**React Js**

[**https://www.javatpoint.com/react-version**](https://www.javatpoint.com/react-version)

A complete release history for React is given below. You can also see the full documentation for recent releases on GitHub.

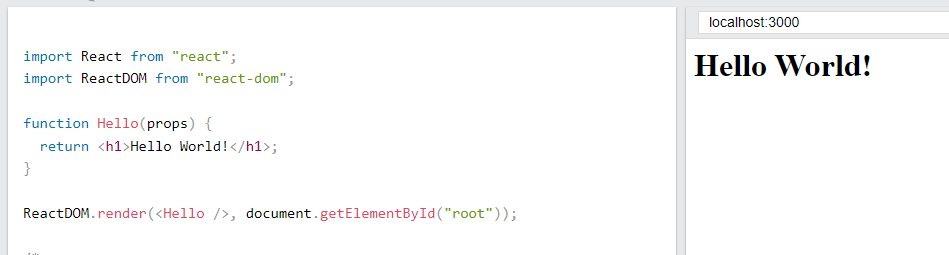
# React Features

Currently, ReactJS gaining quick popularity as the best JavaScript framework among web developers. It is playing an essential role in the front-end ecosystem. The important features of ReactJS are as following.

* JSX
* Components
* One-way Data Binding
* Virtual DOM
* Simplicity
* Performance

[**https://www.javatpoint.com/reactjs-vs-angularjs**](https://www.javatpoint.com/reactjs-vs-angularjs)

<https://www.w3schools.com/REACT/default.asp>



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

\*\*\* Also we have to learn react in html code in later chapter sample code is



What is ES6?

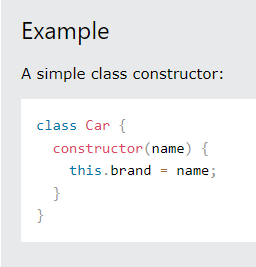
ES6 stands for ECMAScript 6.

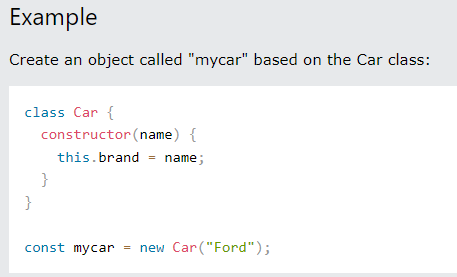
ECMAScript was created to standardize JavaScript, and ES6 is the 6th version of ECMAScript, it was published in 2015, and is also known as ECMAScript 2015.

React uses ES6, and you should be familiar with some of the new features like:

* [Classes](https://www.w3schools.com/REACT/react_es6_classes.asp)
* [Arrow Functions](https://www.w3schools.com/REACT/react_es6_arrow.asp)
* [Variables](https://www.w3schools.com/REACT/react_es6_variables.asp) (let, const, var)
* [Array Methods](https://www.w3schools.com/REACT/react_es6_array_methods.asp) like .map()
* [Destructuring](https://www.w3schools.com/REACT/react_es6_destructuring.asp)
* [Modules](https://www.w3schools.com/REACT/react_es6_modules.asp)
* [Ternary Operator](https://www.w3schools.com/REACT/react_es6_ternary.asp)
* [Spread Operator](https://www.w3schools.com/REACT/react_es6_spread.asp)

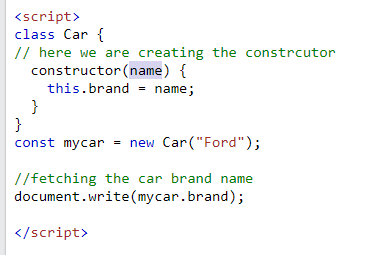
Class



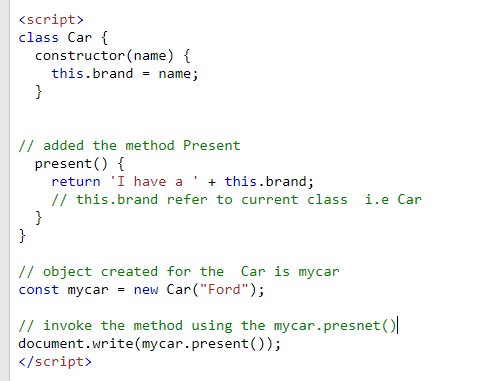


Code

How to define the constructor



How to write the method ?



## Class Inheritance

## Class a

## 

## Class b

## 

The super() method refers to the parent class.

By calling the super() method in the constructor method, we call the parent's constructor method and gets access to the parent's properties and methods.

## Invoke the method show()..

## 

## Output

## 

## Functions

## 

## Output

## 

## <p>A demonstration of a simple arrow function.</p>

## 

## Not required to define the return key.

## <p>The arrow function expects a return value, and returns the value by default, without the <strong>return</strong> keyword.</p>

## 

## <p>A demonstration of an arrow function in one line, with parameters.</p>

## 

## <p>As you can see in this example, you can skip the parentheses when you have only one parameter.</p>

## 

## VARIABLES : -

var x = 5.6;

If you use var outside of a function, it belongs to the global scope.

If you use var inside of a function, it belongs to that function.

If you use var inside of a block, i.e. a for loop, the variable is still available outside of that block.

let x = 5.6;

let is the block scoped version of var, and is limited to the block (or expression) where it is defined.

If you use let inside of a block, i.e. a for loop, the variable is only available inside of that loop.

let has a *block* scope.

### const

const x = 5.6;

const is a variable that once it has been created, its value can never change.

const has a block scope.

The keyword const is a bit misleading.

It does not define a constant value. It defines a constant reference to a value.

Because of this you can NOT:

* Reassign a constant value
* Reassign a constant array
* Reassign a constant object

But you CAN:

* Change the elements of constant array
* Change the properties of constant object

# React Render HTML

React's goal is in many ways to render HTML in a web page.

React renders HTML to the web page by using a function called ReactDOM.render().

* The ReactDOM.render() function takes two arguments, HTML code and an HTML element.

ReactDOM.render(<p>Hello</p>, document.getElementById('root'));

* The purpose of the function is to display the specified HTML code inside the specified HTML element.
* But render where?
* There is another folder in the root directory of your React project, named "public". In this folder, there is an index.html file.
* You'll notice a single <div> in the body of this file. This is where our React application will be rendered.

## The HTML Code

The HTML code in this tutorial uses JSX which allows you to write HTML tags inside the JavaScript code:

Do not worry if the syntax is unfamiliar, you will learn more about JSX in the next chapter.

const myelement (

//html code

)

ReactDOM.render(myelement, document.getElementById('root'));



You can note.. root element

ReactDOM.render(myelement, document.getElementById('root'));

You can define any thing in the root element- like defined below



# React JSX

## What is JSX?

* JSX stands for JavaScript XML.
* JSX allows us to write HTML in React.
* JSX makes it easier to write and add HTML in React.

Advantages

## Coding JSX

JSX allows us to write HTML elements in JavaScript and place them in the DOM without any createElement()  and/or appendChild() methods.

JSX converts HTML tags into react elements.

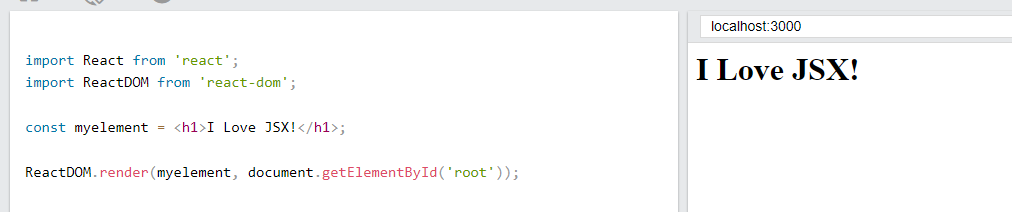
You are not required to use JSX, but JSX makes it easier to write React applications.

Using jsx

const myelement = <h1>I Love JSX!</h1>;

// defined just html code

ReactDOM.render(myelement, document.getElementById('root'));



As you can see in the first example, JSX allows us to write HTML directly within the JavaScript code.

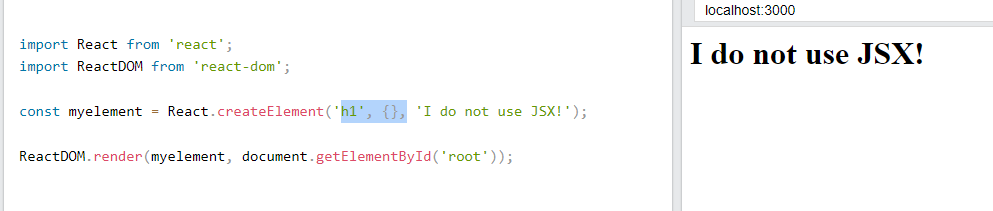
JSX is an extension of the JavaScript language based on ES6, and is translated into regular JavaScript at runtime.

With out jsx

const myelement = React.createElement('h1', {}, 'I do not use JSX!');

using the method called React.createElement()

ReactDOM.render(myelement, document.getElementById('root'));



## Expressions in JSX

## Execute the expression 5 + 5: // add/sub/mul… etc

const myelement = <h1>React is {5 + 5} times better with JSX</h1>;

## 

## Inserting a Large Block of HTML

## LIST -

Create a list with three list items:

const myelement = (

// list of items…

<ul>

<li>Apples</li>

<li>Bananas</li>

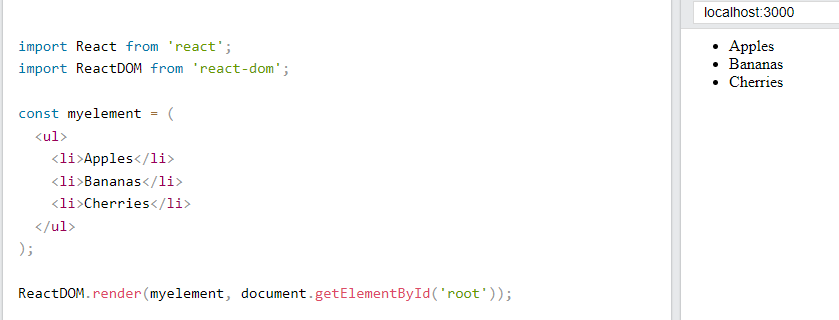
<li>Cherries</li>

</ul>

);

// how render on the browser

ReactDOM.render(myelement, document.getElementById('root'));



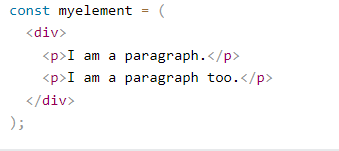
## 

## Also you can use<></>

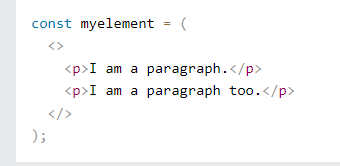
Alternatively, you can use a "fragment" to wrap multiple lines. This will prevent unnecessarily adding extra nodes to the DOM.

A fragment looks like an empty HTML tag: <></>.

Case 1: // <div> </div>



Case 2:  <></>.



Rule-JSX follows XML rules, and therefore HTML elements must be properly closed

Close empty elements with />

const myelement = <input type="text" />;



## Attribute class = className

The class attribute is a much used attribute in HTML, but since JSX is rendered as JavaScript, and the class keyword is a reserved word in JavaScript, you are not allowed to use it in JSX.

Use attribute className instead.

JSX solved this by using className instead. When JSX is rendered, it translates className attributes into class attributes.

const myelement = <h1 className="myclass">Hello World</h1>;

## 

## Conditions - if statements

React supports if statements, but not *inside*JSX.

To be able to use conditional statements in JSX, you should put the if statements outside of the JSX, or you could use a ternary expression instead:

Case 1- using if condition out side jsx code.

const x = 5; // value not to be change

let text = "Goodbye";// let is defined so that value can change

if (x < 10) { // condition..

text = "Hello";

}

const myelement = <h1>{text}</h1>; //jsx code

Code : -

## 

#### Option 2:

Use ternary expressions instead:

### Example

Write "Hello" if x is less than 10, otherwise "Goodbye":

const x = 5; // value not to be changed

const myelement = <h1>{(x) < 10 ? "Hello" : "Goodbye"}</h1>; //jsx code

// where ternary operation used

## Code:-

## 

# React Components

## Components are like functions that return HTML elements.

## React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML.

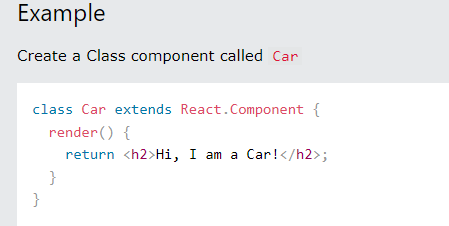
Components come in two types, Class components and Function components, in this tutorial we will concentrate on Function components.

Note :-

In older React code bases, you may find Class components primarily used. It is now suggested to use Function components along with Hooks, which were added in React 16.8.

There is an optional section on Class components for your reference.

Class components

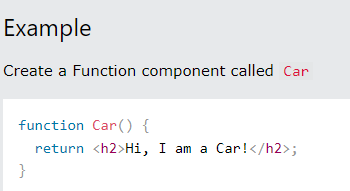


Function component

### Function Component-re easier to understand, and will be preferred in this tutorial

Here is the same example as above, but created using a Function component instead.

A Function component also returns HTML, and behaves much the same way as a Class component, but Function components can be written using much less code, are easier to understand, and will be preferred in this tutorial.



## Rendering a Component

Now your React application has a component called Car, which returns an <h2> element.

To use this component in your application, use similar syntax as normal HTML: <Car />

### Example

Display the Car component in the "root" element:

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

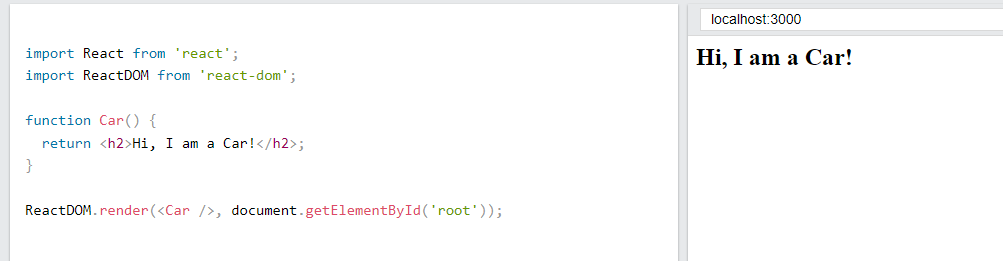
IN PREVIOUSLY WE USE const xyx = html code

const myelement = <h1>I Love JSX!</h1>;

// defined just html code

ReactDOM.render(myelement, document.getElementById('root'));

Now lets use the component-



## Props

Components can be passed as props, which stands for properties.

Props are like function arguments, and you send them into the component as attributes.

### Example

Use an attribute to pass a color to the Car component, and use it in the render() function:

function Car(props) {

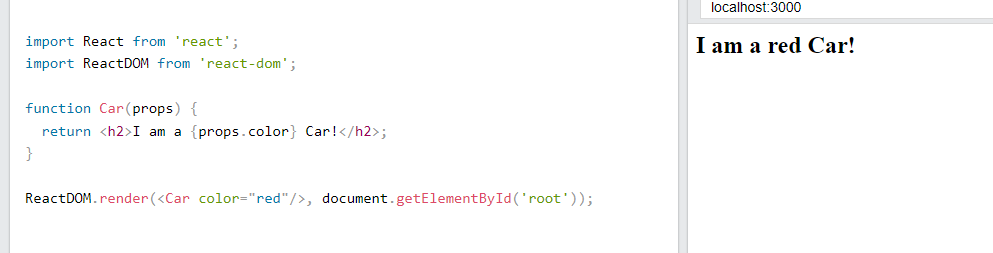
return <h2>I am a {props.color} Car!</h2>;

}

ReactDOM.render(<Car color="red"/>, document.getElementById('root'));

In Below code.. props are yet not defined type like string or integer etc.

You will learn more about props in the next chapter.



## Components in Components

We can refer to components inside other components:

Use the Car component inside the Garage component:

// component 1- car

function Car() {

return <h2>I am a Car!</h2>;

}

// component 2- garage

function Garage() {

return (

<> // using the only brackets its html code.

<h1>Who lives in my Garage?</h1>

<Car />

// here we are invoked Car – component

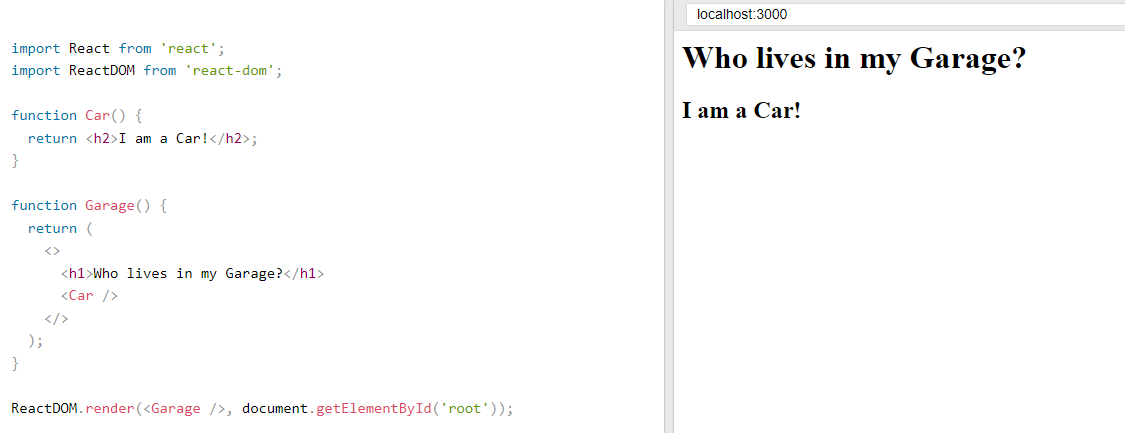
</>

);

}

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

Code :-



## Components in Files

React is all about re-using code, and it is recommended to split your components into separate files.

To do that, create a new file with a .js file extension and put the code inside it:

Note that the filename must start with an uppercase character.

## Create car component as car.js

## 

## Import in the main class ie. App.js

## 

## https://www.w3schools.com/REACT/react\_class.asp

## More about the classes

# React Class ComponentsBefore React 16.8, Class components were the only way to track state and lifecycle on a React component. Function components were considered "state-less".

With the addition of Hooks, Function components are now almost equivalent to Class components. The differences are so minor that you will probably never need to use a Class component in React.

Even though Function components are preferred, there are no current plans on removing Class components from React.

This section will give you an overview of how to use Class components in React.

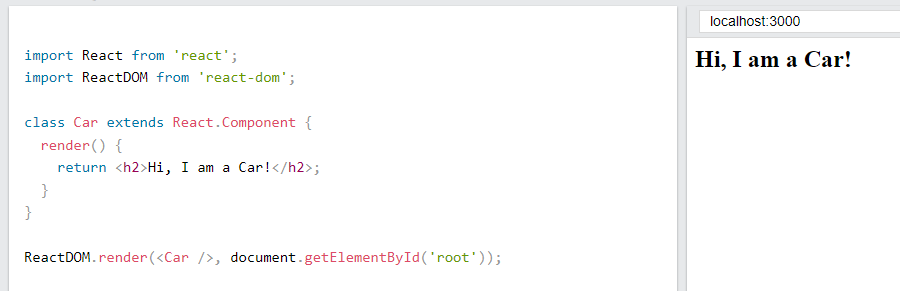
Def – component

## React Components

Components are independent and reusable bits of code. They serve the same purpose as JavaScript functions, but work in isolation and return HTML via a render() function.

Components come in two types, Class components and Function components, in this chapter you will learn about Class components.

Class components-



## Component Constructor

If there is a constructor() function in your component, this function will be called when the component gets initiated.

The constructor function is where you initiate the component's properties.

In React, component properties should be kept in an object called state.

You will learn more about state later in this tutorial.

\*\*\*\*

The constructor function is also where you honor the inheritance of the parent component by including the super() statement, which executes the parent component's constructor function, and your component has access to all the functions of the parent component (React.Component).

### Example

Create a constructor function in the Car component, and add a color property:

class Car extends React.Component {

constructor() {

super() // calling the parent constructor;

this.state = {color: "red"}

//this. ->define the current class;

//state is attribute and value is defined as key and value i.e

{color: “red”}

}

render() {

return <h2>I am a {this.state.color} Car!</h2>;

}

}

## Code :-

## 

## Props

Another way of handling component properties is by using props.

Props are like function arguments, and you send them into the component as attributes.

You will learn more about props in the next chapter.

Use an attribute to pass a color to the Car component, and use it in the render() function:

class Car extends React.Component {

render() {

return <h2>I am a {this.props.color} Car!</h2>;

}

}

//this.props.color is replaced by Red

ReactDOM.render(<Car color="red"/>, document.getElementById('root'));

## Code

## 

## Props in the Constructor

If your component has a constructor function, the props should always be passed to the constructor and also to the React.Component via the super() method.

class Car extends React.Component {

// passing the props in constructor and super keyword

constructor(props)//to current class {

super(props);// to React.Component

}

render() {

return <h2>I am a {this.props.model}!</h2>;

}

}

ReactDOM.render(<Car model="Mustang"/>, document.getElementById('root'));

## Components in Components

We can refer to components inside other components:

Use the Car component inside the Garage component:

Case1-// we are using the class component

class Car extends React.Component {

render() {

return <h2>I am a Car!</h2>;

}

}

class Garage extends React.Component {

render() {

return (

<div>

<h1>Who lives in my Garage?</h1>

<Car />

</div>

);

}

}

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

Case 2- using the function component

// component 1- car

function Car() {

return <h2>I am a Car!</h2>;

}

// component 2- garage

function Garage() {

return (

<> // using the only brackets its html code.

<h1>Who lives in my Garage?</h1>

<Car />

// here we are invoked Car – component

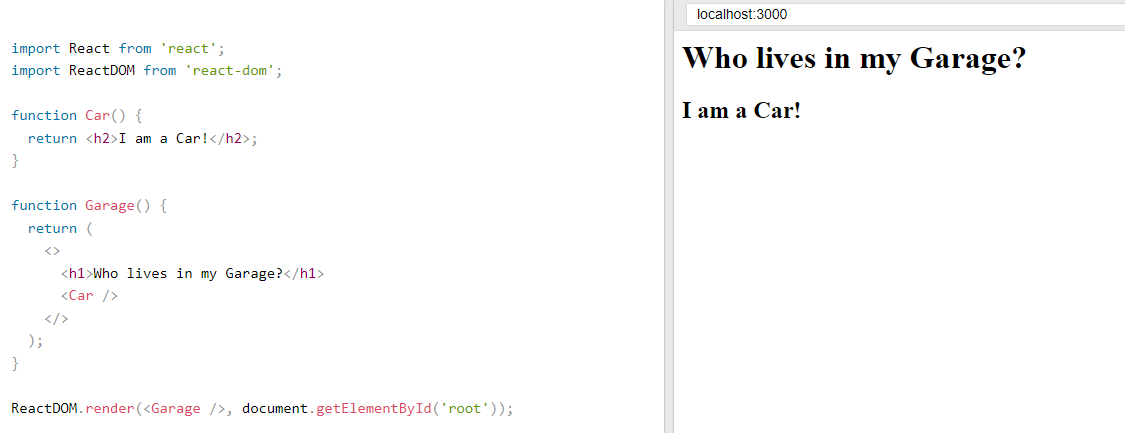
</>

);

}

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

Code :-



## Components in Files

React is all about re-using code, and it can be smart to insert some of your components in separate files.

To do that, create a new file with a .js file extension and put the code inside it:

Note that the file must start by importing React (as before), and it has to end with the statement export default Car;.

## S1-

## 

## S2-

## 

## https://www.w3schools.com/REACT/react\_class.asp

## React Class Component State

React Class components have a built-in state object.

You might have noticed that we used state earlier in the component constructor section.



The state object is where you store property values that belongs to the component.

When the state object changes, the component re-renders.

## Single value

class Car extends React.Component {

constructor(props) {

super(props);

// only one property

this.state = {brand: "Ford"};

}

render() {

return (

<div>

<h1>My Car</h1>

</div>

);

}

}

## 

## Specify all the properties your component need:

class Car extends React.Component {

constructor(props) {

super(props);

// specify multiple properties

this.state = {

brand: "Ford",

model: "Mustang",

color: "red",

year: 1964

};

}

render() {

return (

<div>

<h1>My Car</h1>

</div>

);

}

}

## 

## 

## Using the state Object

Refer to the state object anywhere in the component by using the this.state.propertyname syntax:

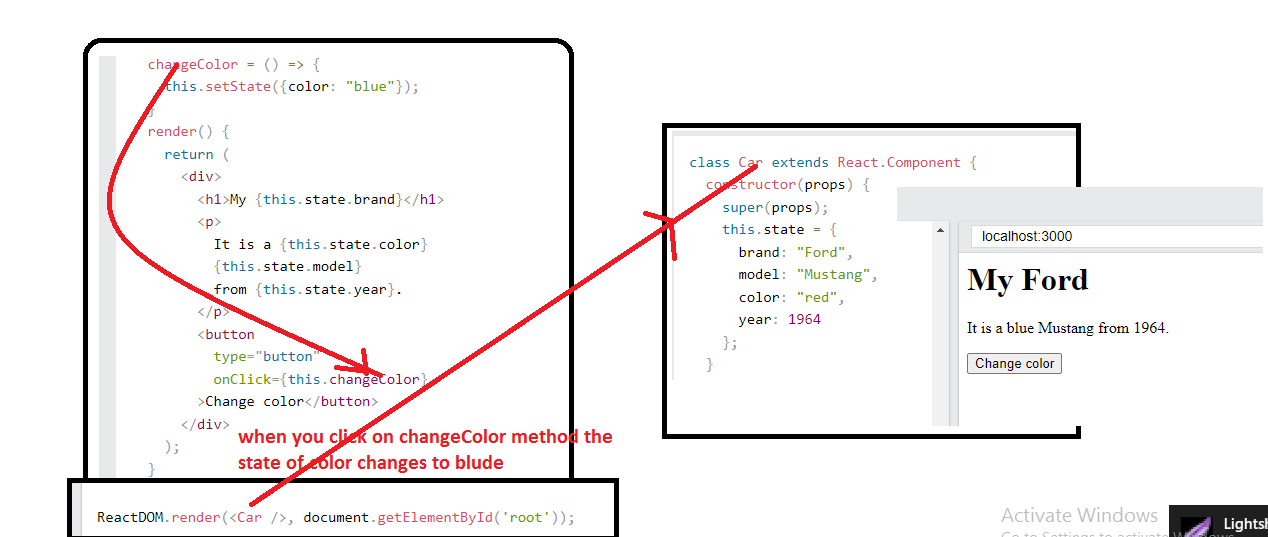


Change the state …..

## Changing the state Object

To change a value in the state object, use the this.setState() method.

When a value in the state object changes, the component will re-render, meaning that the output will change according to the new value(s).



## Lifecycle of Components

Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

The three phases are: **Mounting**, **Updating**, and **Unmounting**.

React has four built-in methods that gets called, in this order, when mounting a component:

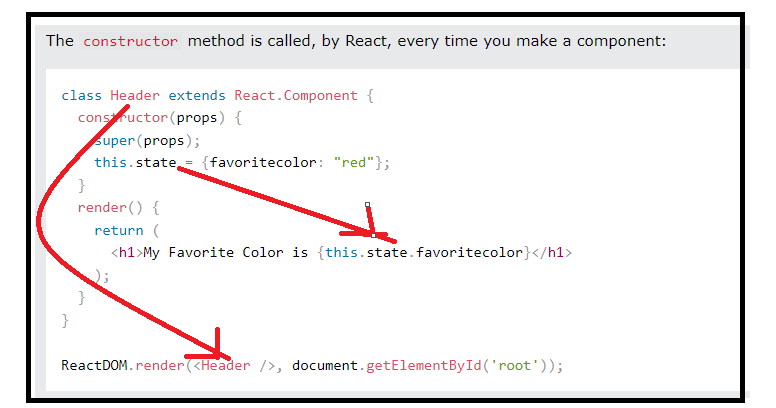
1. constructor()
2. getDerivedStateFromProps()
3. render()
4. componentDidMount()

The render() method is required and will always be called, the others are optional and will be called if you define them.

### constructor

The constructor() method is called before anything else, when the component is initiated, and it is the natural place to set up the initial state and other initial values.

The constructor() method is called with the props, as arguments, and you should always start by calling the super(props) before anything else, this will initiate the parent's constructor method and allows the component to inherit methods from its parent (React.Component).



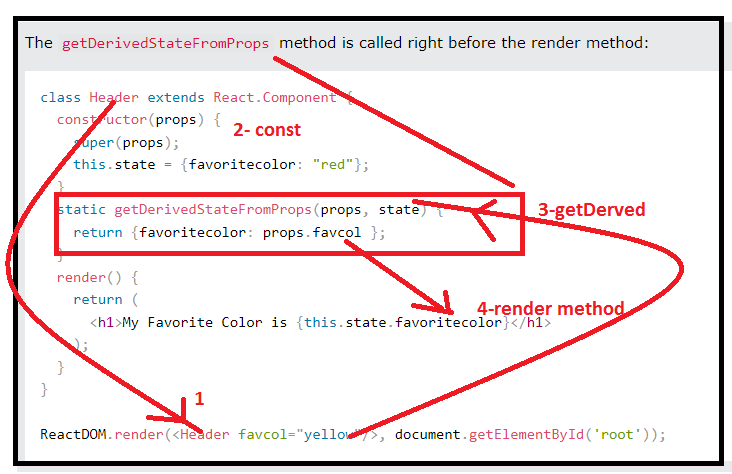
### getDerivedStateFromProps

The getDerivedStateFromProps() method is called right before rendering the element(s) in the DOM.

This is the natural place to set the state object based on the initial props.

It takes state as an argument, and returns an object with changes to the state.

The example below starts with the favorite color being "red", but the getDerivedStateFromProps() method updates the favorite color based on the favcol attribute:



Output



### render

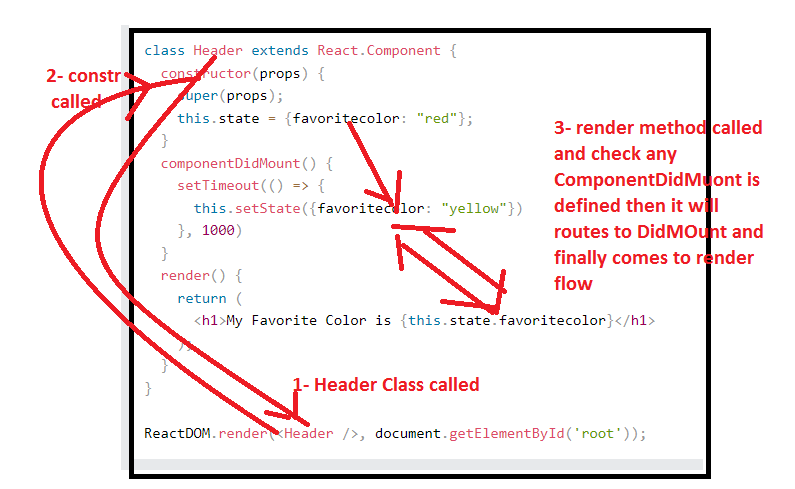
The render() method is required, and is the method that actually outputs the HTML to the DOM.



### componentDidMount

The componentDidMount() method is called after the component is **rendered**.

This is where you run statements that requires that the component is already placed in the DOM.

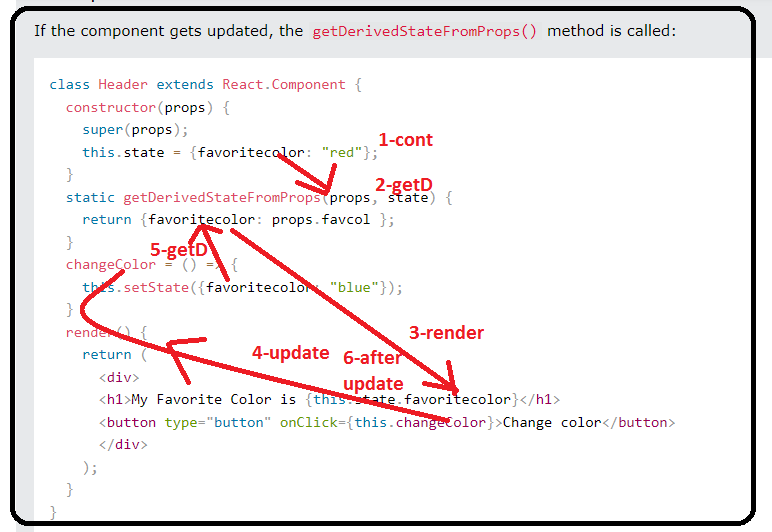


### getDerivedStateFromProps

Also at updates the getDerivedStateFromProps method is called. This is the first method that is called when a component gets updated.

This is still the natural place to set the state object based on the initial props.

The example below has a button that changes the favorite color to blue, but since the getDerivedStateFromProps() method is called, which updates the state with the color from the favcol attribute, the favorite color is still rendered as yellow:



### shouldComponentUpdate

In the shouldComponentUpdate() method you can return a Boolean value that specifies whether React should continue with the rendering or not.

The default value is true.

**If true –it will update or else it will update ….**

The example below shows what happens when the shouldComponentUpdate() method returns false:



### render

The render() method is of course called when a component gets updated, it has to re-render the HTML to the DOM, with the new changes.

The example below has a button that changes the favorite color to blue:



## Pass Data

Props are also how you pass data from one component to another, as parameters.



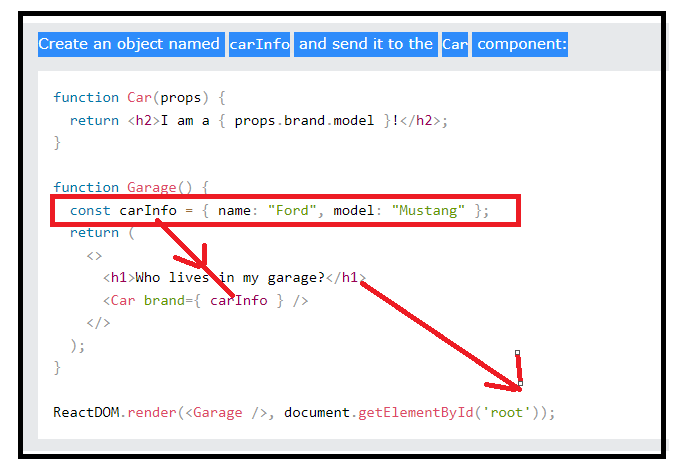
If you have a variable to send, and not a string as in the example above, you just put the variable name inside curly brackets:

### Example

Create a variable named carName and send it to the Car component:



Create an object named carInfo and send it to the Car component:



# React Events

Just like HTML DOM events, React can perform actions based on user events.

React has the same events as HTML: click, change, mouseover etc.

## Adding Events

React events are written in camelCase syntax:

onClick instead of onclick.

React event handlers are written inside curly braces:

onClick={shoot}  instead of onClick="shoot()".

### React:

<button onClick={shoot}>Take the Shot!</button>

### HTML:

<button onclick="shoot()">Take the Shot!</button>

Code

return (

<button onClick={shoot}>Take the shot!</button>

);



Send "Goal!" as a parameter to the shoot function, using arrow function:

return (

<button onClick={() => shoot("Goal!")}>Take the shot!</button>

);

## 

## React Event Object

Event handlers have access to the React event that triggered the function.

In our example the event is the "click" event.

return (

<button onClick={(event) => shoot("Goal!", event)}>Take the shot!</button>

);

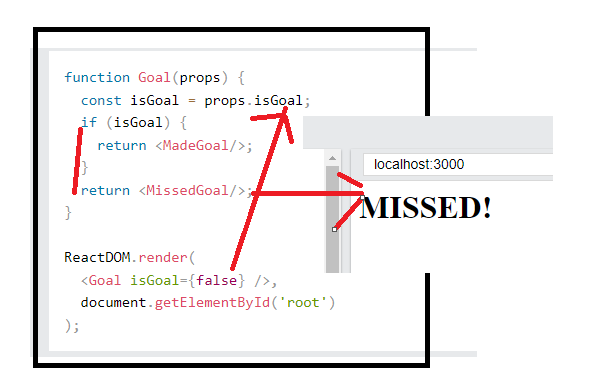


# React Conditional Rendering

## if Statement

We can use the if JavaScript operator to decide which component to render.

Case -1 false



Case 2-

ReactDOM.render(

<Goal isGoal={true} />,

document.getElementById('root')

);

## Logical && Operator

Another way to conditionally render a React component is by using the && operator.

const cars = props.cars;

const cars = ['Ford', 'BMW', 'Audi']

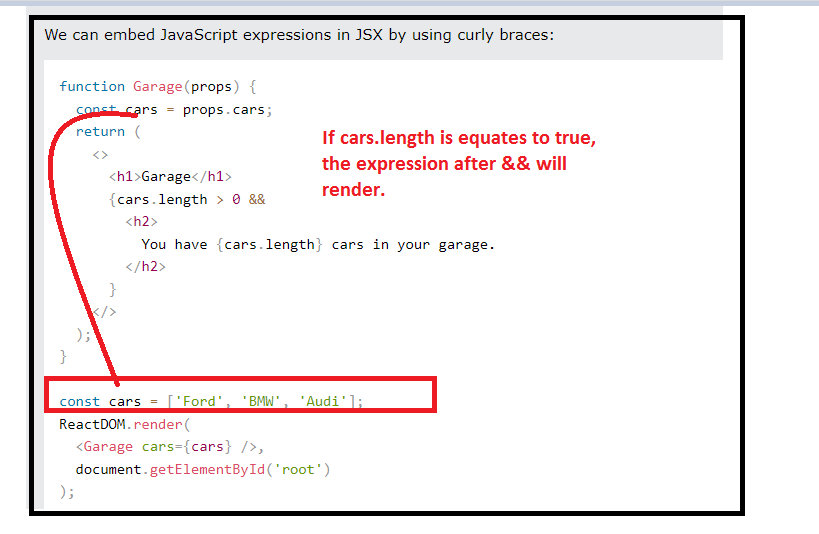
ReactDOM.render(Garage cars={cars} />,document.getElementById('root')

cars.length > 0 &&

<h2>

You have {cars.length} cars in your garage.

</h2>

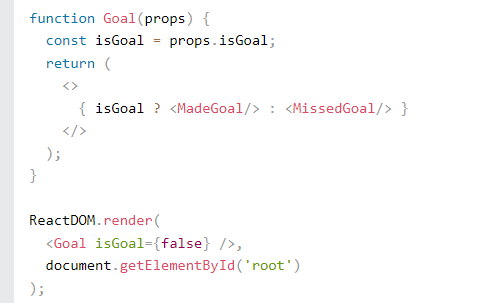


## Ternary Operator

Another way to conditionally render elements is by using a ternary operator.

condition ? true : false

Return the MadeGoal component- if isGoal is true, otherwise return the MissedGoal component:



# React Lists

In React, you will render lists with some type of loop.

The JavaScript map() array method is generally the preferred method.

If you need a refresher on the map() method, check out the [ES6 section](https://www.w3schools.com/REACT/react_es6.asp).

import React from 'react';

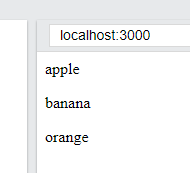
import ReactDOM from 'react-dom';

const myArray = ['apple', 'banana', 'orange'];

const myList = myArray.map((item) => <p>{item}</p>)

ReactDOM.render(myList, document.getElementById('root'));

**Output : -**



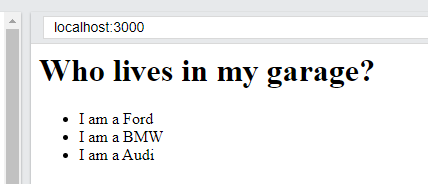
Logic

const cars = ['Ford', 'BMW', 'Audi'];

{cars.map((car) => <Car brand={car} />)}



Output

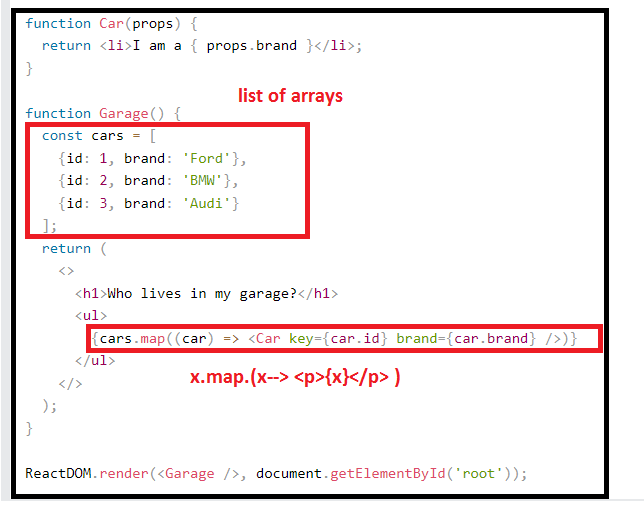


## Keys

Keys allow React to keep track of elements. This way, if an item is updated or removed, only that item will be re-rendered instead of the entire list.

Keys need to be unique to each sibling. But they can be duplicated globally.

Generally, the key should be a unique ID assigned to each item. As a last resort, you can use the array index as a key.



**Forms**

[**https://www.w3schools.com/REACT/react\_forms.asp**](https://www.w3schools.com/REACT/react_forms.asp)

<label>Enter your name:

<input type="text" />

</label>

**output**



**Code**



## Handling Forms

Handling forms is about how you handle the data when it changes value or gets submitted.

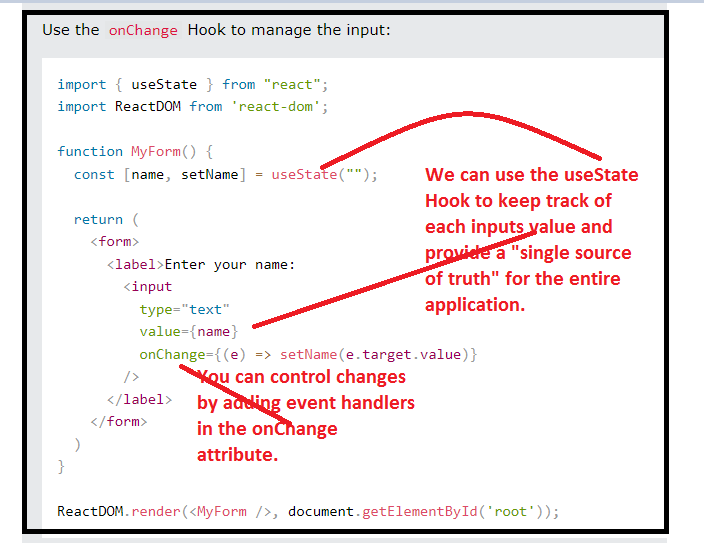
In HTML, form data is usually handled by the **DOM**.

In React, form data is usually handled by the **components**.

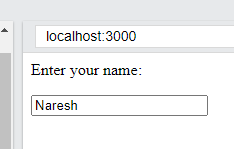
When the data is handled by the components, all the data is stored in the component state.

You can control changes by adding event handlers in the onChange attribute.

We can use the useState Hook to keep track of each inputs value and provide a "single source of truth" for the entire application.



Output



## Submitting Forms

You can control the submit action by adding an event handler in the onSubmit attribute for the <form>:



## Multiple Input Fields

You can control the values of more than one input field by adding a name attribute to each element.

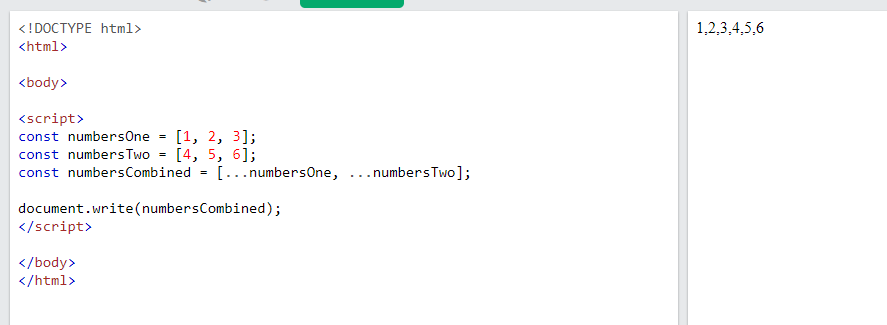
We will initialize our state with an empty object.

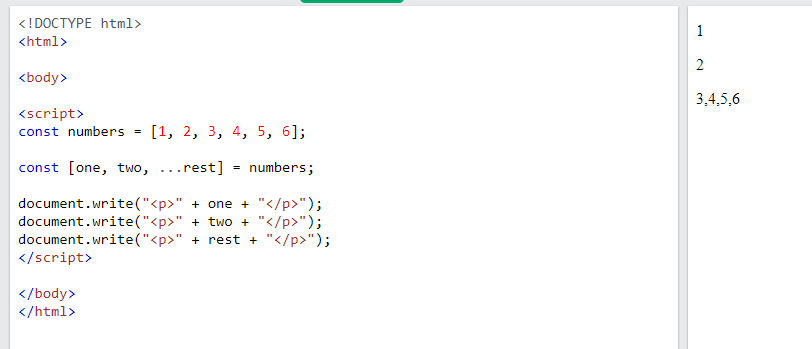
To access the fields in the event handler use the event.target.name and event.target.value syntax.

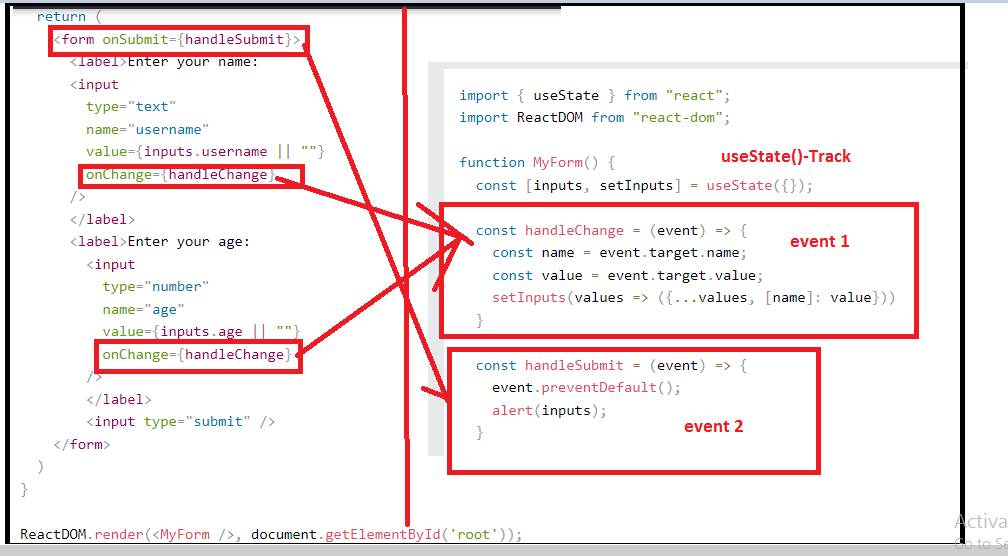
To update the state, use square brackets [bracket notation] around the property name.

**Spread Operator**

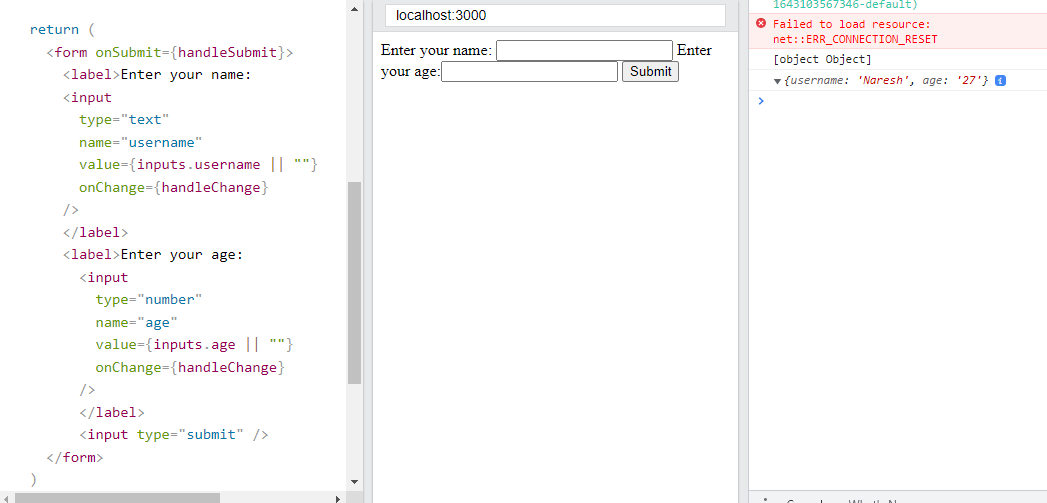
The JavaScript spread operator (...) allows us to quickly copy all or part of an existing array or object into another array or object.





* For submit- we defined handlesubmit –event
* For Onchange(update)-we are defined the handleChange-event. 

On console output



Other conpcets too textarea and select –

<https://www.w3schools.com/REACT/react_forms.asp>

# React Router

<https://www.w3schools.com/REACT/react_router.asp>

<https://www.javaguides.net/2020/09/reactjs-tutorial-for-beginners-14-routing.html>

import { BrowserRouter, Routes, Route } from "react-router-dom";



## Pages / Components

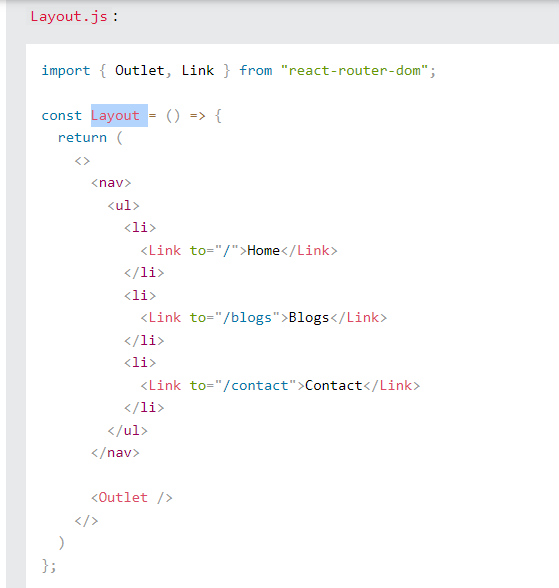
The Layout component has <Outlet> and <Link> elements.

The <Outlet> renders the current route selected.

<Link> is used to set the URL and keep track of browsing history.

Anytime we link to an internal path, we will use <Link> instead of <a href="">.

The "layout route" is a shared component that inserts common content on all pages, such as a navigation menu.



# Configure Routing in App Component

You'll need to import **BrowserRouter**, **Route**, **Link***,* and **Switch**elements from the *react-router-dom* package:

import { BrowserRouter as Router, Route, Switch,Link } from 'react-router-dom';

Now open the *App* component and add the following routing code to it:

import { BrowserRouter as Router, Route, Switch,Link } from 'react-router-dom';

import Home from './components/Home';

import Services from './components/Services';

import About from './components/About';

import Contact from './components/Contact';

function App() {

return (

<Router>

<div>

<h2>React Router Tutorial for Beginners</h2>

<nav>

<ul>

<li><Link to={'/'} > Home </Link></li>

<li><Link to={'/contact'} >Contact</Link></li>

<li><Link to={'/about'} >About</Link></li>

<li><Link to={'/services'} >Services</Link></li>

</ul>

</nav>

<Switch>

<Route path = "/" exact component = {Home}></Route>

<Route path = "/contact" component = {Contact}></Route>

<Route path = "/about" component = {About}></Route>

<Route path = "/services" component = {Services}></Route>

</Switch>

</div>

</Router>

);

}

export default App;

Note that we have provided the *Router* as an alias to the *BrowserRouter*module.

We have added the *Switch* element (open and closing tags). These ensure that only one component is rendered at a time.

<Switch>

<Route path = "/" exact component = {Home}></Route>

<Route path = "/contact" component = {Contact}></Route>

<Route path = "/about" component = {About}></Route>

<Route path = "/services" component = {Services}></Route>

</Switch>

We have defined <Route> tags inside the <Switch> tag to configure the route with its components.

In this <Route> tag, we simply add a **path**attribute and the name of the component you want to load with **component**attribute.

<Route path = "/" exact component = {Home}></Route>

<Route path = "/contact" component = {Contact}></Route>

<Route path = "/about" component = {About}></Route>

<Route path = "/services" component = {Services}></Route>

We have used *Link* for each component in the app and use **to="URL"** to link them:

<nav>

<ul>

<li><Link to={'/'} > Home </Link></li>

<li><Link to={'/contact'} >Contact</Link></li>

<li><Link to={'/about'} >About</Link></li>

<li><Link to={'/services'} >Services</Link></li>

</ul>

</nav>

<https://www.w3schools.com/REACT/react_memo.asp>

# Styling React Using CSS

<https://www.w3schools.com/REACT/react_css_styling.asp>

There are many ways to style React with CSS, this tutorial will take a closer look at three common ways:

* Inline styling
* CSS stylesheets
* CSS Modules

## Inline Styling

To style an element with the inline style attribute, the value must be a JavaScript object:



**Note:** In JSX, JavaScript expressions are written inside curly braces, and since JavaScript objects also use curly braces, the styling in the example above is written inside two sets of curly braces {{}}.

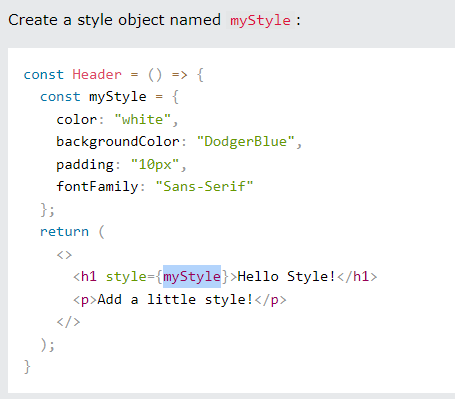
### camelCased Property Names

Since the inline CSS is written in a JavaScript object, properties with hyphen separators, like background-color, must be written with camel case syntax:



### JavaScript Object – Better option..

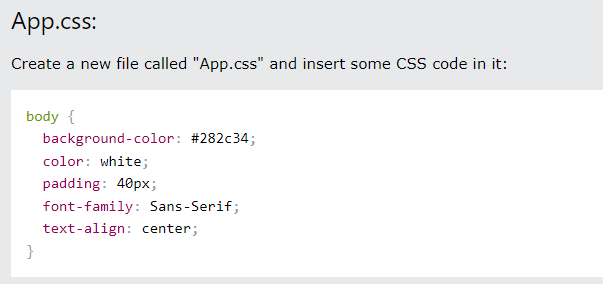
You can also create an object with styling information, and refer to it in the style attribute:



**Prefer to outside**

## CSS Stylesheet

You can write your CSS styling in a separate file, just save the file with the .css file extension, and import it in your application.

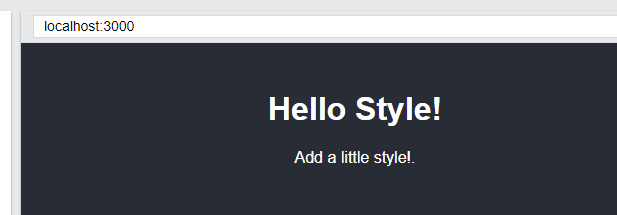


**Note:** You can call the file whatever you like, just remember the correct file extension.

Import the stylesheet in your application:



output



# Styling React Using Sass

## What is Sass

Sass is a CSS pre-processor.

Sass files are executed on the server and sends CSS to the browser.

You can learn more about Sass in our [Sass Tutorial](https://www.w3schools.com/sass/default.php).

<https://www.w3schools.com/sass/default.php>

## What is Sass?

* **Sass** stands for **S**yntactically **A**wesome **S**tyle**s**heet
* Sass is an extension to CSS
* Sass is a CSS pre-processor
* Sass is completely compatible with all versions of CSS
* Sass reduces repetition of CSS and therefore saves time
* Sass was designed by Hampton Catlin and developed by Natalie Weizenbaum in 2006
* Sass is free to download and use

## Why Use Sass?

Stylesheets are getting larger, more complex, and harder to maintain. This is where a CSS pre-processor can help.

Sass lets you use features that do not exist in CSS, like variables, nested rules, mixins, imports, inheritance, built-in functions, and other stuff.

## How Does Sass Work?

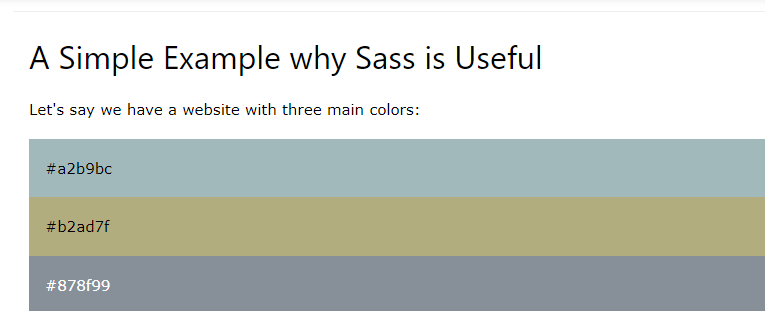
A browser does not understand Sass code. Therefore, you will need a Sass pre-processor to convert Sass code into standard CSS.

This process is called transpiling. So, you need to give a transpiler (some kind of program) some Sass code and then get some CSS code back.

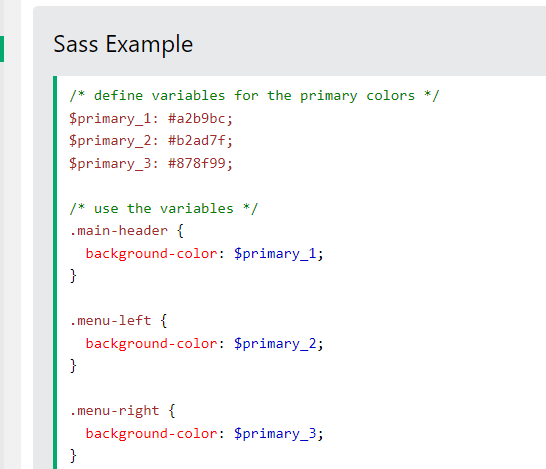
**Tip:** Transpiling is a term for taking a source code written in one language and transform/translate it into another language.

## Sass File Type

Sass files has the ".scss" file extension.



Example



More –

<https://www.w3schools.com/sass/default.php>

## Can I use Sass?

If you use the create-react-app in your project, you can easily install and use Sass in your React projects.

Install Sass by running this command in your terminal:

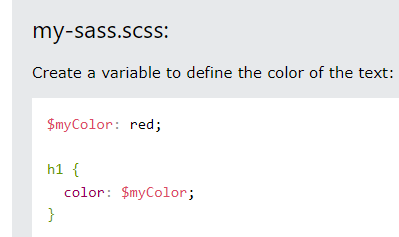
>npm i sass

Now you are ready to include Sass files in your project!

## Create a Sass file

Create a Sass file the same way as you create CSS files, but Sass files have the file extension .scss

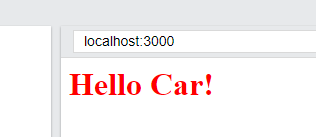
In Sass files you can use variables and other Sass functions:



**Import in the application**



**Output**



**Hooks \*\*\*\*\*\*\*\*\*\*\*\***

<https://www.w3schools.com/REACT/react_hooks.asp>

Hooks were added to React in version 16.8.

Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

Although Hooks generally replace class components, there are no plans to remove classes from React.

## What is a Hook?

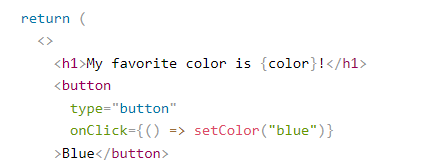
Hooks allow us to "hook" into React features such as state and lifecycle methods.



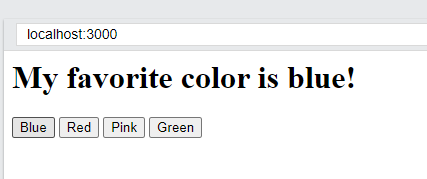
Default by useState is **red**



When you click on Blue.



Whnen you click on button its goest to change the color to Blue.



You must import Hooks from react.

Here we are using the useState Hook to keep track of the application state.

State generally refers to application data or properties that need to be tracked.



## Hook Rules

There are 3 rules for hooks:

* Hooks can only be called inside React function components.
* Hooks can only be called at the top level of a component.
* Hooks cannot be conditional

**Note:** Hooks will not work in React class components.

## Custom Hooks

If you have stateful logic that needs to be reused in several components, you can build your own custom Hooks.

We'll go into more detail in the [Custom Hooks section](https://www.w3schools.com/REACT/react_customhooks.asp).

<https://www.w3schools.com/REACT/react_customhooks.asp>

# React useState Hook

The React useState Hook allows us to track state in a function component.

State generally refers to data or properites that need to be tracking in an application.

## Import useState

To use the useState Hook, we first need to import it into our component.

### Example:

At the top of your component, import the useState Hook.

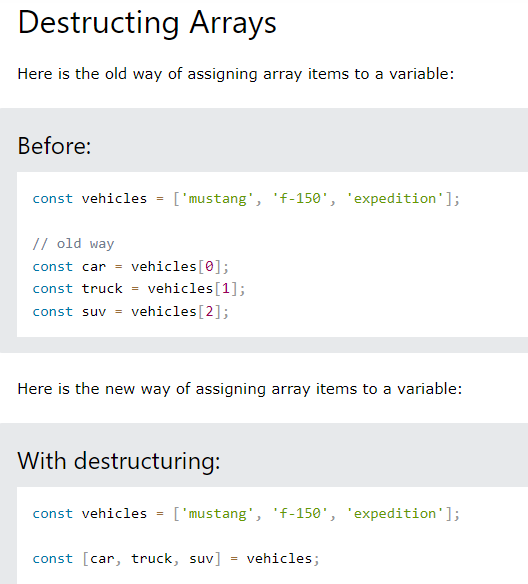
import { useState } from "react";

Notice that we are destructuring useState from react as it is a named export.

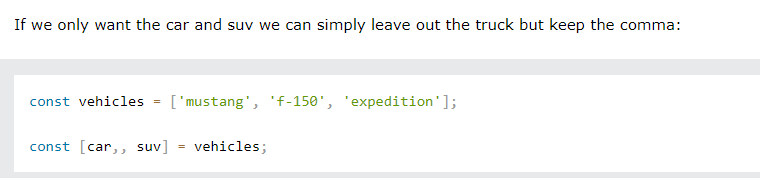
<https://www.w3schools.com/REACT/react_es6_destructuring.asp>

Destructuring makes it easy to extract only what is needed.

Destructuring is exactly the same. We may have an array or object that we are working with, but we only need some of the items contained in these.



When destructuring arrays, the order that variables are declared is important.



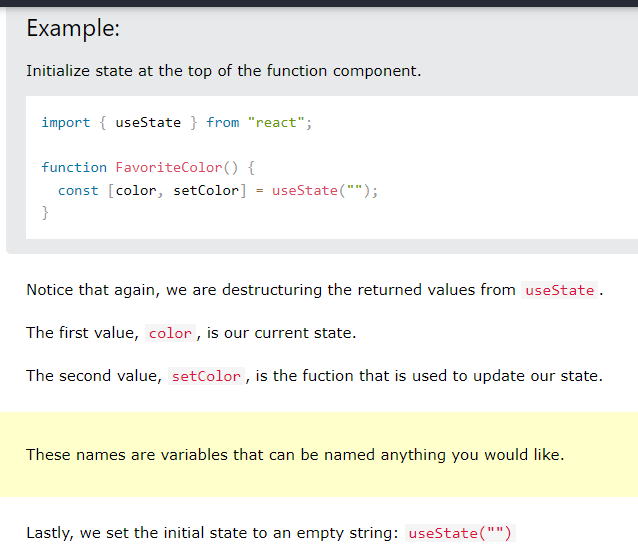
# Lets come to topic useState Hook

## Initialize useState

We initialize our state by calling useState in our function component.

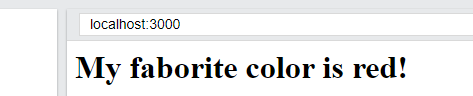
useState accepts an initial state and returns two values:

* The current state.
* A function that updates the state.



Lets assign the value to the color through the useState(red)

output



## Update State

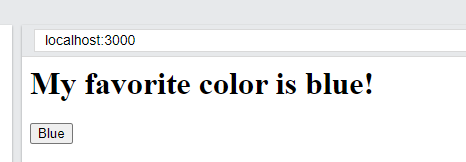
To update our state, we use our state updater function.

We should never directly update state. Ex: color = "red" is not allowed.

We achieve by button onClick.



Output –after click



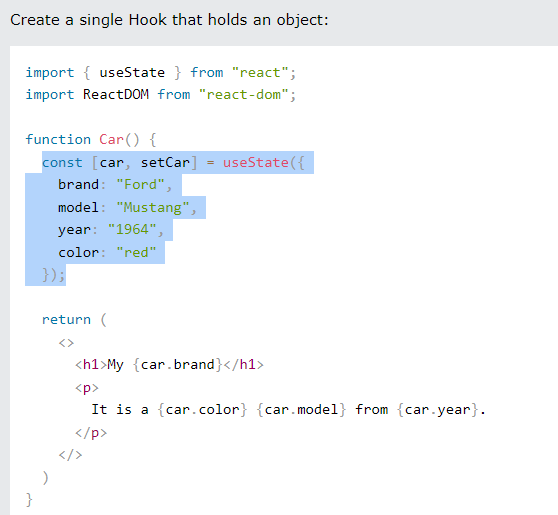
## What Can State Hold

The useState Hook can be used to keep track of strings, numbers, booleans, arrays, objects, and any combination of these!

We could create multiple state Hooks to track individual values.



Or, we can just use one state and include an object instead!



## Updating Objects and Arrays in State

When state is updated, the entire state gets overwritten.

What if we only want to update the color of our car?

If we only called setCar({color: "blue"}), this would remove the brand, model, and year from our state.

const updateColor = () => {

setCar(previousState => {

return { ...previousState, color: "blue" }

or

return { ...previousState, brand: "br" }

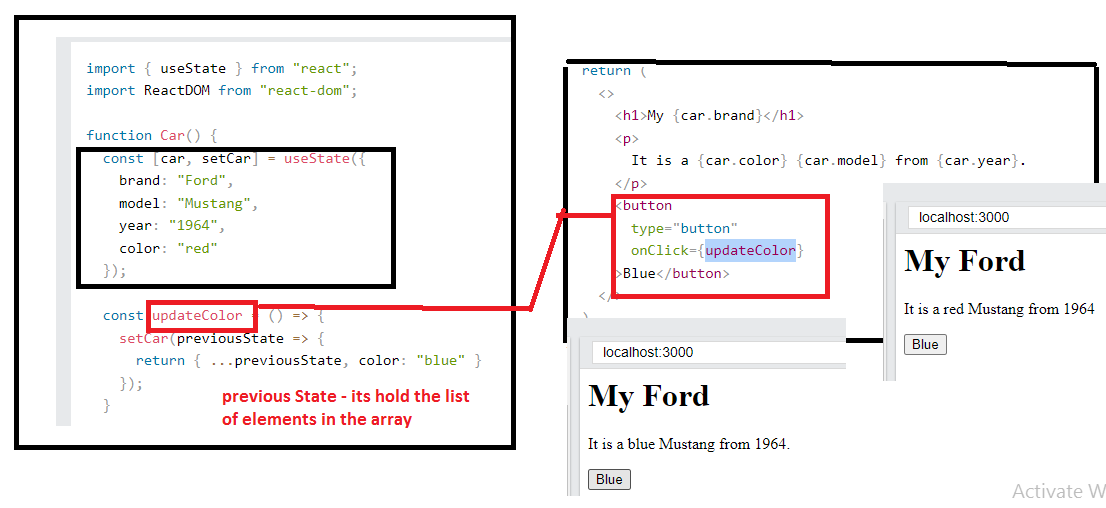
or

return { ...previousState, year: "2020 " }

});

}

We can use the JavaScript spread operator to help us.



# React useEffect Hooks

The useEffect Hook allows you to perform side effects in your components.

Some examples of side effects are: fetching data, directly updating the DOM, and timers.

useEffect accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

Use setTimeout() to count 1 second after initial render:

import { useState, useEffect } from "react";



But wait!! I keeps counting even though it should only count once!

useEffect runs on every render. That means that when the count changes, a render happens, which then triggers another effect.

This is not what we want. There are several ways to control when side effects run.

We should always include the second parameter which accepts an array. We can optionally pass dependencies to useEffect in this array.

### 1. No dependency passed:

useEffect(() => {

//Runs on every render

});

This code

useEffect(() => {

setTimeout(() => {

setCount((count) => count + 1);

}, 1000);

});

### 2. An empty array:

useEffect(() => {

//Runs only on the first render

}, []);

### 3. Props or state values:

useEffect(() => {

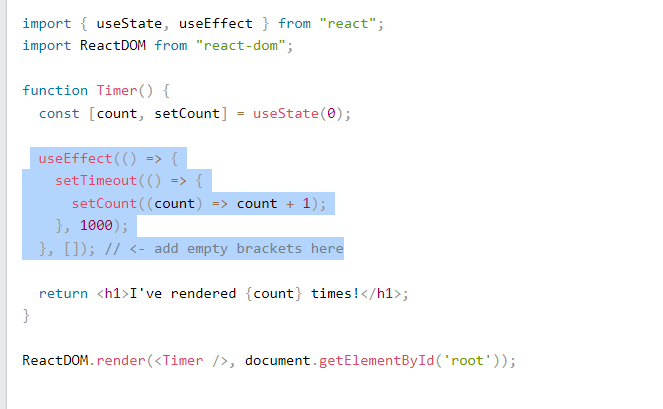
//Runs on the first render

//And any time any dependency value changes

}, [prop, state]);

So, to fix this issue, let's only run this effect on the initial render.

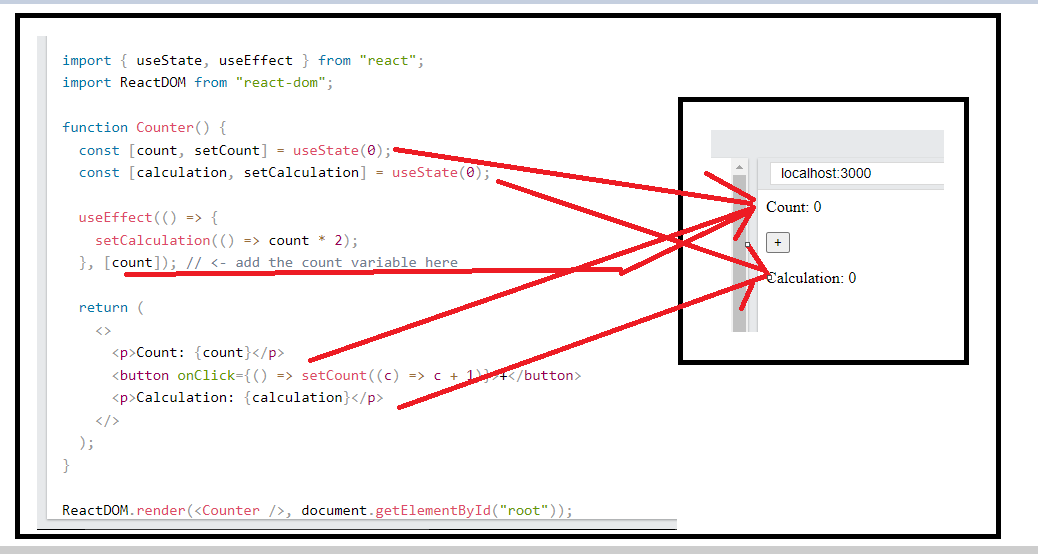
//Runs only on the first render for [] empty array



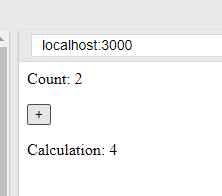
Output



Here is an example of a useEffect Hook that is dependent on a variable. If the count variable updates, the effect will run again:



Output



## Effect Cleanup

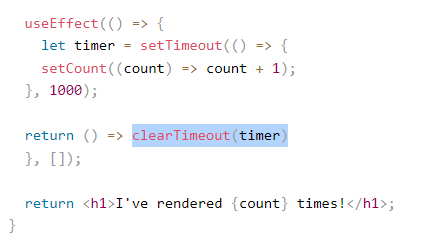
Some effects require cleanup to reduce memory leaks.

Timeouts, subscriptions, event listeners, and other effects that are no longer needed should be disposed.

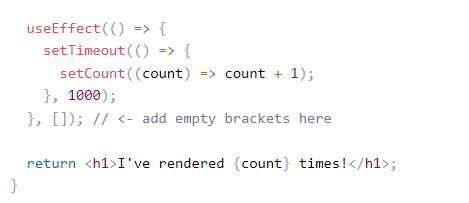
We do this by including a return function at the end of the useEffect Hook.



Code



Below code is same as above // Only run the effect on the initial render:



## React Context

## <https://www.w3schools.com/REACT/react_usecontext.asp>

React Context is a way to manage state globally.

It can be used together with the useState Hook to share state between deeply nested components more easily than with useState alone.

## The Problem

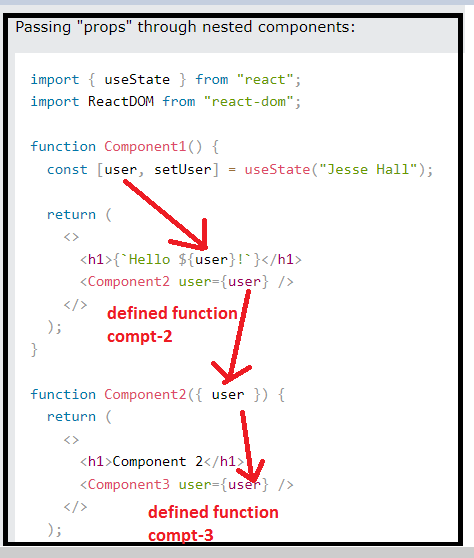
State should be held by the highest parent component in the stack that requires access to the state.

Eg :- useState("Jesse Hall");

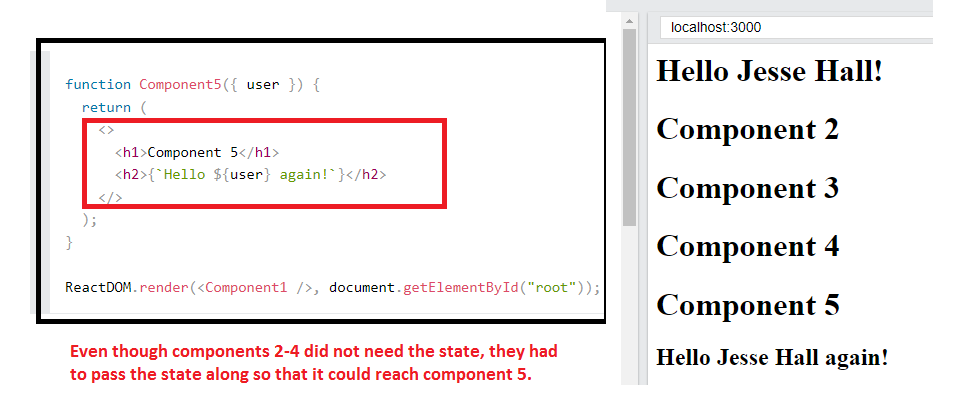
To illustrate, we have many nested components. The component at the top and bottom of the stack need access to the state.

To do this without Context, we will need to pass the state as "props" through each nested component. This is called "prop drilling".

Passing "props" through nested components:



Even though components 2-4 did not need the state, they had to pass the state along so that it could reach component 5.



Even though components 2-4 did not need the state, they had to pass the state along so that it could reach component 5.

## The Solution

The solution is to create context.

### Create Context

To create context, you must Import createContext and initialize it:

**Step 1**



Step 2

Next we'll use the Context Provider to wrap the tree of components that need the state Context.

### Context Provider

Wrap child components in the Context Provider and supply the state value.



Step 3

Now, all components in this tree will have access to the user Context.

### Use the useContext Hook

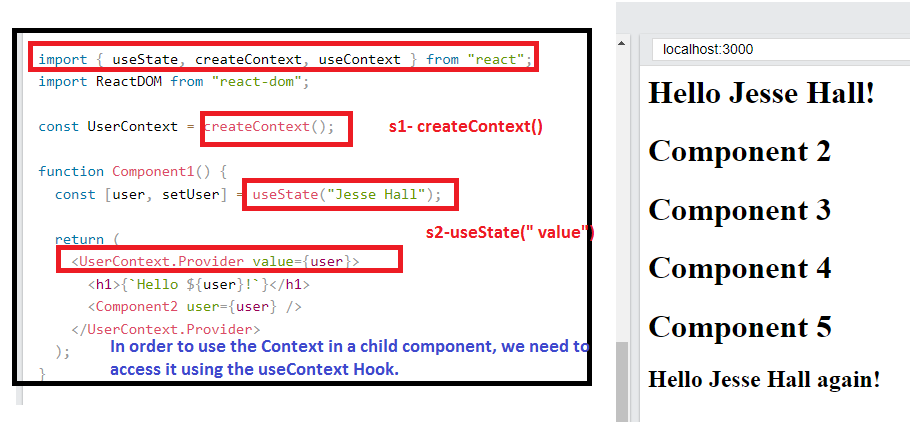
In order to use the Context in a child component, we need to access it using the useContext Hook.

First, include the useContext in the import statement:

import { useState, createContext, useContext } from "react";



Flow ->





# React useRef Hook

<https://www.w3schools.com/REACT/react_useref.asp>

The useRef Hook allows you to persist values between renders.

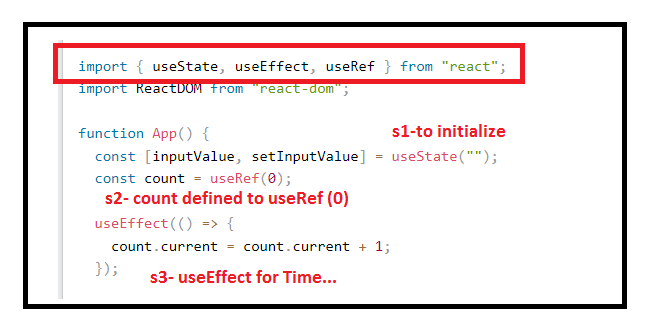
It can be used to store a mutable value that does not cause a re-render when updated.

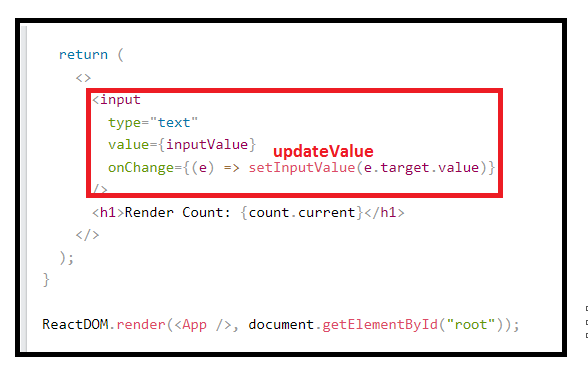
It can be used to access a DOM element directly.

## Does Not Cause Re-renders

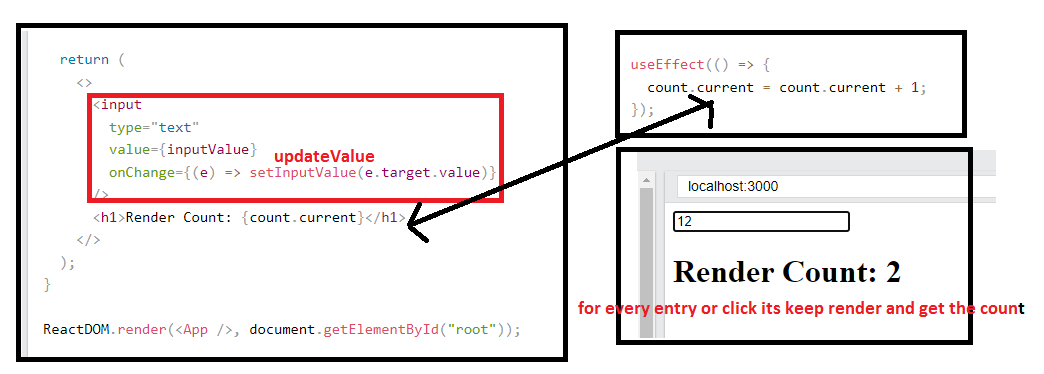
If we tried to count how many times our application renders using the useState **Hook, we would be caught in an infinite loop since this Hook itself causes a re-render.**

To avoid this, we can use the useRef Hook.

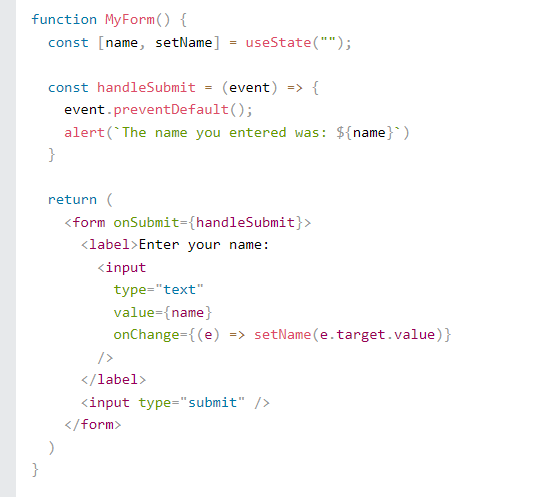




**Flow :-**

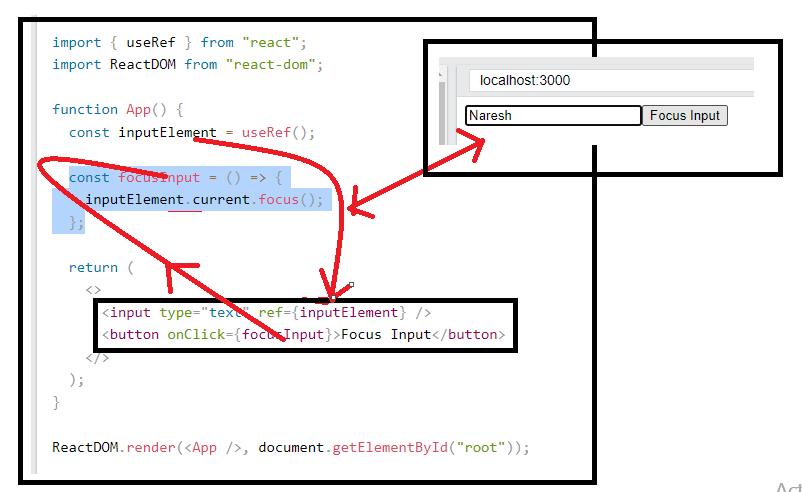


Same code in the **useState** by update by submit button handle by the event-handleSubmit



## Accessing DOM Elements

* In general, we want to let React handle all DOM manipulation.
* But there are some instances where useRef can be used without causing issues.
* In React, we can add a ref attribute to an element to access it directly in the DOM.

.

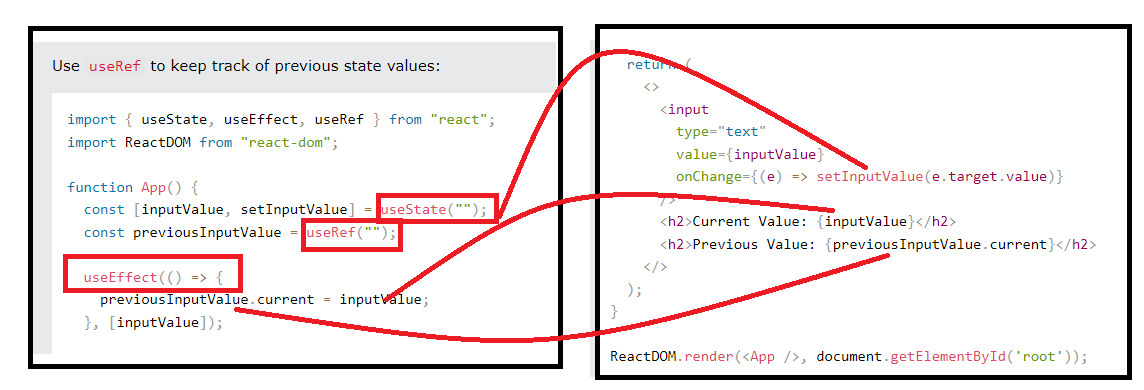
## Tracking State Changes

The useRef Hook can also be used to keep track of previous state values.

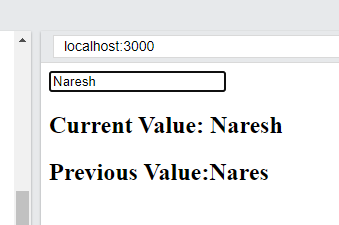
This is because we are able to persist useRef values between renders.

This time we use a combination of useState, useEffect, and useRef to keep track of the previous state.

In the useEffect, we are updating the useRef current value each time the inputValue is updated by entering text into the input field.



Output



# React useReducer Hook

<https://www.w3schools.com/REACT/react_usereducer.asp>

* The useReducer Hook is similar to the useState Hook.
* It allows for custom state logic.
* If you find yourself keeping track of multiple pieces of state that rely on complex logic, useReducer may be useful.

## Syntax

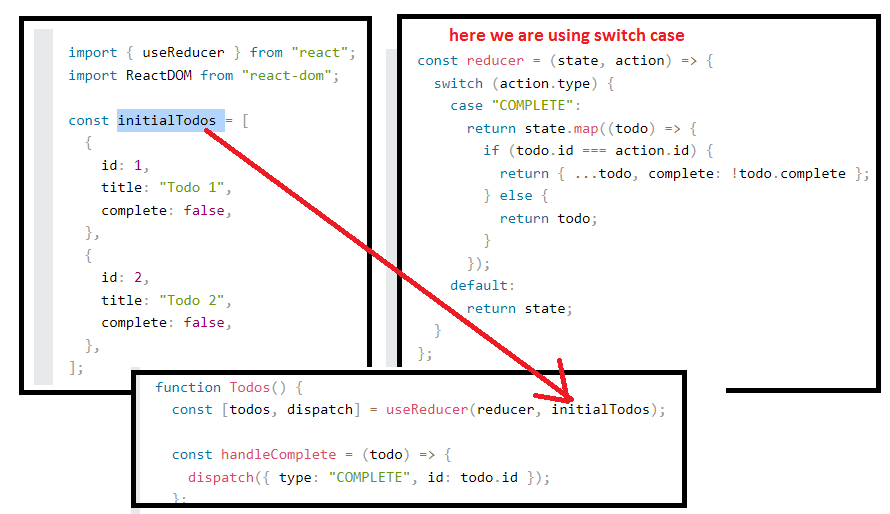
The useReducer Hook accepts two arguments.

useReducer(<reducer>, <initialState>)

The reducer function contains your custom state logic and the initialStatecan be a simple value but generally will contain an object.

The useReducer Hook returns the current stateand a dispatchmethod.

Here is an example of useReducer in a counter app:



Output:-



This is just the logic to keep track of the todo complete status.

All of the logic to add, delete, and complete a todo could be contained within a single useReducer Hook by adding more actions.

# React Custom Hooks

Hooks are reusable functions.

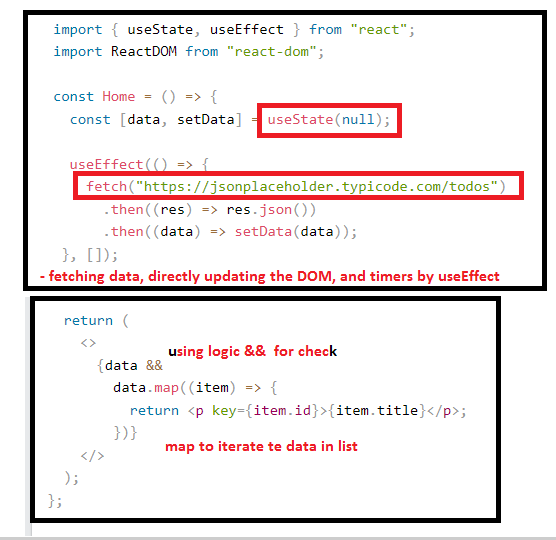
When you have component logic that needs to be used by multiple components, we can extract that logic to a custom Hook.

Custom Hooks start with "use". Example: useFetch.

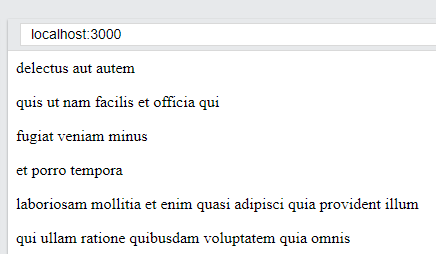
So, how to build a custom Hook.

In the following code, we are fetching data in our Home component and displaying it. (which is used for other components).

We will use the [JSONPlaceholder](https://jsonplaceholder.typicode.com/) service to fetch fake data. This service is great for testing applications when there is no existing data.



Ouptput



The fetch logic may be needed in other components as well, so we will extract that into a custom Hook.

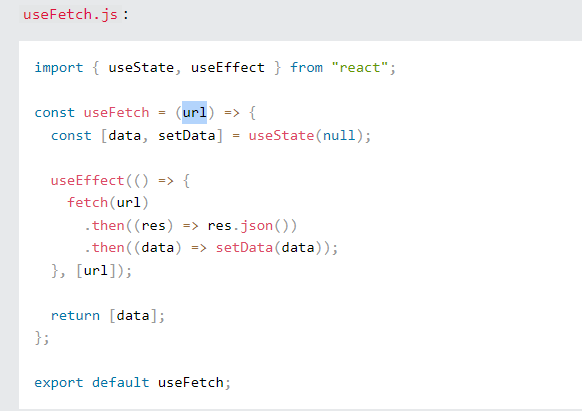
Move the fetch logic to a new file to be used as a custom Hook:

Step 1-

We have created a new file called useFetch.js containing a function called useFetch which contains all of the logic needed to fetch our data.

We removed the hard-coded URL and replaced it with a url variable that can be passed to the custom Hook.

Lastly, we are returning our data from our Hook.



Step 2-

In index.js, we are importing our useFetch Hook and utilizing it like any other Hook. This is where we pass in the URL to fetch data from.

Now we can reuse this custom Hook in any component to fetch data from any URL.



Flow

