root')
);
```

Handling Events

Handling events with React elements is very similar to handling events on DOM elements.

- React events are named using camelCase, rather than lowercase.
- With JSX you pass a function as the event handler, rather than a string.

```
<button onclick="activateLasers()">
  Activate Lasers
</button>
```

React

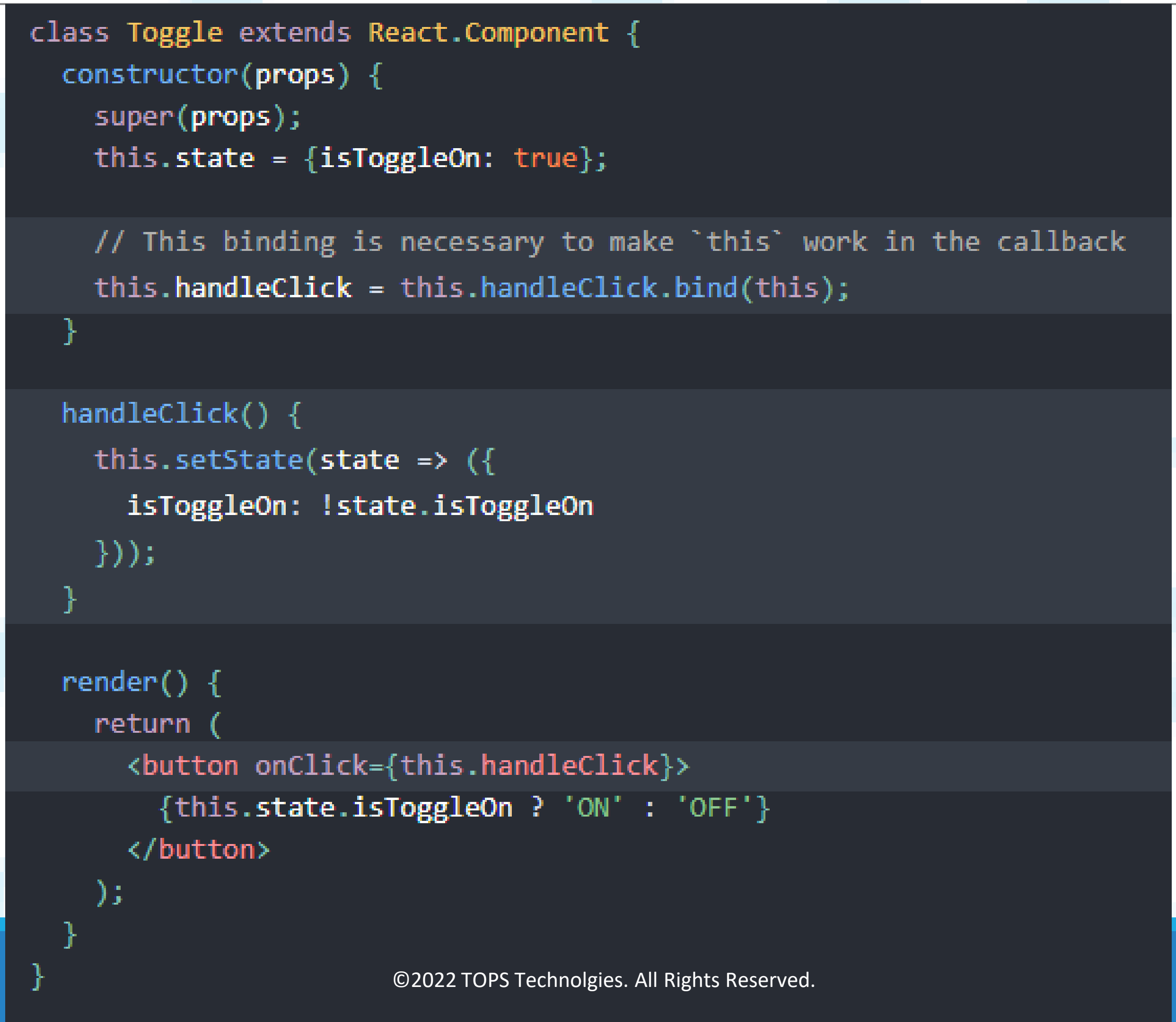
```
<button onClick={activateLasers}>
  Activate Lasers
</button>
```

```
a href="#" onclick="console.log('The link was clicked.'); return false">
 Click me
/a>
```

HTML

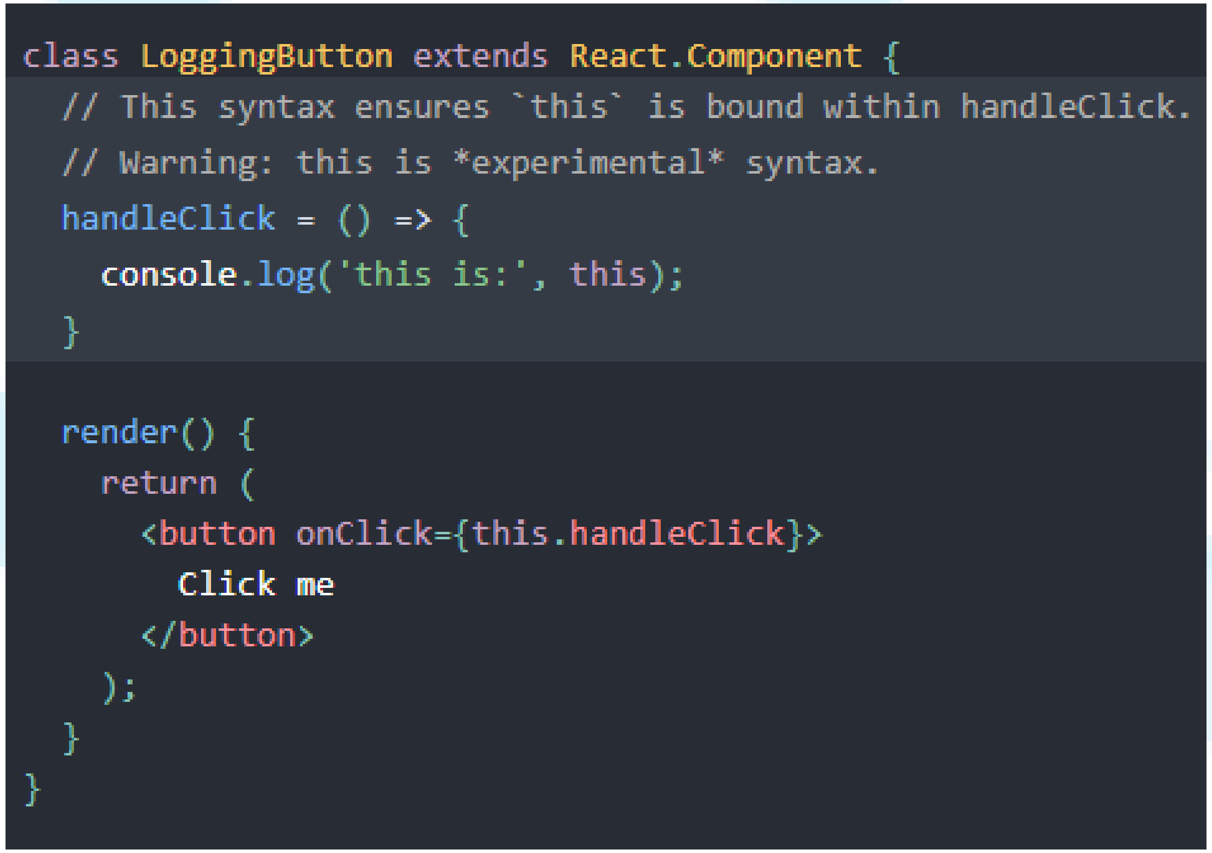
```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

Event Handling in ES6 Class



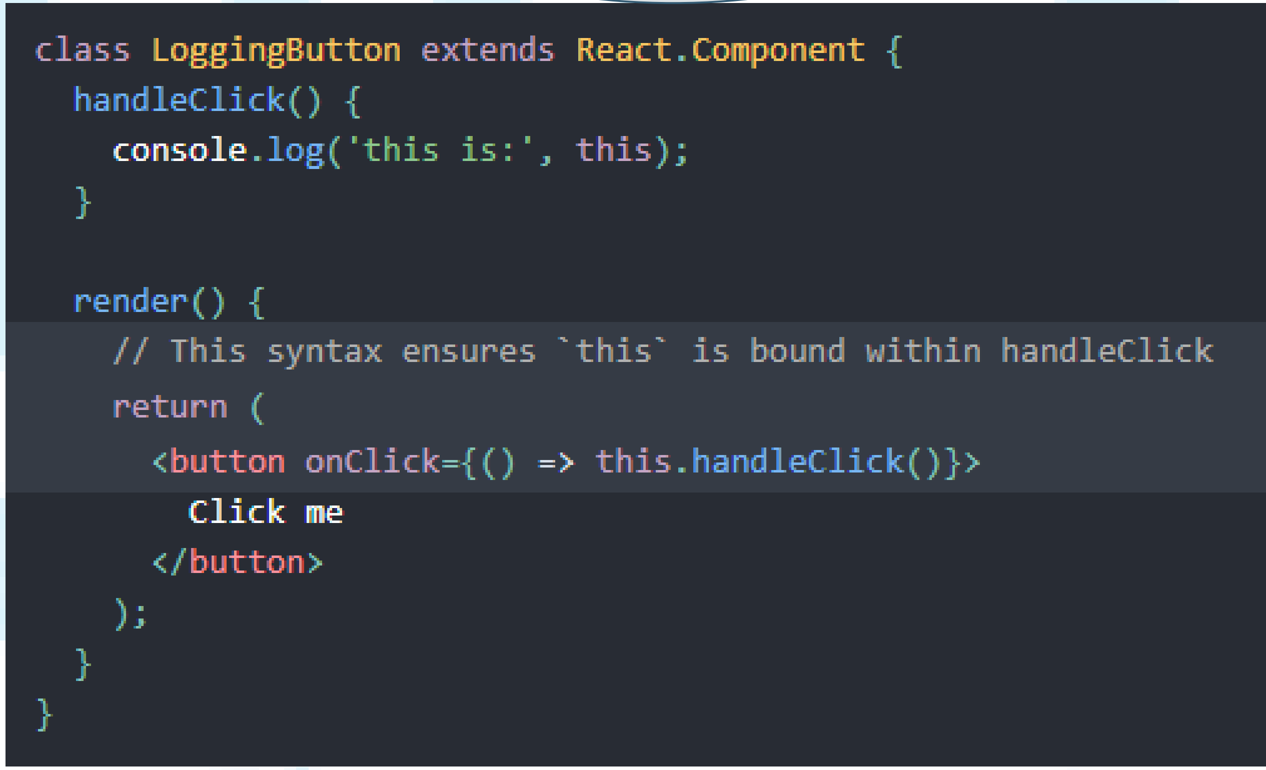
```
class Toggle extends React.Component {
  constructor(props) {
    super(props);
    this.state = {isToggleOn: true};

    // This binding is necessary to make `this` work in the callback
    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    this.setState(state => ({
      isToggleOn: !state.isToggleOn
    }));
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        {this.state.isToggleOn ? 'ON' : 'OFF'}
      </button>
    );
  }
}
```

Ignore bind method



```
class LoggingButton extends React.Component {
  // This syntax ensures `this` is bound within handleClick.
  // Warning: this is *experimental* syntax.
  handleClick = () => {
    console.log('this is:', this);
  }

  render() {
    return (
      <button onClick={this.handleClick}>
        Click me
      </button>
    );
  }
}
```

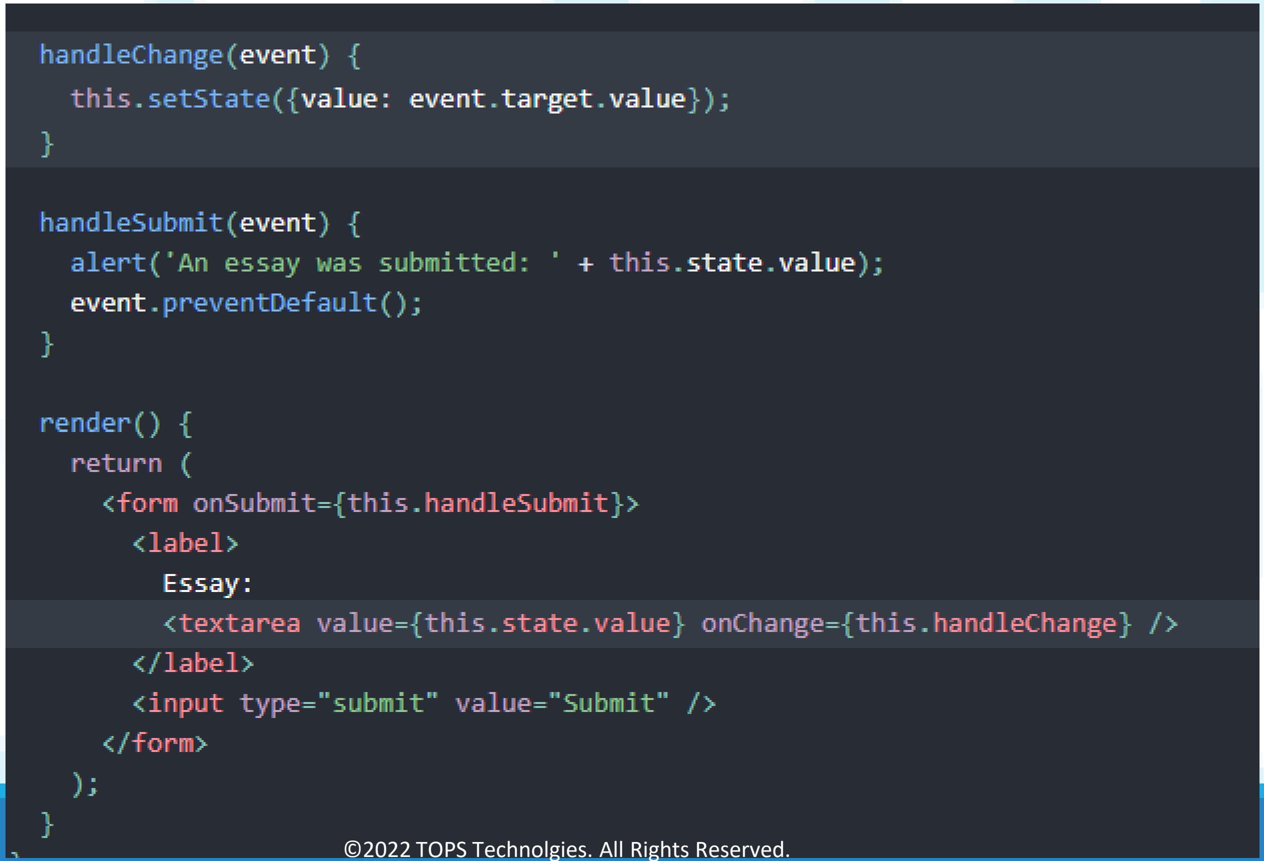
OR



```
class LoggingButton extends React.Component {
  handleClick() {
    console.log('this is:', this);
  }

  render() {
    // This syntax ensures `this` is bound within handleClick
    return (
      <button onClick={() => this.handleClick()}>
        Click me
      </button>
    );
  }
}
```

Forms

HTML form elements work a little bit differently from other DOM elements in React, because form elements naturally keep some internal state.



```
handleChange(event) {
  this.setState({value: event.target.value});
}

handleSubmit(event) {
  alert('An essay was submitted: ' + this.state.value);
  event.preventDefault();
}

render() {
  return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Essay:
        <textarea value={this.state.value} onChange={this.handleChange} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  );
}
```

Hooks - Introduction

- Hooks are a new feature addition in React version 16.8 which allow you to use React features without having to write a class
- Ex : State of component
- Hooks don’t work inside classes

Why Hooks?

- Understand how **this** keyword works in JavaScript
- Remember to bind event handlers in class components
- Classes don’t minify very well and make hot reloading very unreliable

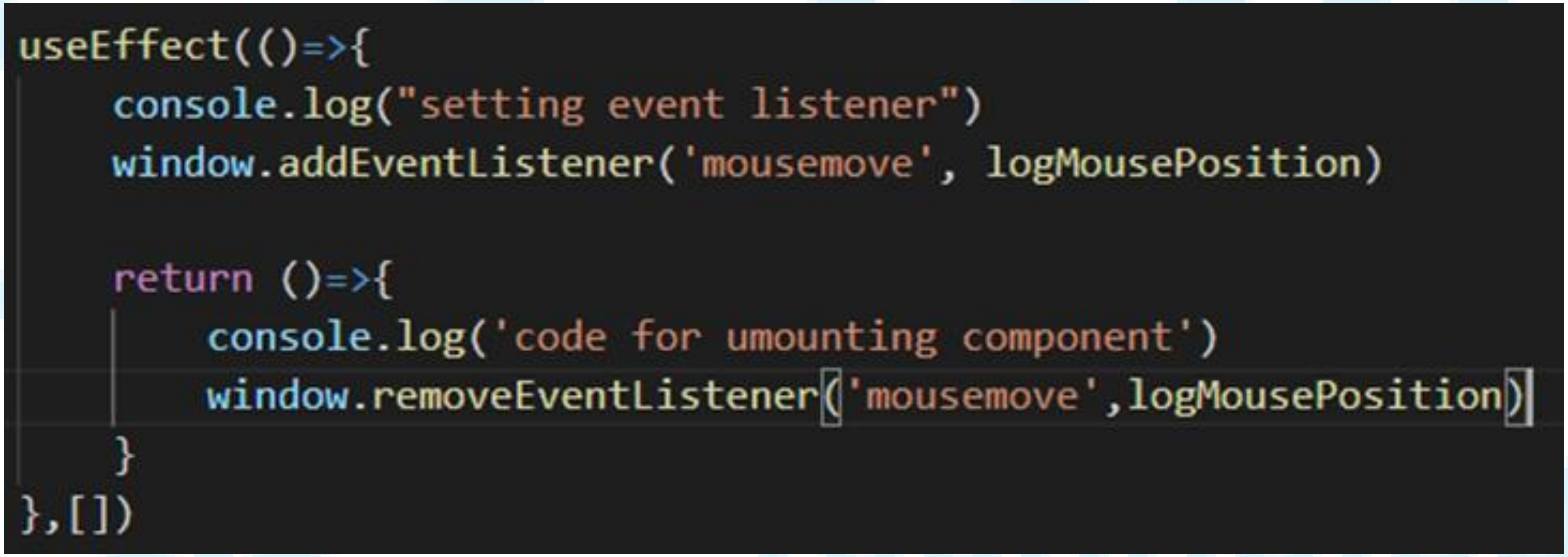
Why Hooks?

- Create components for complex scenarios such as data fetching and subscribing to events
- Related code is not organized in one place
- Ex : Data fetching – In componentDidMount and componentDidUpdate
- Ex : Event Listeners – In componentDidMount and componentWillUnmount
- Because of stateful logic – Cannot break components into smaller ones

useState hook

- useState with Previous state Example HookCounterTwo
-
- useState with object Example HookCounterThree
-
- useState with Array Example HookCounterFour

useEffect Hook



```
useEffect(()=>{
    console.log("setting event listener")
    window.addEventListener('mousemove', logMousePosition)

    return ()=>{
        console.log('code for umounting component')
        window.removeEventListener('mousemove',logMousePosition)
    }
},[])
```

Fetch Data using useEffect

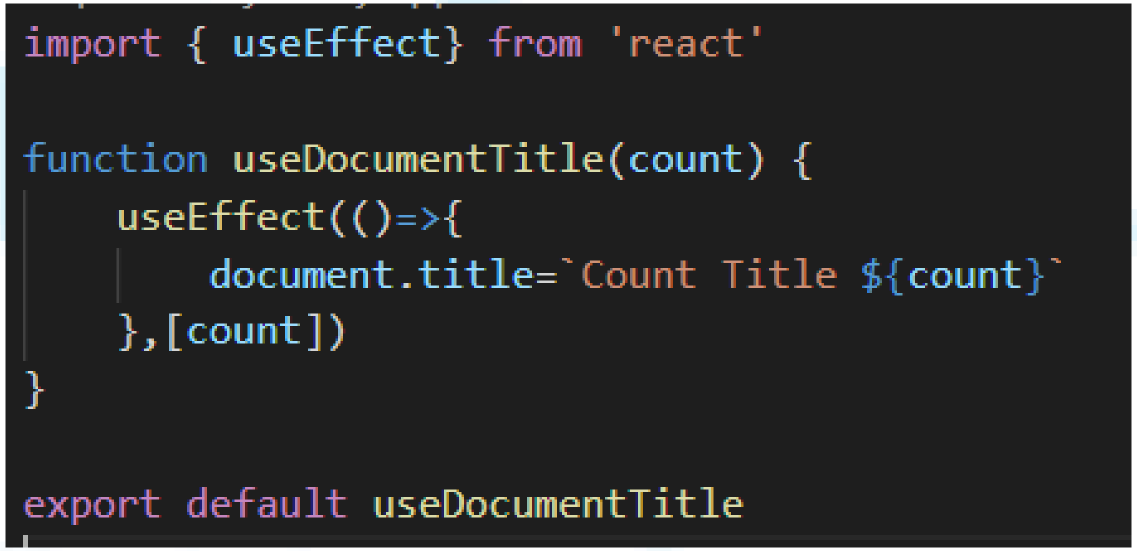
- JsonPlaceholder for API
- Install npm axios for API call
-
- Example fetch all posts
- Example fetch single post input id by user
- Example fetch single post by button click

Rules of Hook

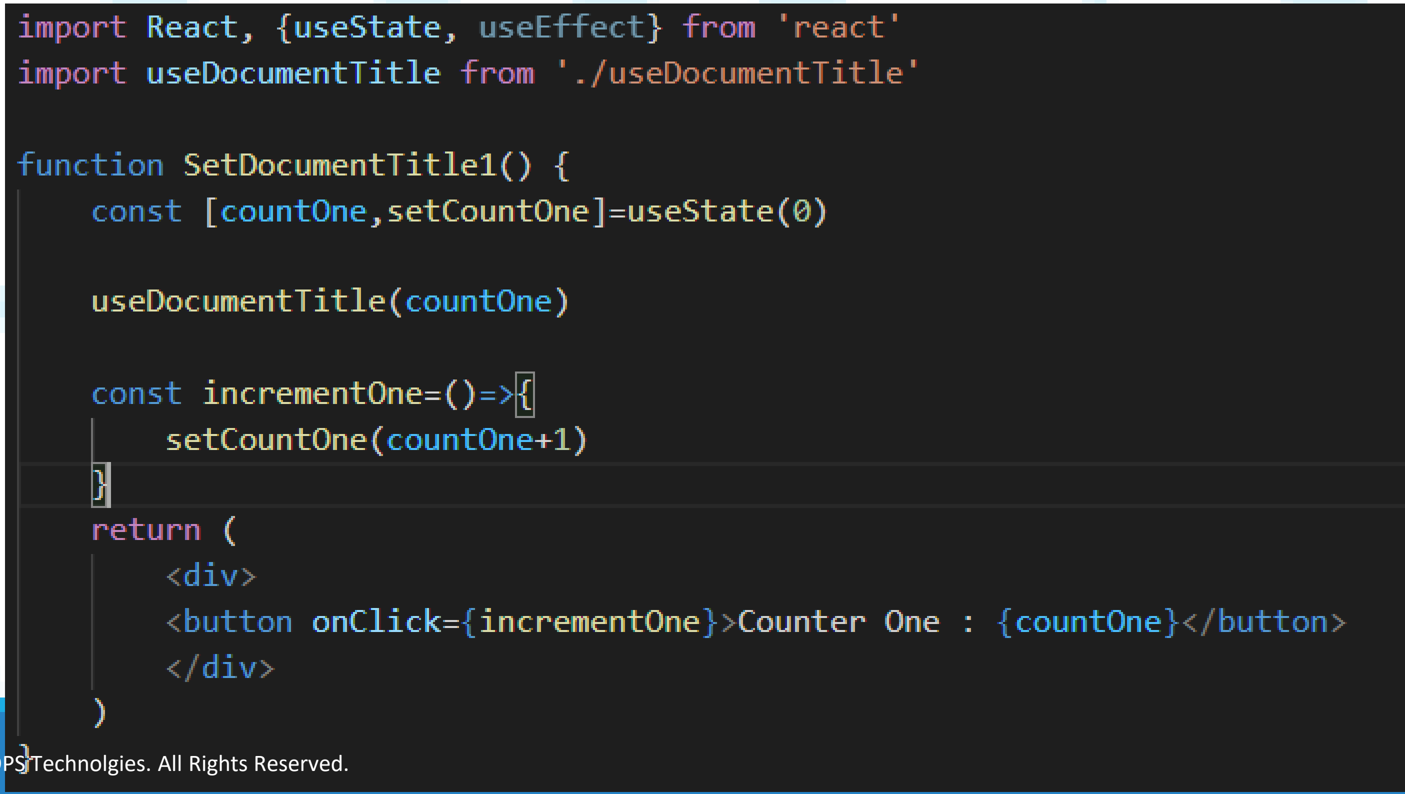
- **“Only call Hooks at the Top Level”**
- Don’t call Hooks inside loops, conditions or nested functions
-
- **“Only call Hooks from React Functions”**
- Call them from within React functional components and not just any regular JavaScript function

Custom Hook

- A custom Hook is basically a JavaScript function whose name starts with “use”
- A custom Hook can also call other Hooks if required



```
import { useEffect} from 'react'

function useDocumentTitle(count) {
    useEffect(()=>{
        document.title=`Count Title ${count}`
    },[count])
}

export default useDocumentTitle
```



```
import React, {useState, useEffect} from 'react'
import useDocumentTitle from './useDocumentTitle'

function SetDocumentTitle1() {
    const [countOne,setCountOne]=useState(0)

    useDocumentTitle(countOne)

    const incrementOne=()=>{
        setCountOne(countOne+1)
    }
    return (
        <div>
        <button onClick={incrementOne}>Counter One : {countOne}</button>
        </div>
    )
}
```

Module 4

STYLING & ADVANCE REACT

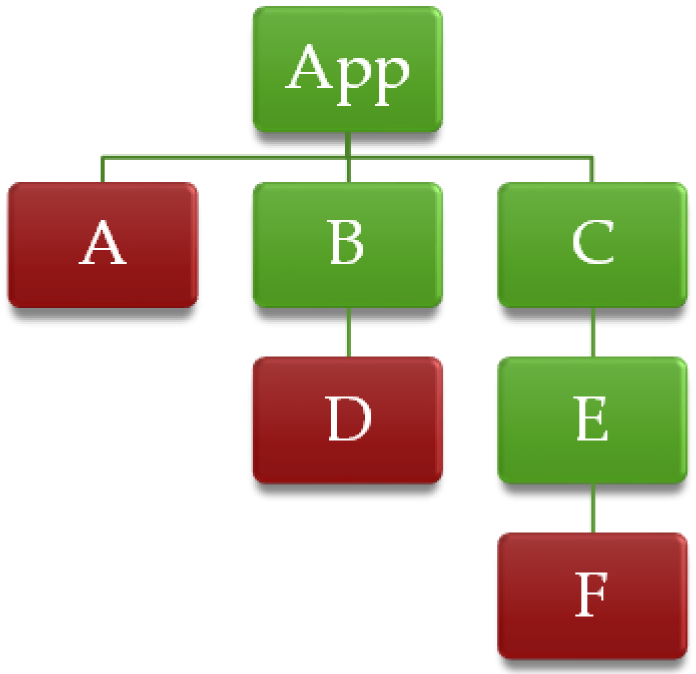
Styling & Advance React

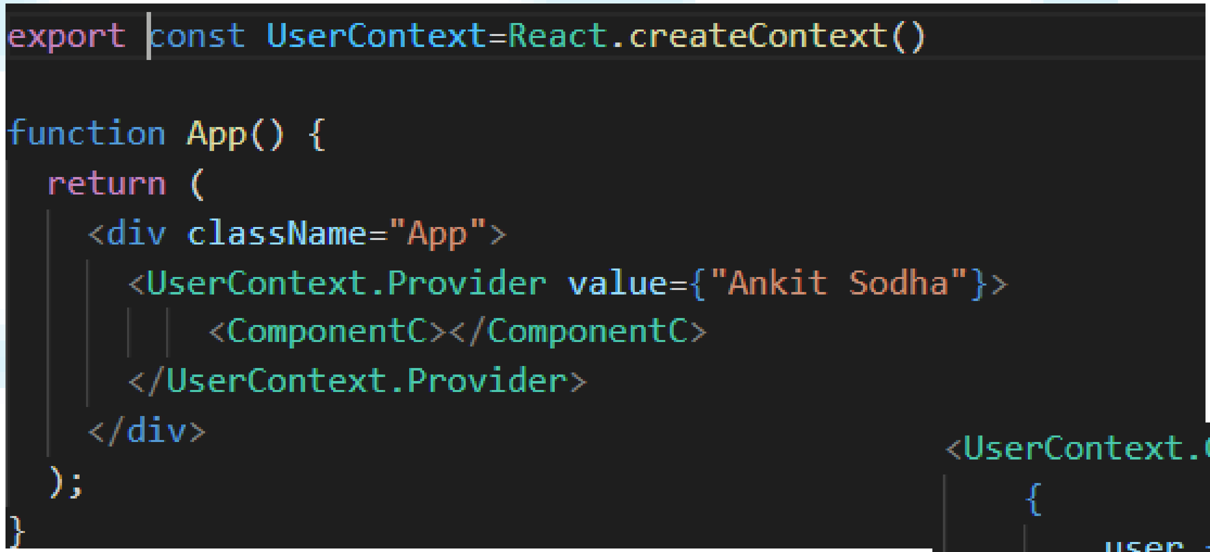
Advance Concepts

- - Context, useContext()
- - Working with Refs and useRefs()
- - Fragments
- - Performance optimization with useMemo()
- - Styling React Components
- - CSS stylesheet
- - Inline Styling
- - CSS Modules
- - CSS in JS Libraries (styled components)
- - Bootstrap with React

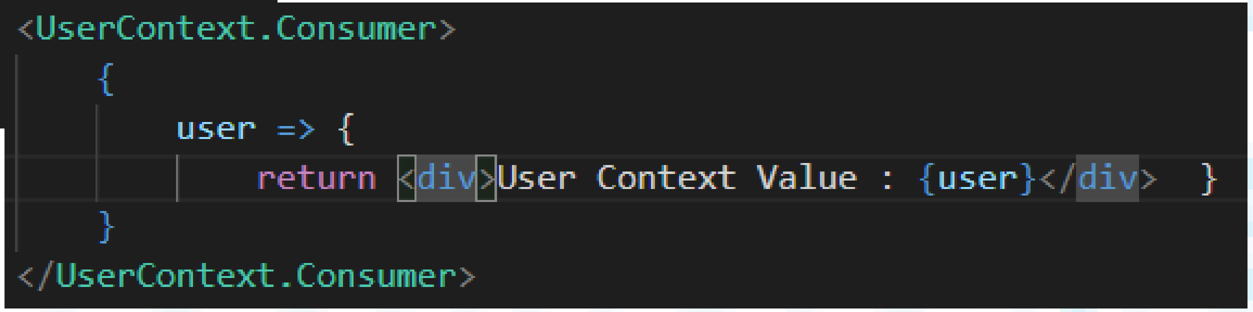
Context

Context provides a way to pass data through the component tree without having to pass props down manually at every level.



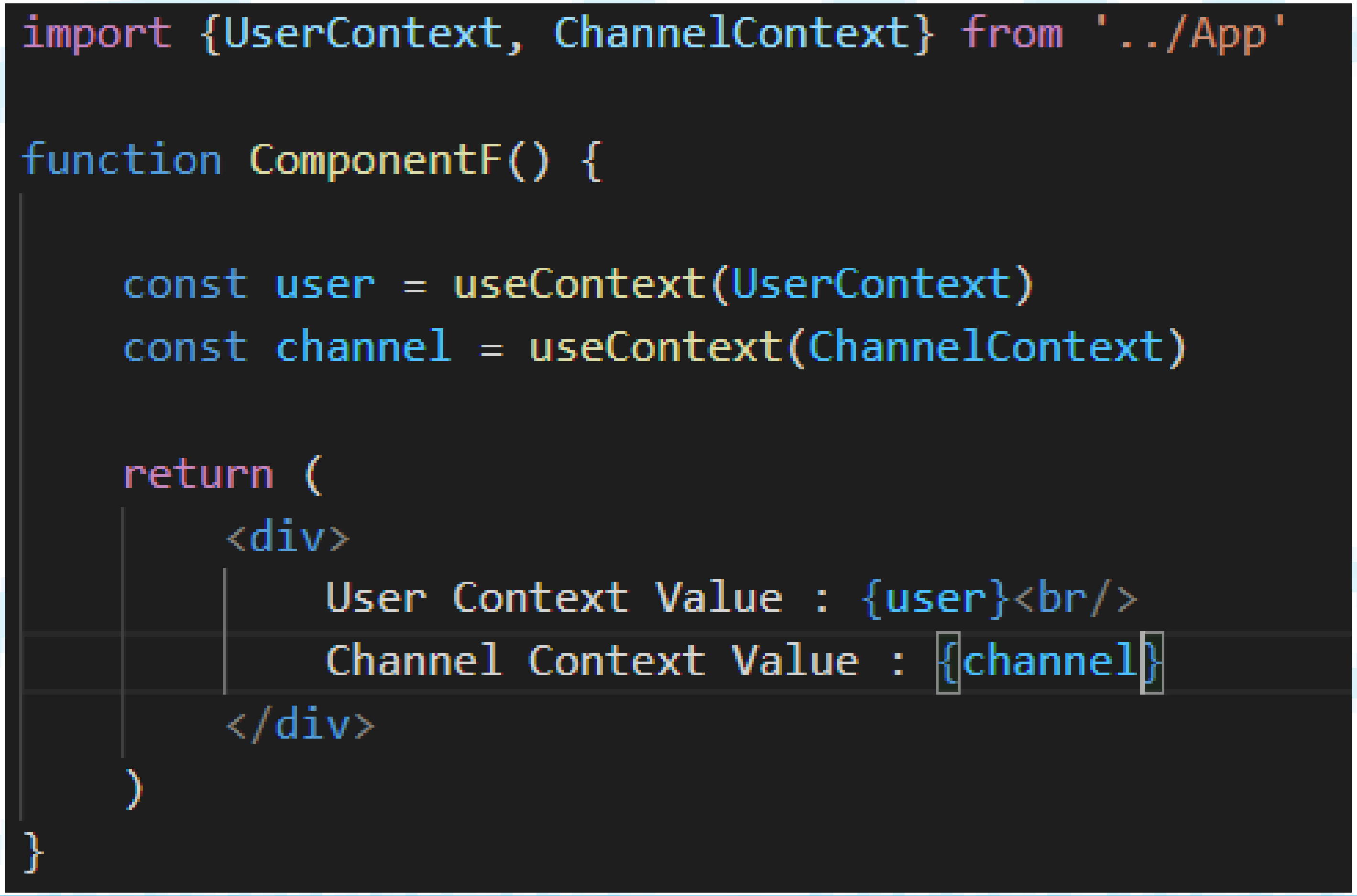


```
export const UserContext=React.createContext()

function App() {
  return (
    <div className="App">
      <UserContext.Provider value={"Ankit Sodha"}>
          <ComponentC></ComponentC>
      </UserContext.Provider>
    </div>
  );
}
```

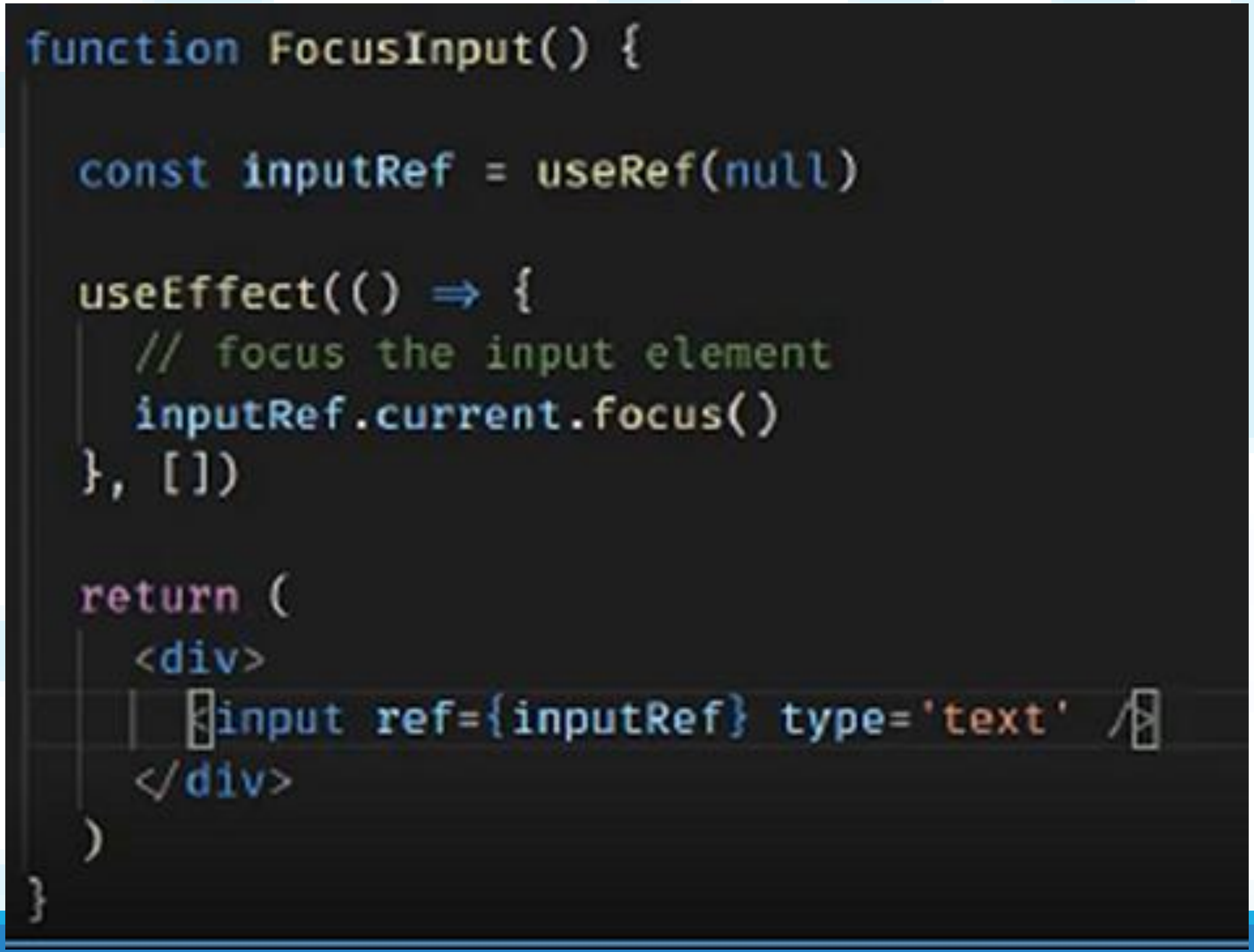


```
<UserContext.Consumer>
    {
        user => {
            return <div>User Context Value : {user}</div>  }
    }
</UserContext.Consumer>
```

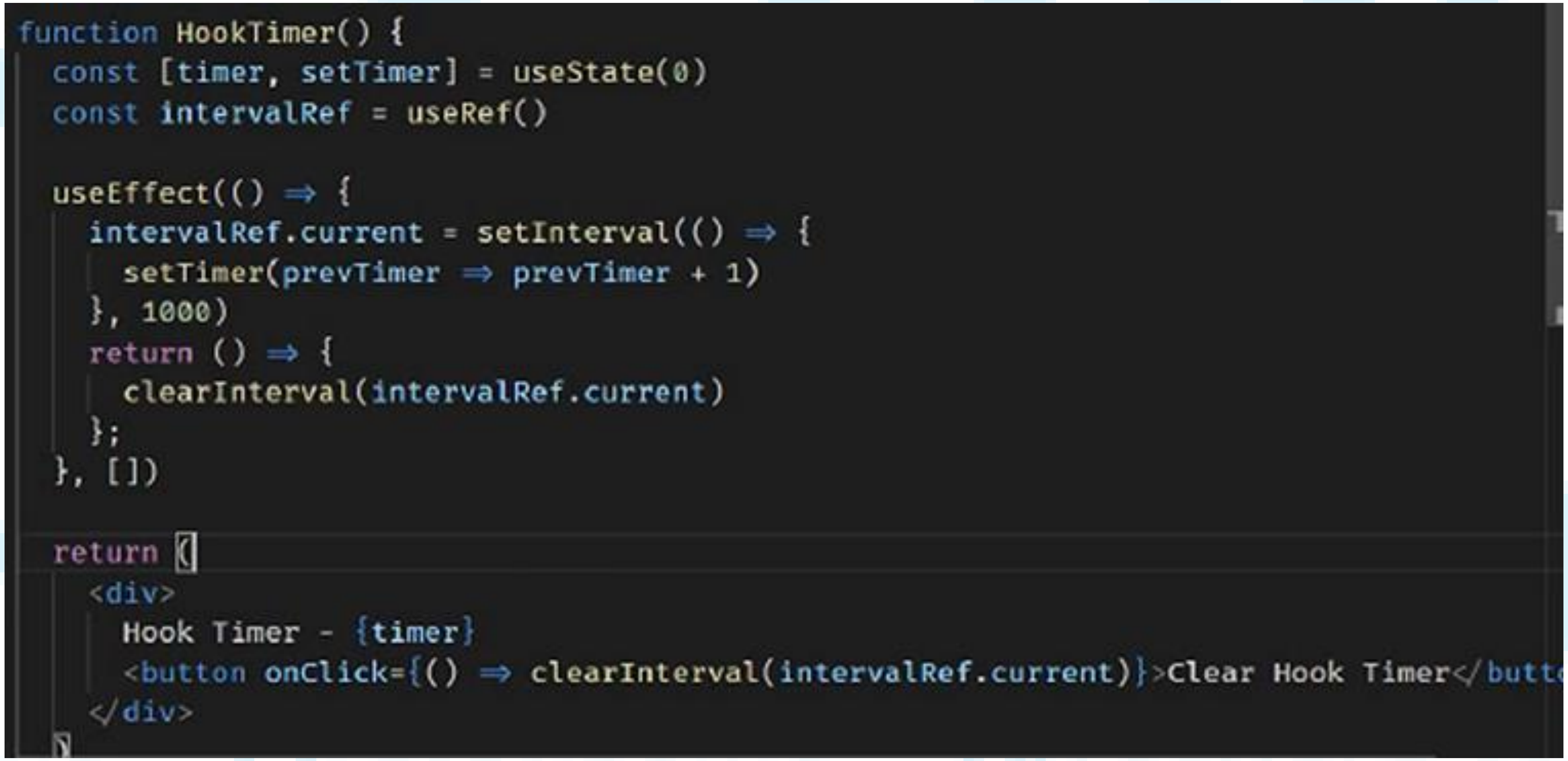
useContext


```
import {UserContext, ChannelContext} from '../App'

function ComponentF() {

    const user = useContext(UserContext)
    const channel = useContext(ChannelContext)

    return (
        <div>
            User Context Value : {user}<br/>
            Channel Context Value : {channel}
        </div>
    )
}
```

useRefs Hook



```
function FocusInput() {

  const inputRef = useRef(null)

  useEffect(() => {
    // focus the input element
    inputRef.current.focus()
  }, [])

  return (
    <div>
      <input ref={inputRef} type='text' />
    </div>
  )
}
```

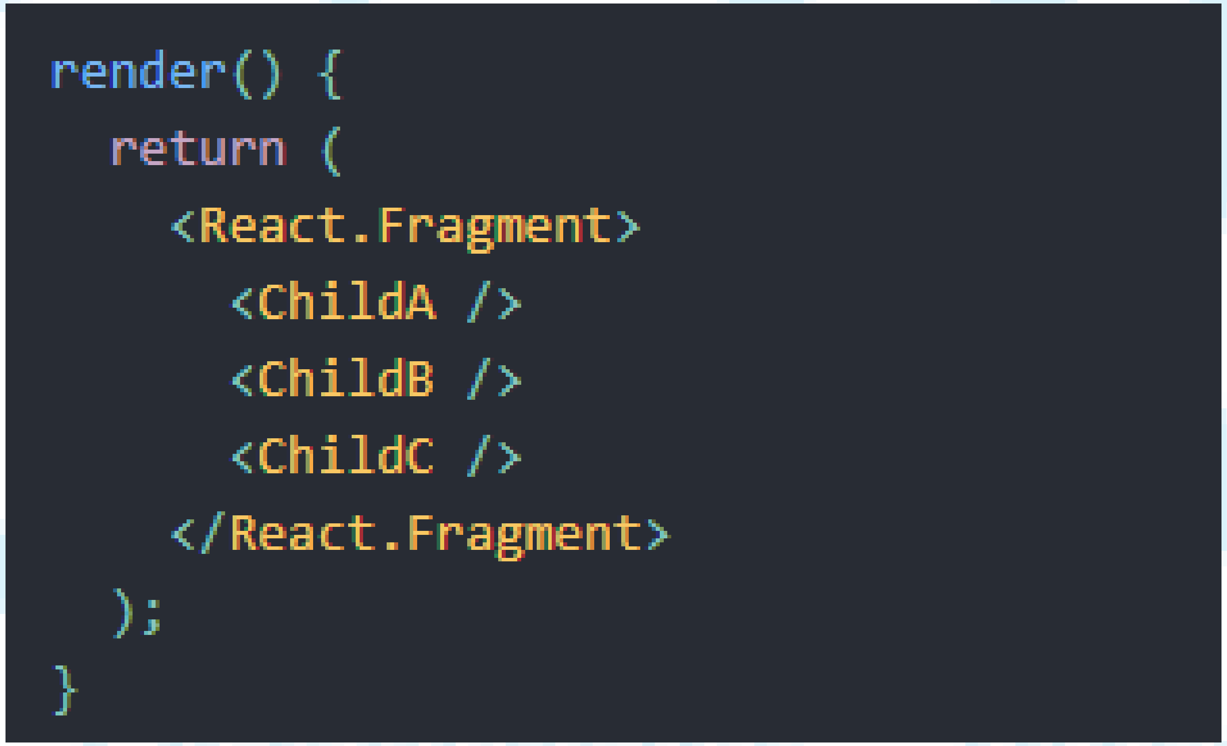


```
function HookTimer() {
  const [timer, setTimer] = useState(0)
  const intervalRef = useRef()

  useEffect(() => {
    intervalRef.current = setInterval(() => {
      setTimer(prevTimer => prevTimer + 1)
    }, 1000)
    return () => {
      clearInterval(intervalRef.current)
    };
  }, [])

  return (
    <div>
      Hook Timer - {timer}
      <button onClick={() => clearInterval(intervalRef.current)}>Clear Hook Timer</butt
    </div>
  )
```

Fragments

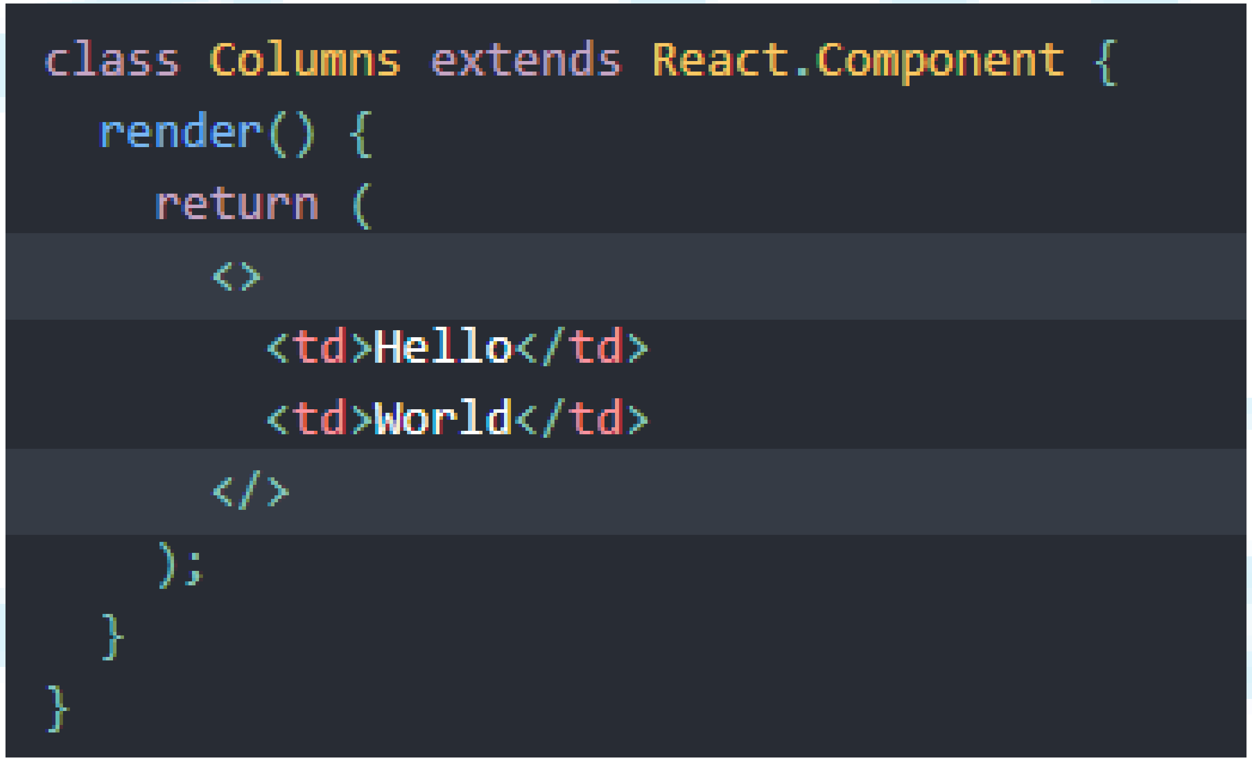
A common pattern in React is for a component to return multiple elements.

Fragments let you group a list of children without adding extra nodes to the DOM.



```
render() {
  return (
    <React.Fragment>
      <ChildA />
      <ChildB />
      <ChildC />
    </React.Fragment>
  );
}
```

Short Syntax



```
class Columns extends React.Component {
  render() {
    return (
      <>
        <td>Hello</td>
        <td>World</td>
      </>
    );
  }
}
```

useMemo Hook

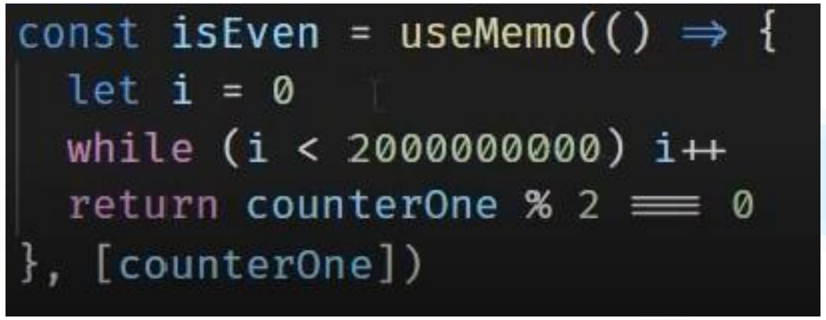
CounterOne – Button click increment

CounterTwo – Button click increment

Check Even – Odd CounterOne

Delay with loop while checking

Un-necessary effect with Counter Two also



```
const isEven = useMemo(() => {
  let i = 0
  while (i < 2000000000) i++
  return counterOne % 2 === 0
}, [counterOne])
```

Styling React Components

CSS stylesheets

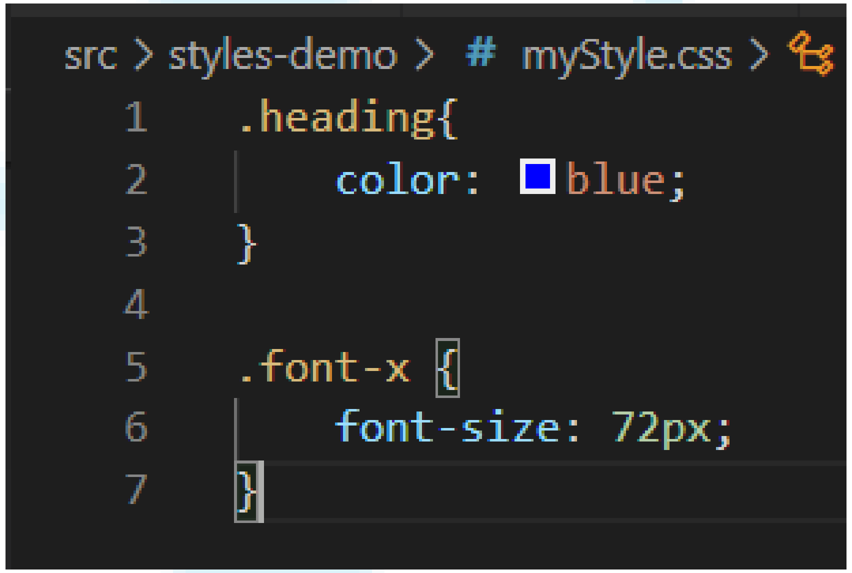
Inline Styling

CSS Modules

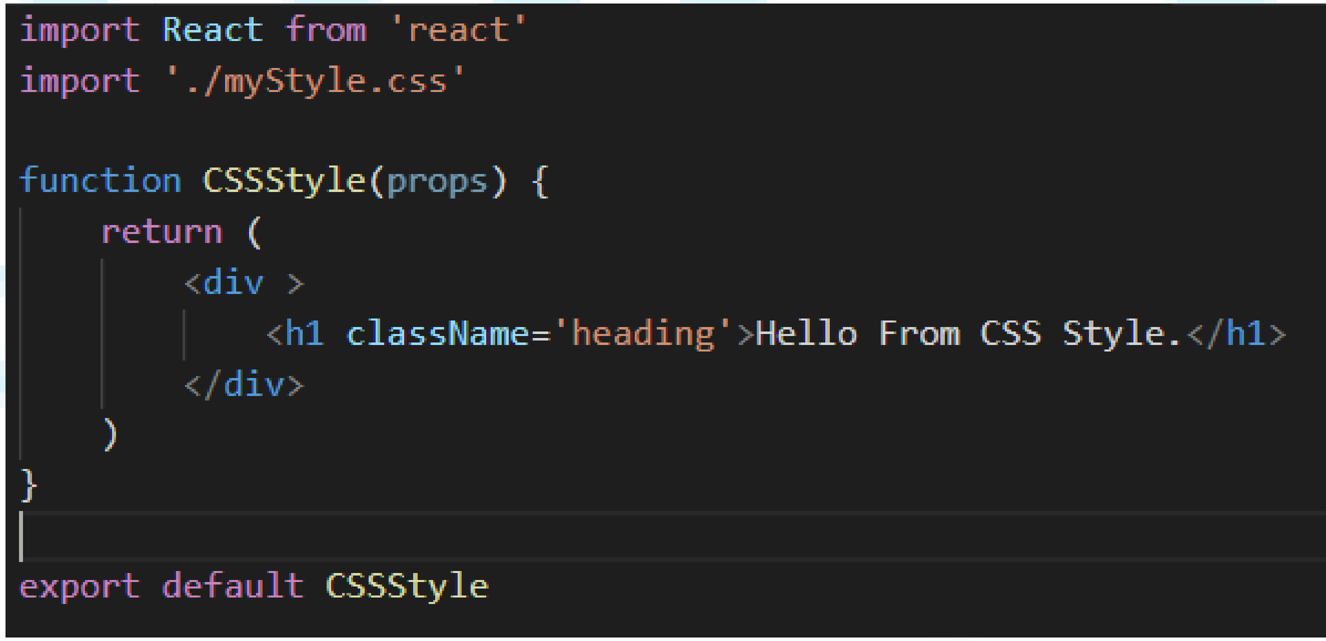
CSS in JS Libraries (styled components)

CSS Stylesheets

CSS stylesheets

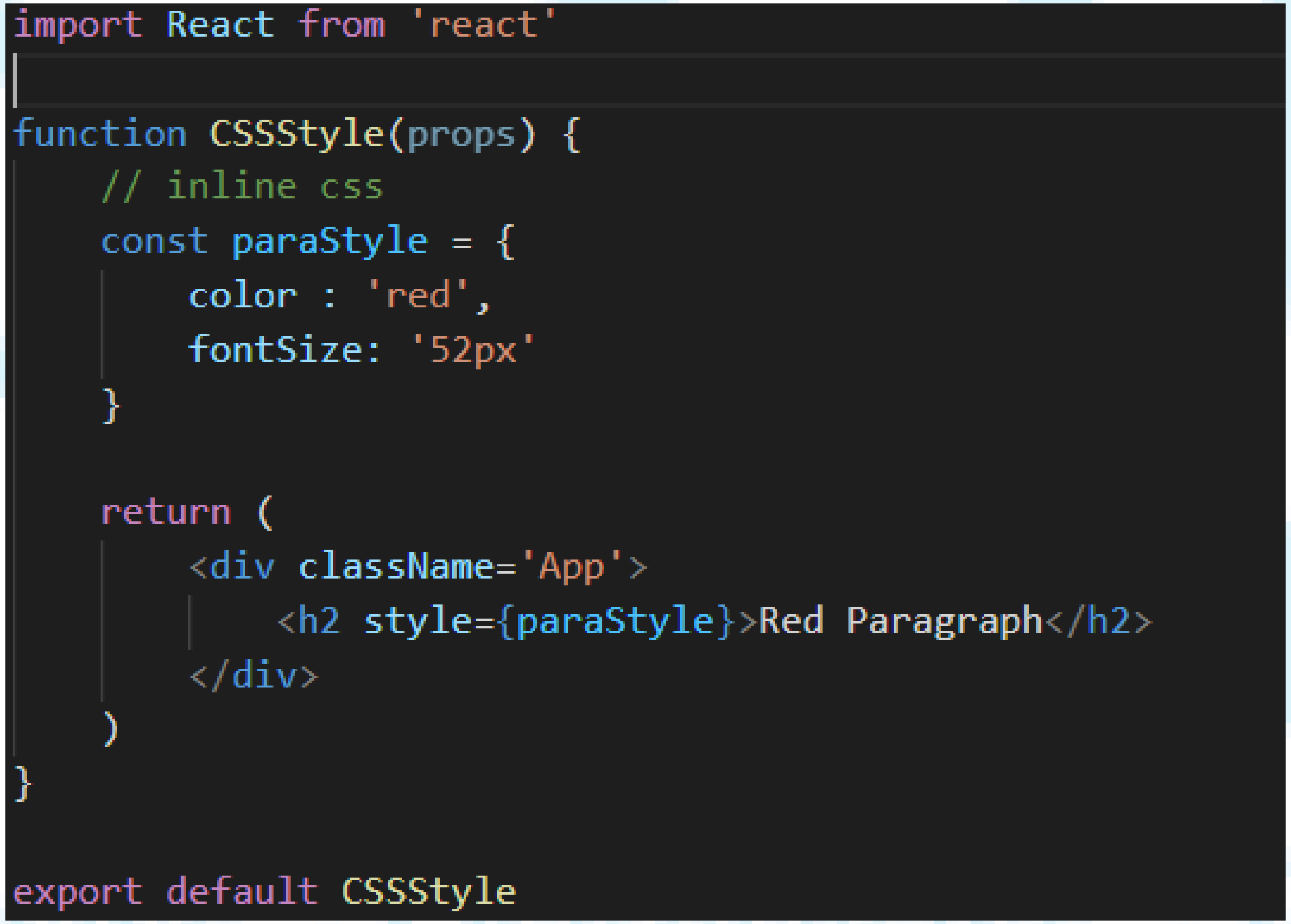


```
src > styles-demo > # myStyle.css >
.heading{
    color: blue;
}

.font-x {
    font-size: 72px;
}
```



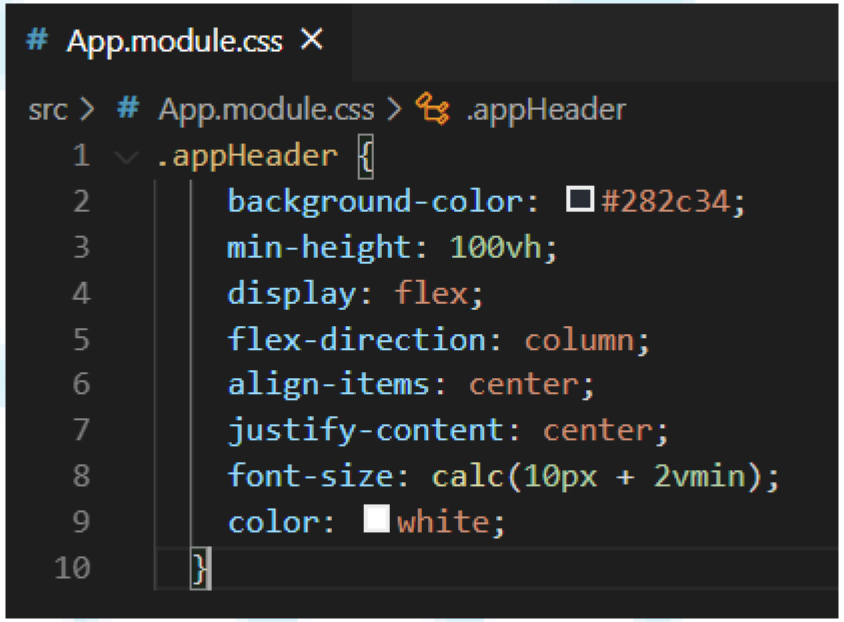
```
import React from 'react'
import './myStyle.css'

function CSSStyle(props) {
    return (
        <div >
            <h1 className='heading'>Hello From CSS Style.</h1>
        </div>
    )
}

export default CSSStyle
```

Inline Styling



```
import React from 'react'

function CSSStyle(props) {
    // inline css
    const paraStyle = {
        color : 'red',
        fontSize: '52px'
    }

    return (
        <div className='App'>
            <h2 style={paraStyle}>Red Paragraph</h2>
        </div>
    )
}

export default CSSStyle
```

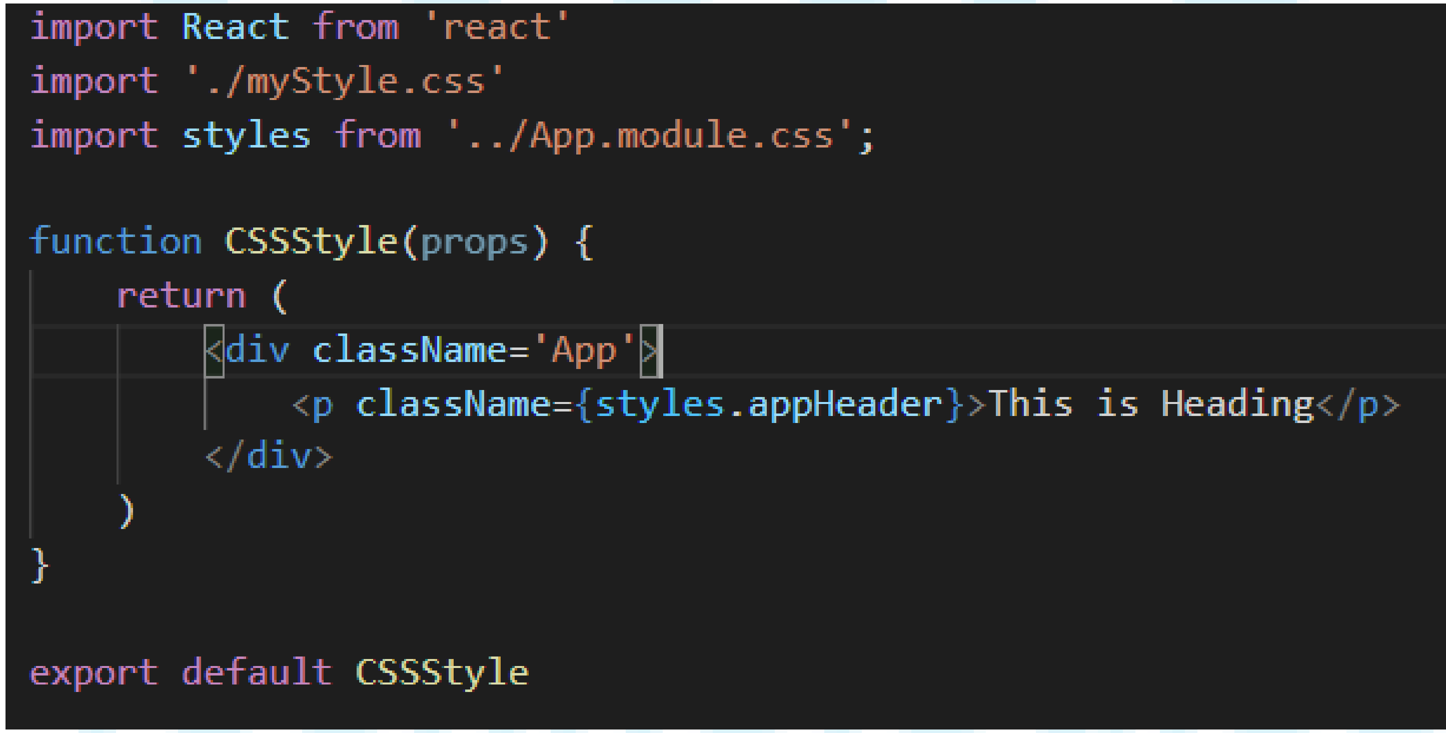
CSS Modules



App.module.css ×

src > # App.module.css > .appHeader

```css
.appHeader {
    background-color: #282c34;
    min-height: 100vh;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    font-size: calc(10px + 2vmin);
    color: white;
}
```



```jsx
import React from 'react'
import './myStyle.css'
import styles from '../App.module.css';

function CSSStyle(props) {
    return (
        <div className='App'>
            <p className={styles.appHeader}>This is Heading</p>
        </div>
    )
}

export default CSSStyle
```

Styled Components React

Styled Components

Installation

Bootstrap with React

React-Bootstrap replaces the Bootstrap JavaScript.

Each component has been built from scratch as a true React component, without unneeded dependencies like jQuery

Getting Started

Components

Module 5

REACT ROUTER

React Router

- Browser - Router
- Link
- Route
- Template integration
- Http Request in React
- Get and Post data

Browser - Router

There's lots of different versions of react routing because developers may find the existing packages to be hard to use or understand and write their own package. Welcome to the wonderful world of JavaScript fragmentation ;p

As you can see from the npm page for `react-router` it is a widely used and well maintained package: https://www.npmjs.com/package/react-router

Router

`BrowserRouter`: Uses the HTML5 history API (pushState, replaceState and the popstate event) to keep your UI in sync with the URL.

`HashRouter`: Uses the hash portion of the URL (i.e. window.location.hash) to keep your UI in sync with the URL.

`MemoryRouter`: Keeps the history of your “URL” in memory (does not read or write to the address bar). Useful in tests and non-browser environments like React Native.

`StaticRouter`: A Router that never changes location. This can be useful in server-side rendering scenarios when the user isn’t actually clicking around, so the location never actually changes. Hence, the name: static. It’s also useful in simple tests when you just need to plug in a location and make assertions on the render output.

HTTP

The Fetch API provides a JavaScript interface for accessing and manipulating parts of the HTTP pipeline, such as requests and responses. It also provides a global `fetch()` method that provides an easy, logical way to fetch resources asynchronously across the network.
This kind of functionality was previously achieved using `XMLHttpRequest`. Fetch provides a better alternative that can be easily used by other technologies such as `Service Workers`. Fetch also provides a single logical place to define other HTTP-related concepts such as CORS and extensions to HTTP.

Fetch HTTP Request

The `fetch` specification differs from `jQuery.ajax()` in three main ways:

- The Promise returned from `fetch()` **won’t reject on HTTP error status** even if the response is an HTTP 404 or 500. Instead, it will resolve normally (with `ok` status set to false), and it will only reject on network failure or if anything prevented the request from completing.

- `fetch()` **won't can receive cross-site cookies**; you can’t can establish a cross site session using fetch. `Set-Cookie` headers from other sites are silently ignored.

- `fetch` **won’t send cookies**, unless you set the *credentials* init option. (Since Aug 25, 2017. The spec changed the default credentials policy to `same-origin`. Firefox changed since 61.0b13.)

Simple GET request using fetch

This sends an HTTP GET request from React to the npm api to search for all react packages using the query q=react, then assigns the total returned in the response to the component state property totalReactPackages so it can be displayed in the render() method.

```
fetch('http://yourlink.com/xyz.json')
 .then(response => response.json())
 .then(data => console.log(data));
```

```
componentDidMount() { // Simple GET request using fetch
  fetch('https://api.npms.io/v2/search?q=react')
  .then(response => response.json())
  .then(data => this.setState({ totalReactPackages: data.total }));
}
```

POST request with a JSON

This sends an HTTP POST request to the JSONPlaceholder api which is a fake online REST api that includes a /posts route that responds to POST requests with the contents of the post body and an id property. The id from the response is assigned to the react component state property postId so it can be displayed in the component render() method.

```
componentDidMount() { // Simple POST request with a JSON body using fetch
  const requestOptions = {
       method: 'POST',
       headers: { 'Content-Type': 'application/json' },
       body: JSON.stringify({ title: 'React POST Request Example' })
   };
   fetch('https://jsonplaceholder.typicode.com/posts', requestOptions)
       .then(response => response.json())
       .then(data => this.setState({ postId: data.id }));
   }
```

Example

https://github.com/TopsCode/ReactJs/tree/main/Module5

Module 6

REDUX

React Redux

- Complexity of Managing state
- Understand the Redux Flow
- Setting up Reducer and store
- Dispatching Actions
- Passing and Retrieving Data with Action
- Combining Multiple Reducers
- Adding Middleware
- Redux Dev tools

Redux

“Redux is predictable state container for JavaScript apps”

It is for JavaScript apps

It is a state container

It is predictable

Redux is for JavaScript applications

Redux is not tied to React

Can be used with React, Angular, Vue and even vanilla JavaScript

Redux is a library for JavaScript applications

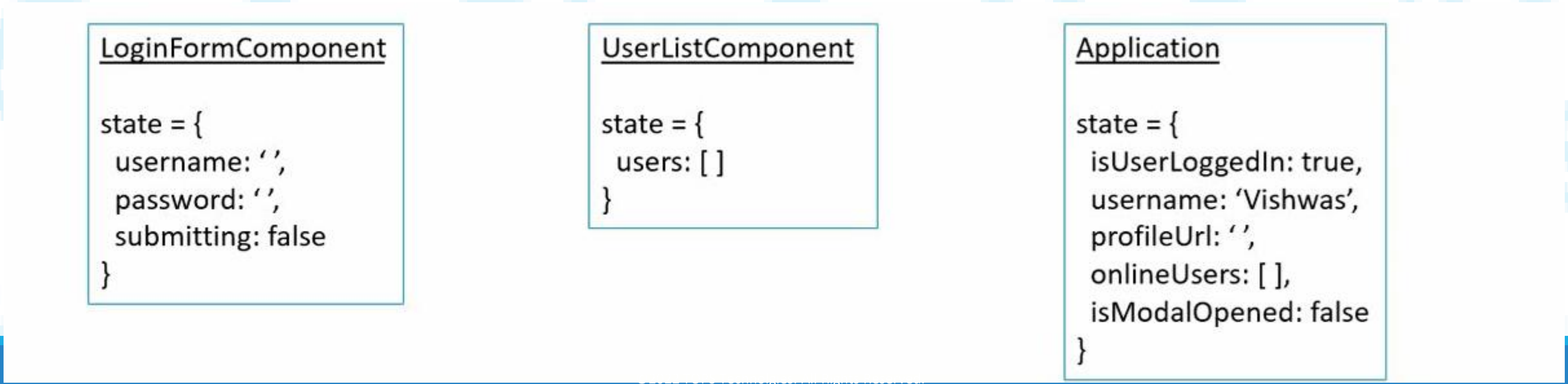
Redux is a state container

Redux stores the state of your application

Consider a React app – state of component

State of an app is the state represented by all the individual components of that app

Redux will store and manage the application state



LoginFormComponent

```
state = {
  username: ‘ ’,
  password: ‘ ’,
  submitting: false
}
```

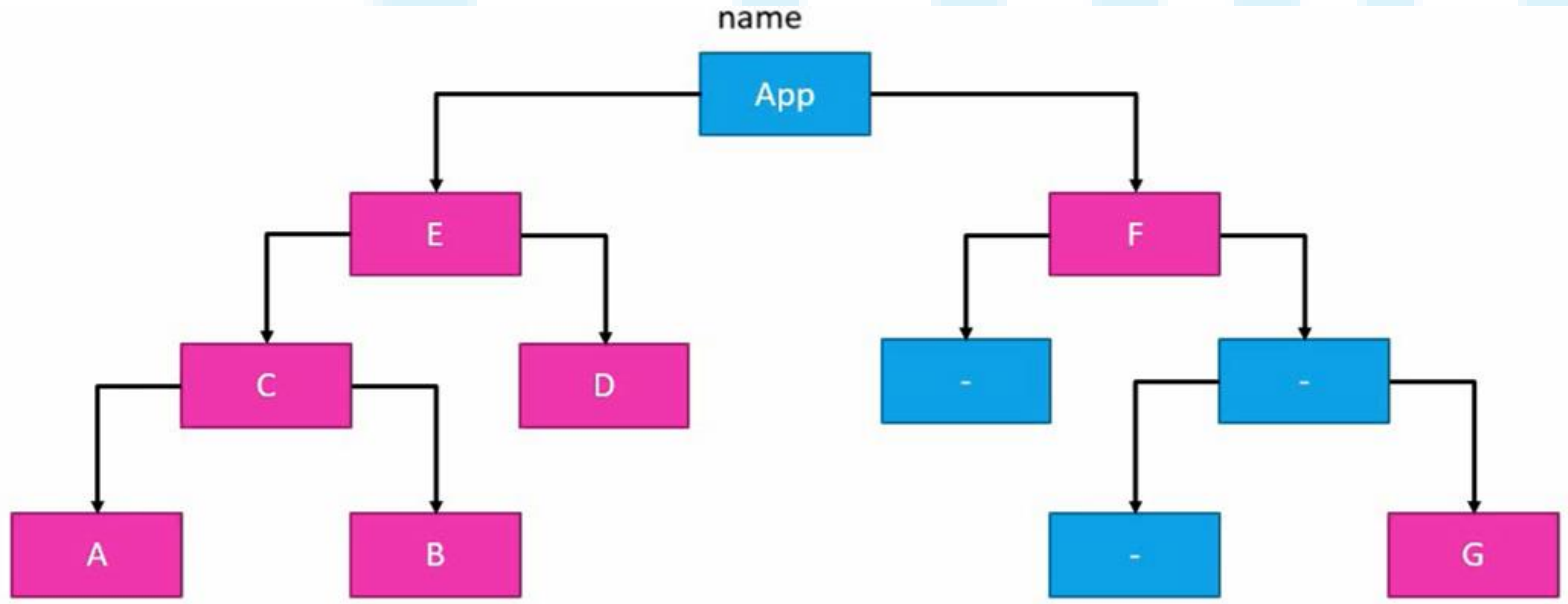
UserListComponent

```
state = {
  users: [ ]
}
```

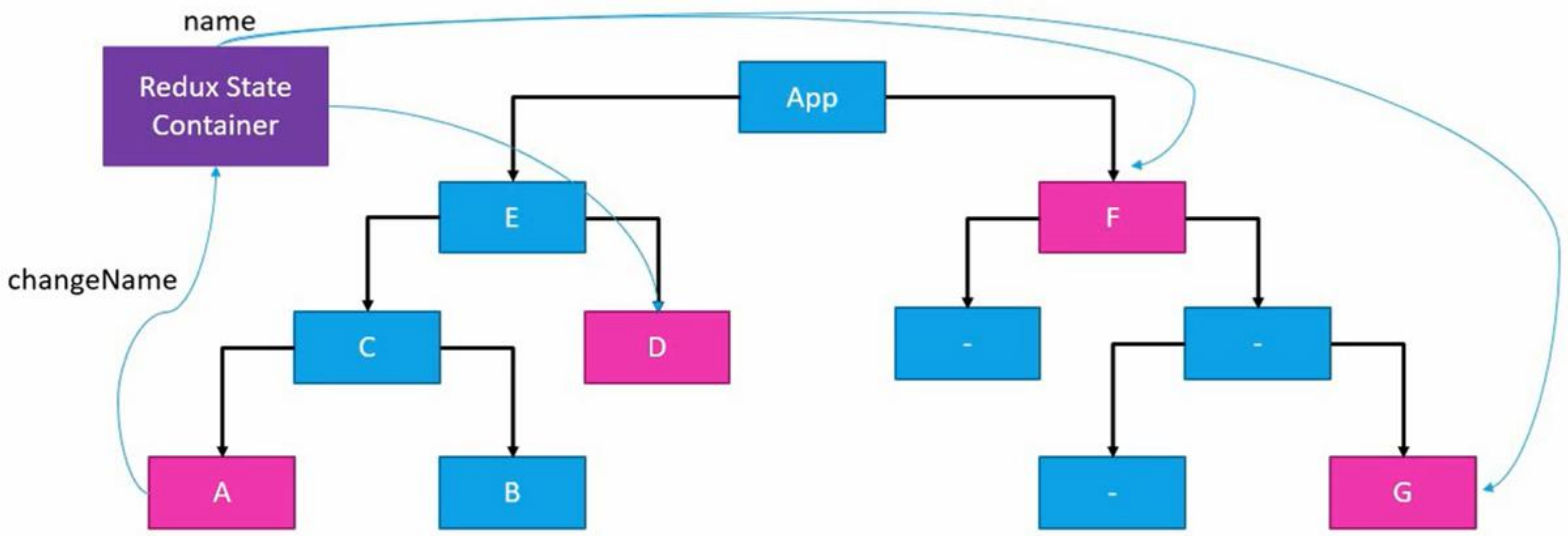
Application

```
state = {
  isUserLoggedIn: true,
  username: ‘Vishwas’,
  profileUrl: ‘ ’,
  onlineUsers: [ ],
  isModalOpened: false
}
```

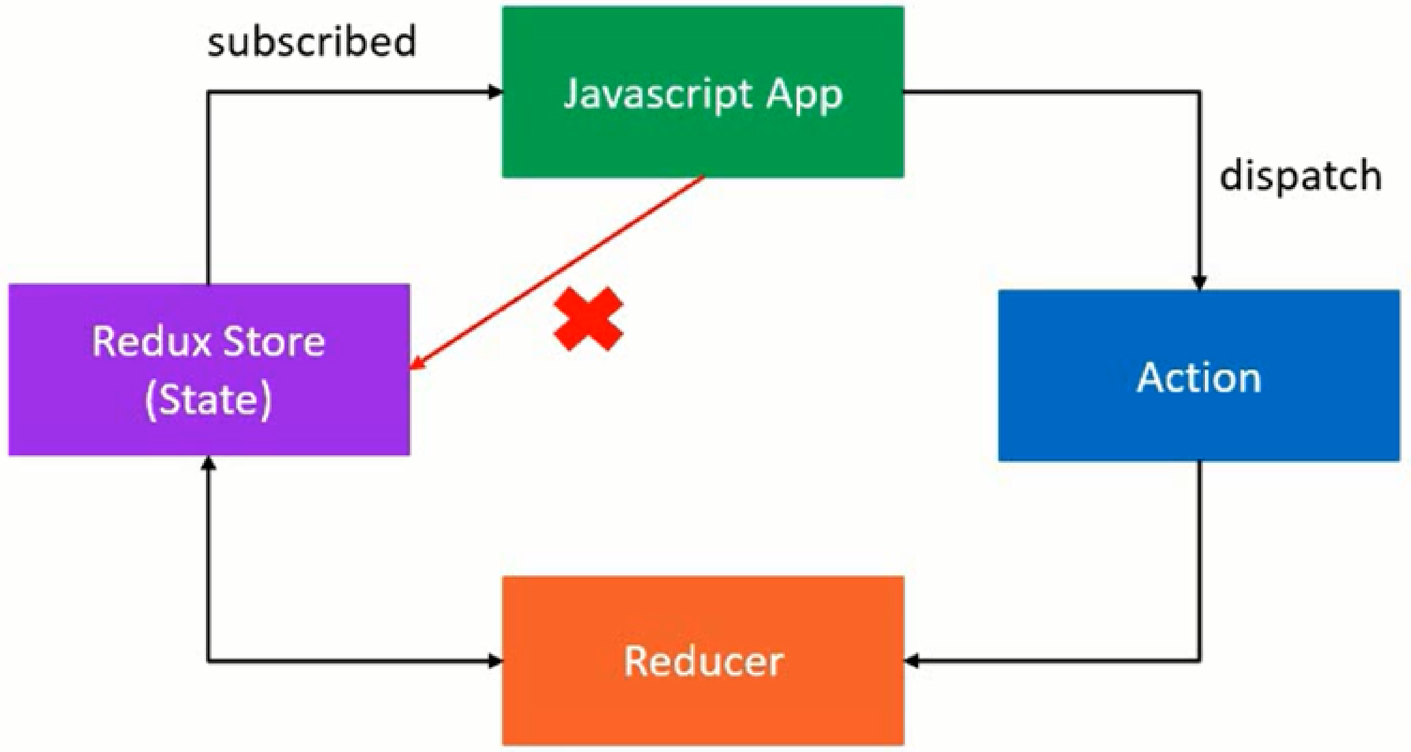
Complexity of Managing state



React + Redux



Understanding the Redux Flow



Actions

Synchronous Actions

As soon as an action was dispatched, the state was immediately updated.

If you dispatch the BUY_CAKE action, the numOfCakes was right away decremented by 1.

Same with BUY_ICECREAM action as well.

Async Actions

Asynchronous API calls to fetch data from an end point and use that data in your application.

Our Application

Fetches a list of users from an API end point and stores it in the redux store.

State ?

Actions ?

Reducer ?

State

```
state = {
  loading: true,
  data: [ ],
  error: ‘ ’
 }
```

loading - Display a loading spinner in your component
data - List of users
error – Display error to the user

Actions

FETCH_USERS_REQUEST – Fetch list of users

FETCH_USERS_SUCCESS – Fetched successfully

FETCH_USERS_FAILURE – Error fetching the data

Reducers

case: **FETCH_USERS_REQUEST**

loading: true

case: **FETCH_USERS_SUCCESS**

loading: false

users: data (from API)

case: **FETCH_USERS_FAILURE**

loading: false

error: error (from API)

Async action creators

axios

Requests to an API end point

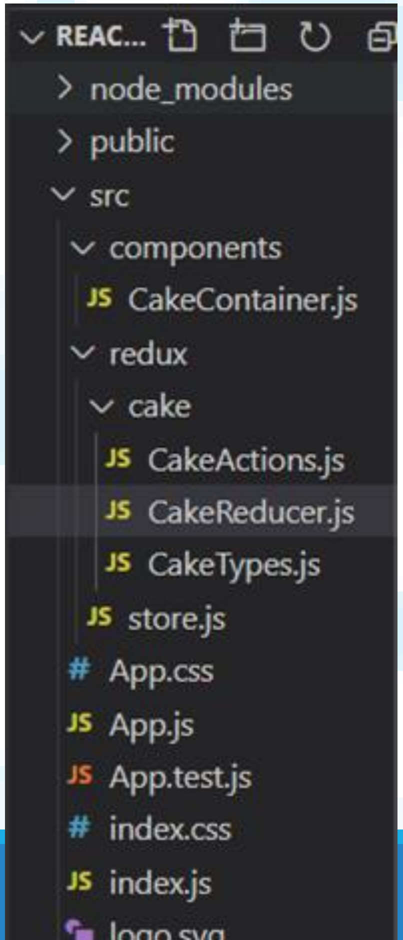
redux-thunk

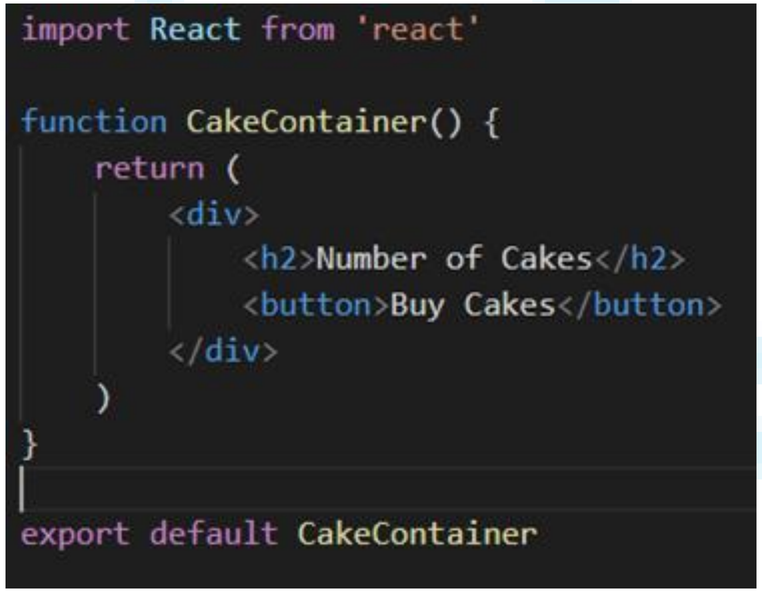
Define async action creators

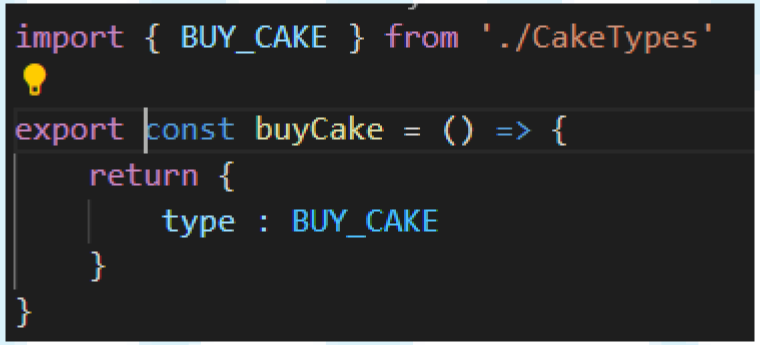
Middleware

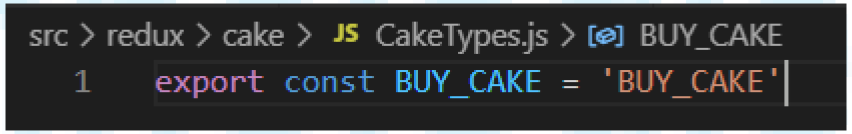
React + Redux Demo

npm install redux react-redux



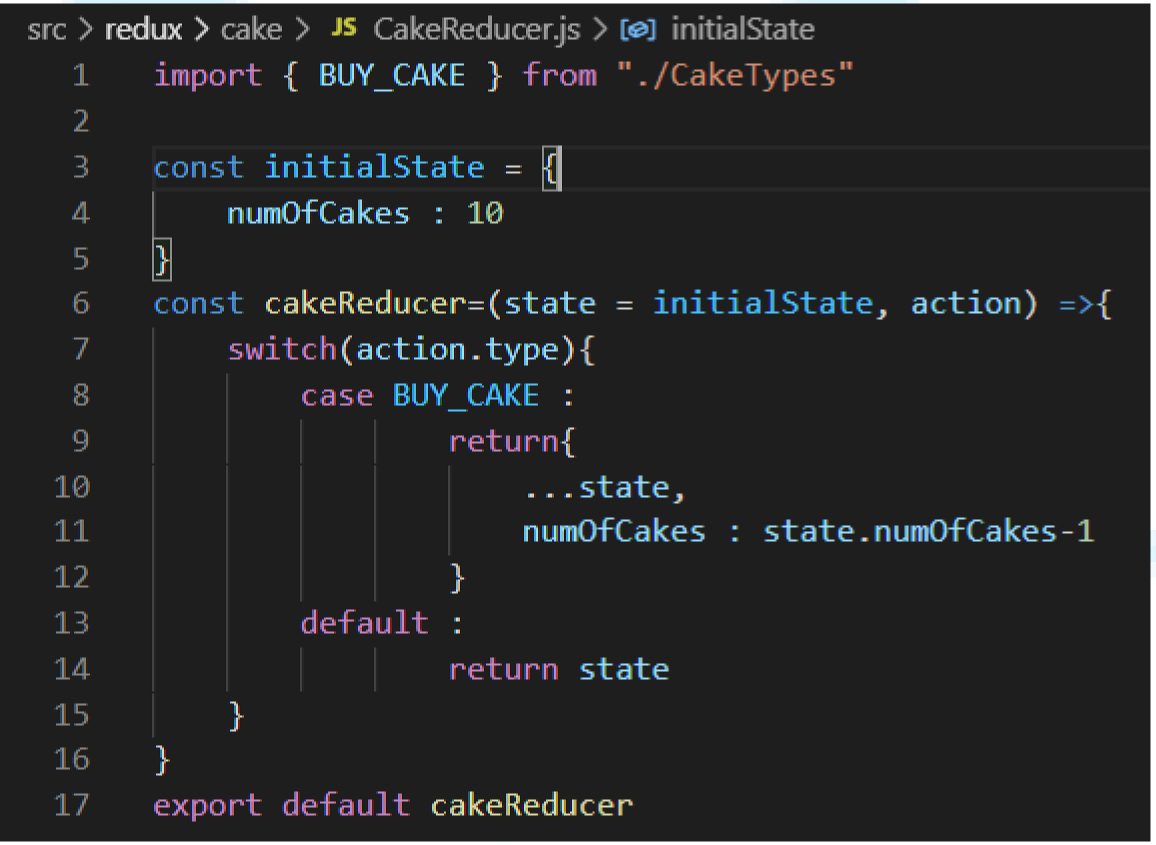

```
import React from 'react'

function CakeContainer() {
    return (
        <div>
            <h2>Number of Cakes</h2>
            <button>Buy Cakes</button>
        </div>
    )
}

export default CakeContainer
```


```
import { BUY_CAKE } from './CakeTypes'

export const buyCake = () => {
    return {
        type : BUY_CAKE
    }
}
```


src > redux > cake > CakeTypes.js > BUY_CAKE

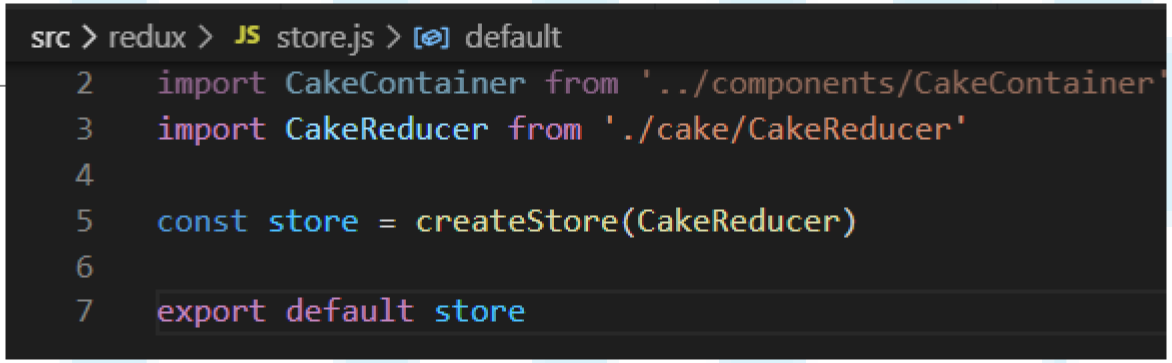
```
export const BUY_CAKE = 'BUY_CAKE'
```

Reducer & store



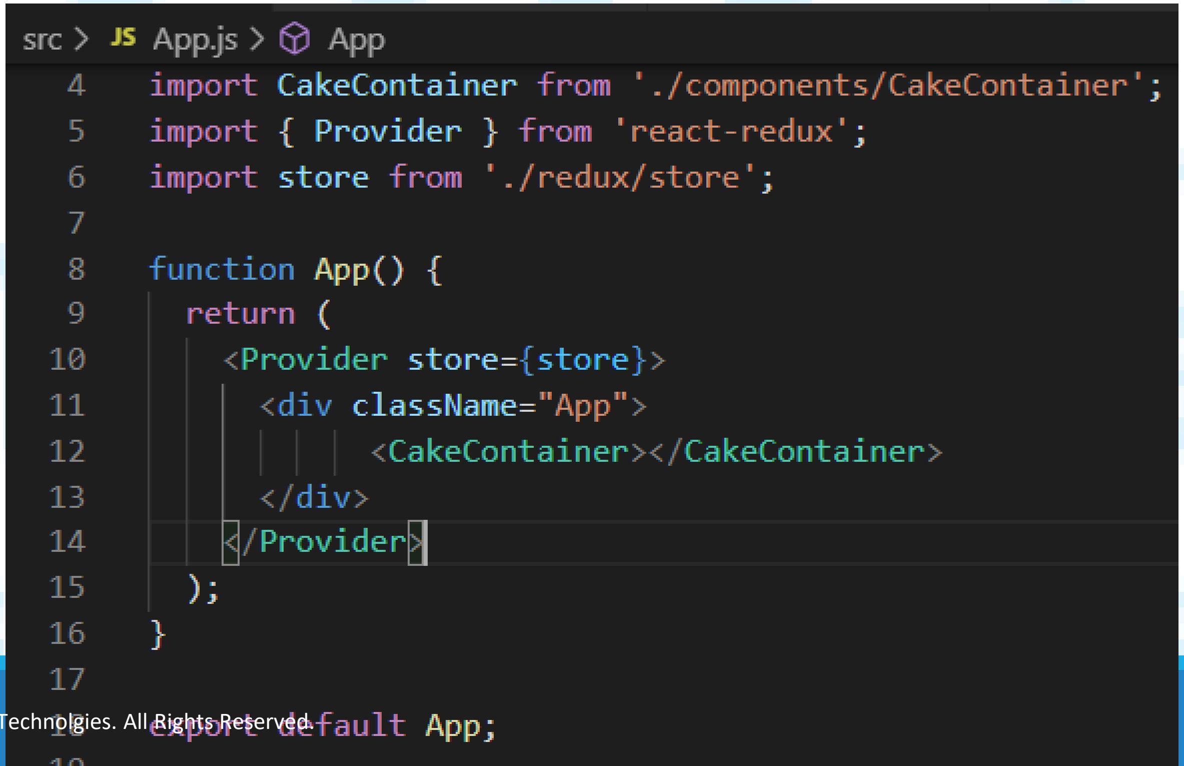
src > redux > cake > JS CakeReducer.js > initialState

```javascript
import { BUY_CAKE } from "./CakeTypes"

const initialState = {
    numOfCakes : 10
}
const cakeReducer=(state = initialState, action) =>{
    switch(action.type){
        case BUY_CAKE :
                return{
                    ...state,
                    numOfCakes : state.numOfCakes-1
                }
        default :
                return state
    }
}
export default cakeReducer
```



src > redux > JS store.js > default

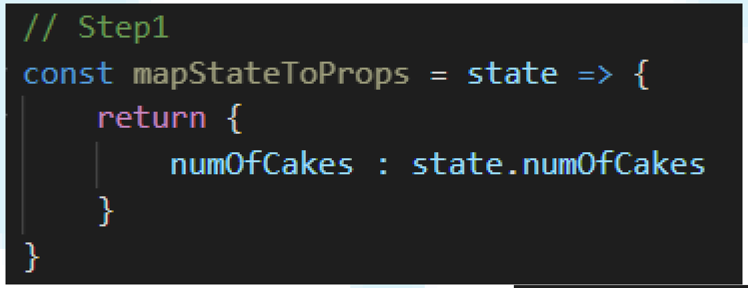
```javascript
import CakeContainer from '../components/CakeContainer'
import CakeReducer from './cake/CakeReducer'

const store = createStore(CakeReducer)

export default store
```



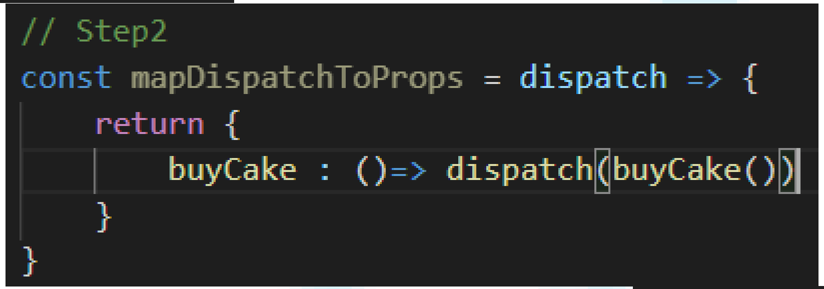
src > JS App.js > App

```javascript
import CakeContainer from './components/CakeContainer';
import { Provider } from 'react-redux';
import store from './redux/store';

function App() {
  return (
    <Provider store={store}>
      <div className="App">
          <CakeContainer></CakeContainer>
      </div>
    </Provider>
  );
}

export default App;
```

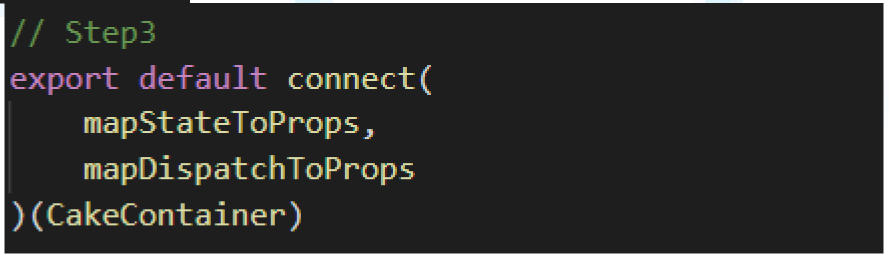
Redux in CakeContainer



```
// Step1
const mapStateToProps = state => {
    return {
        numOfCakes : state.numOfCakes
    }
}
```

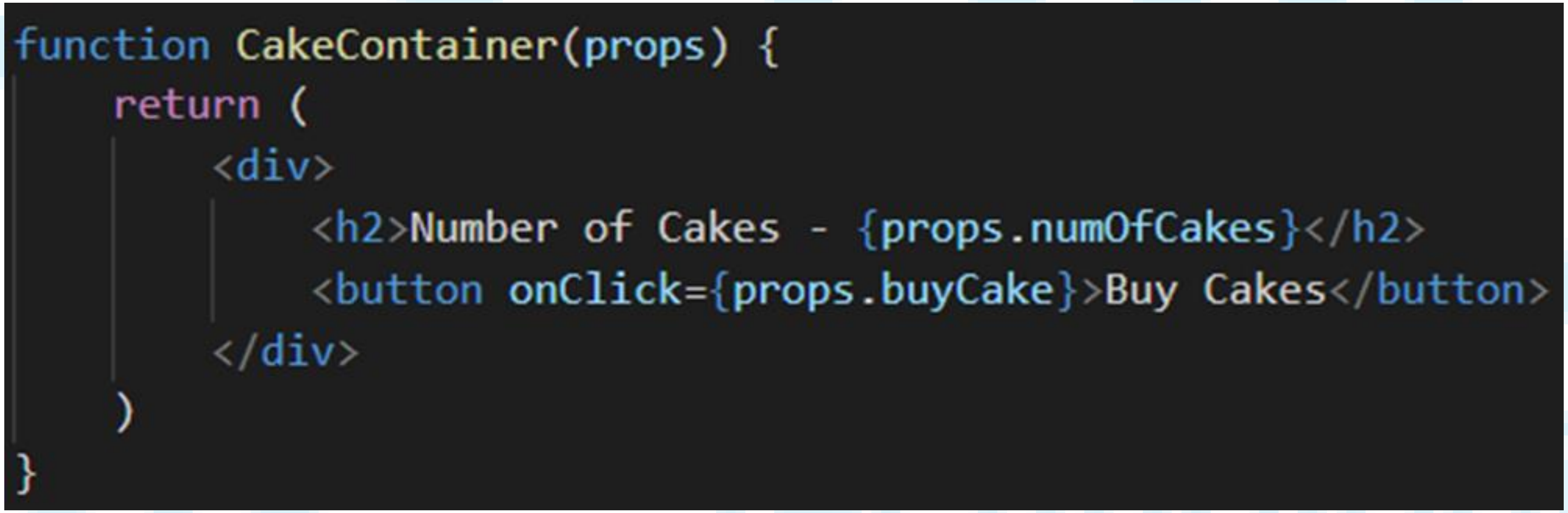


```
// Step2
const mapDispatchToProps = dispatch => {
    return {
        buyCake : ()=> dispatch(buyCake())
    }
}
```



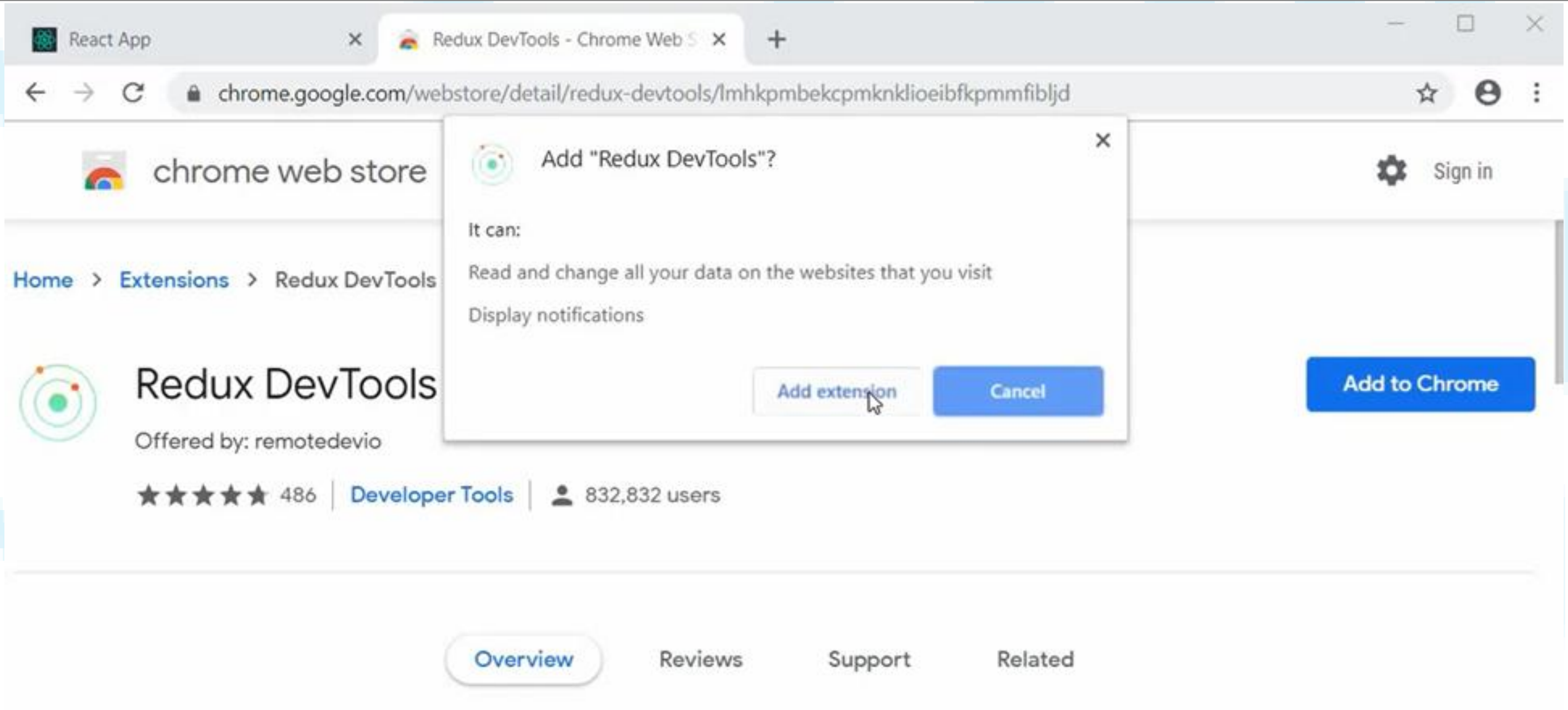
```
// Step3
export default connect(
    mapStateToProps,
    mapDispatchToProps
)(CakeContainer)
```

Finally we get CakeContainer props



```
function CakeContainer(props) {
    return (
        <div>
            <h2>Number of Cakes - {props.numOfCakes}</h2>
            <button onClick={props.buyCake}>Buy Cakes</button>
        </div>
    )
}
```

Redux Dev tools Extension


React App
Redux DevTools - Chrome Web S
chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd
chrome web store
Sign in
Home > Extensions > Redux DevTools
Redux DevTools
Offered by: remotedevio
486
Developer Tools
832,832 users
Add to Chrome
Overview
Reviews
Support
Related

Add "Redux DevTools"?

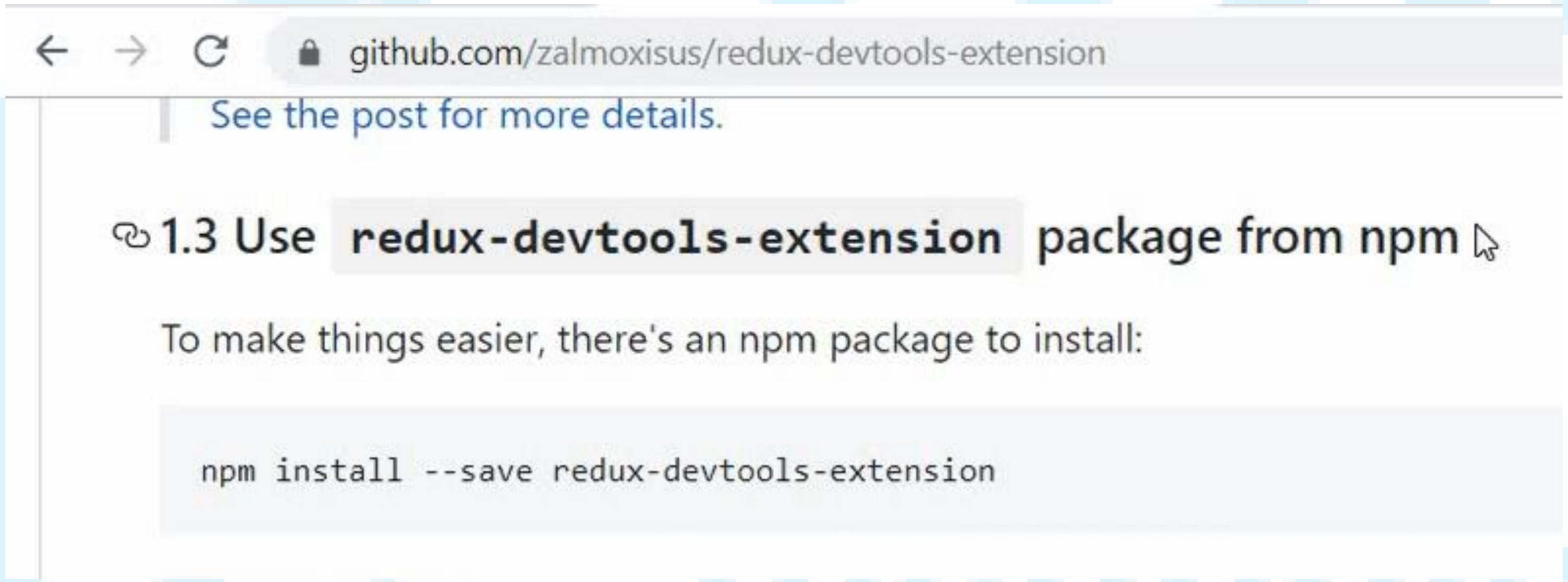
It can:

Read and change all your data on the websites that you visit

Display notifications

Add extension Cancel

Redux-devtools in React

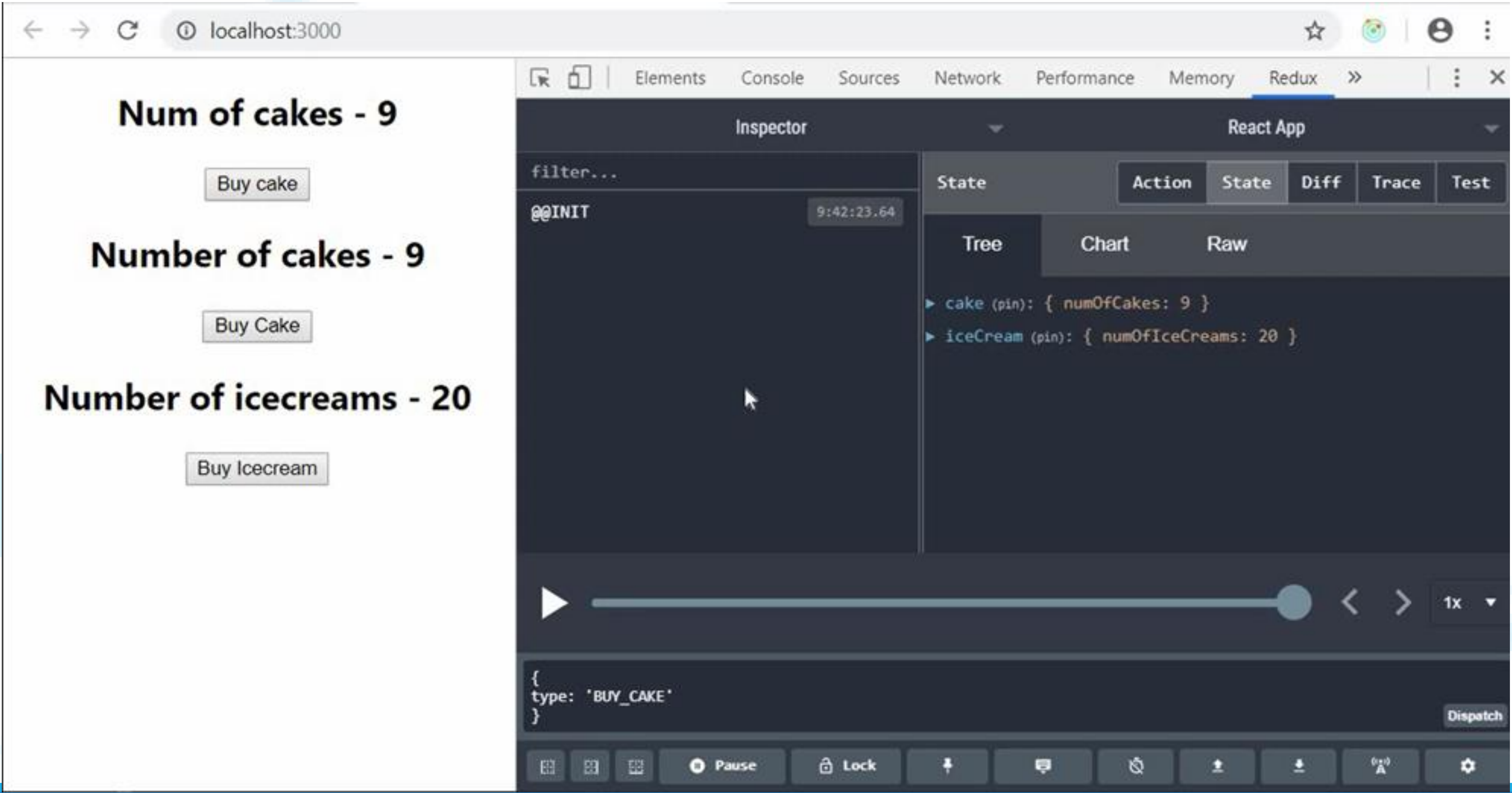

github.com/zalmoxisus/redux-devtools-extension

See the post for more details.

1.3 Use `redux-devtools-extension` package from npm

To make things easier, there's an npm package to install:

```
npm install --save redux-devtools-extension
```



```
▸ cake (pin): { numOfCakes: 9 }
▸ iceCream (pin): { numOfIceCreams: 20 }
```

```
{
type: 'BUY_CAKE'
}
```