**Codevolution**

**Component Based Architecture**

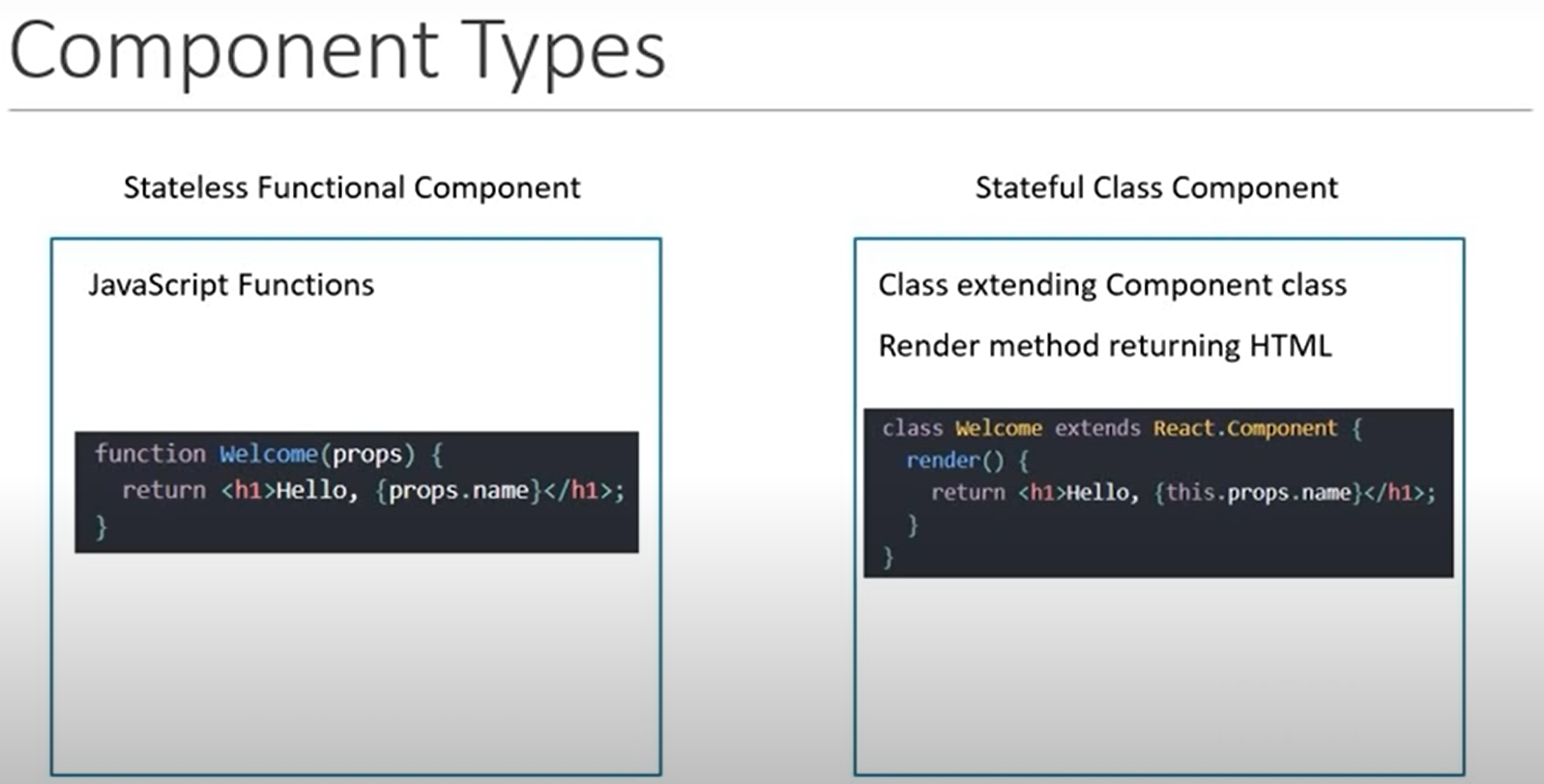
* Header
* Sidenav
* Main content
* Footer

Components:

Components describe a part of the user interface.

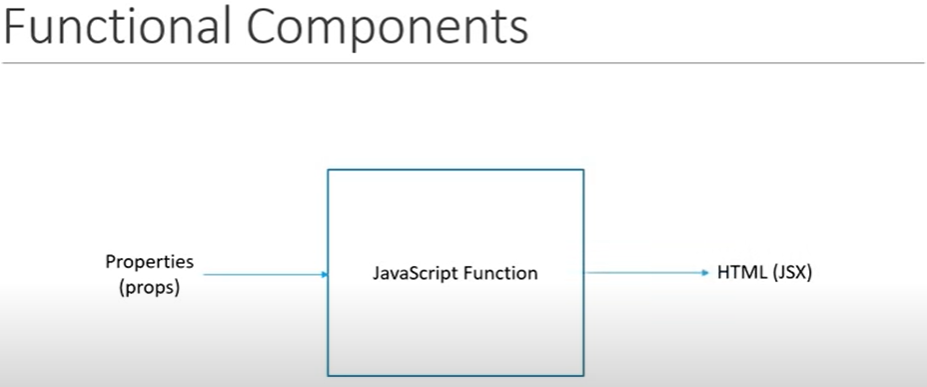
They are re-usable and can be nested inside other components.

Two Types:



**Function Components:**

Functional components are just javascript functions they can optionally receive an object of properties which referred to as props and return HTML. Which describes the UI now the HTML known as JSX



export default Greet;

Exporting and importing components that we are exporting Greet component as default export from Greet.js

**Import Greet from ‘. /components/Greet’;**

**Import MyComponent from ‘. /components/Greet’;**

this is what allows us to import the component with any name. Like MyComponent it will work fine.

**Named Export:**

**export const Greet = () => <h1>Hello </ h1>** This called Named Export. In this situation you have to import the component with the exact same name.

**Import {Greet} from ‘./components/Greet’;**

Class Components:

Class components are basically ES6 classes similar to a functional component a class component also can optionally receive props as input and return HTML apart from the props a class component can have maintain a private internal state. It can maintain some information which is private to that component and use that information describe the user interface.

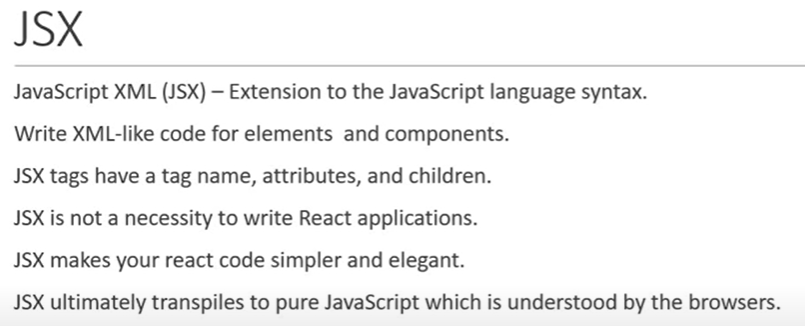
**Render Method:**

Render method which will return null or some HTML

**What is advantage of class components and functional components when should be use?**

|  |  |
| --- | --- |
| **Class** | **Functional** |
| More feature rich.  Maintain their own private data – state.  Complex UI logic.  Provide lifecycle Hooks.  stateful/Smart/Container | Simple function receiving props and returning a declaration.  Use Func components as much as possible. Possible to crate both the approaches always go with the function components.  Absence of ‘this’ keyword.  Solution without using state.  Mainly responsible for the UI  stateless/Dumb/presentational |

**JSX:**



**React.createElement:**

**React.createElement (‘div’, null, Hello Murali)**

React.createElement method at minimum accepts three parameters.

1st parameter is a string which specifies HTML tag to rendered like ‘div’ tag to rendered.

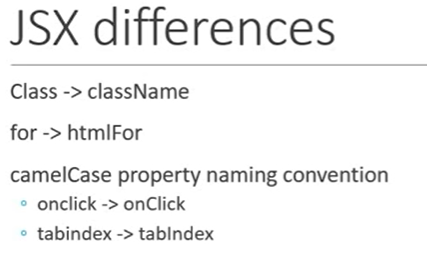
2nd parameter we get to pass any optional properties. Right now, we no need pass any additional properties so we can pass value of ‘null’.

3rd parameter is children for HTML element that is children for the ‘div’ tag I'm passing Hello Murali. For its not applied for h1 tag then we have to 3rd parameter

**React.createElement (‘div’, null,** **React.createElement (‘h1’, null, Hello Murali));** this right way for this

**Why React is Imported?**

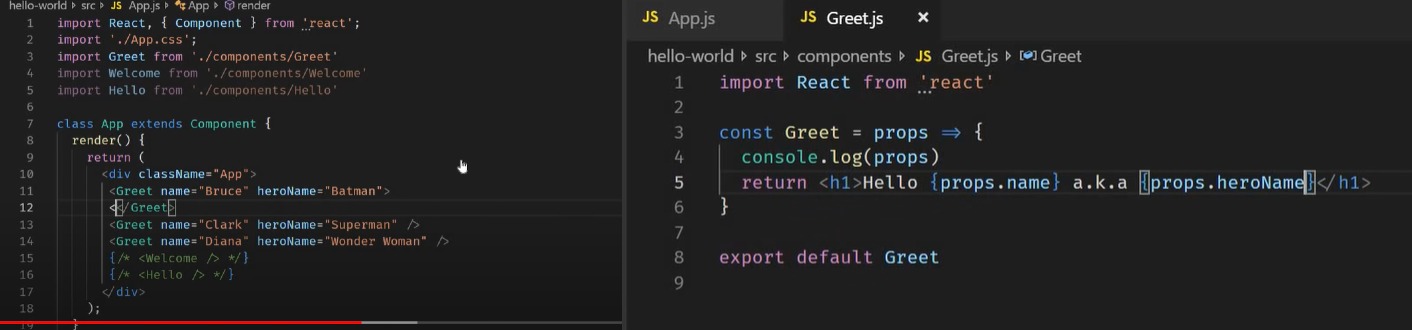
JSX is translates into React.createElement which in turn uses the react library.



**Props:**

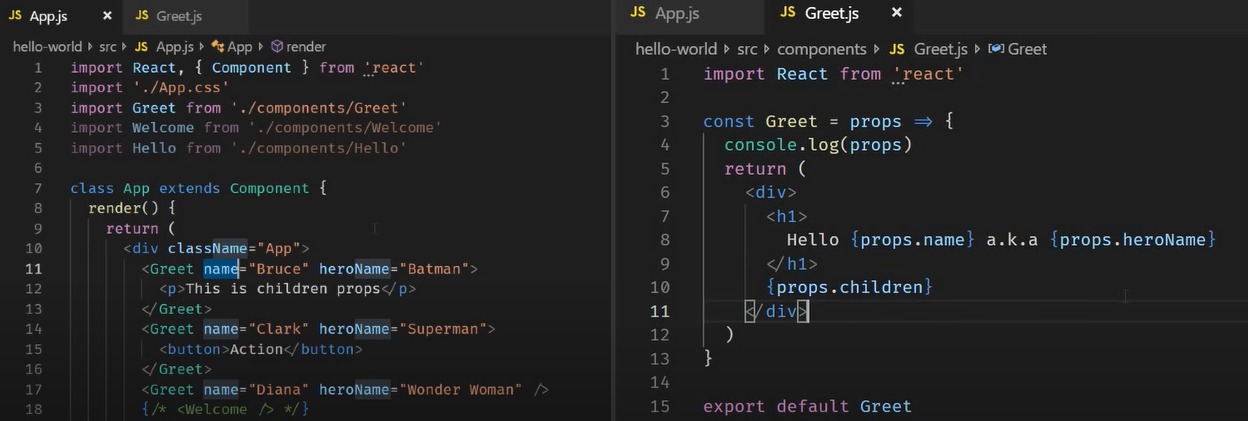
Props is an object that contains the attributes and their values which have been passed from the parent component. So, I have to display the name that have to passed to the greet component we need to use **props.name** in functional components. We can access class components like **this.props.name**. Props are immutable their value can’t be changed.

**Props using functional component:**

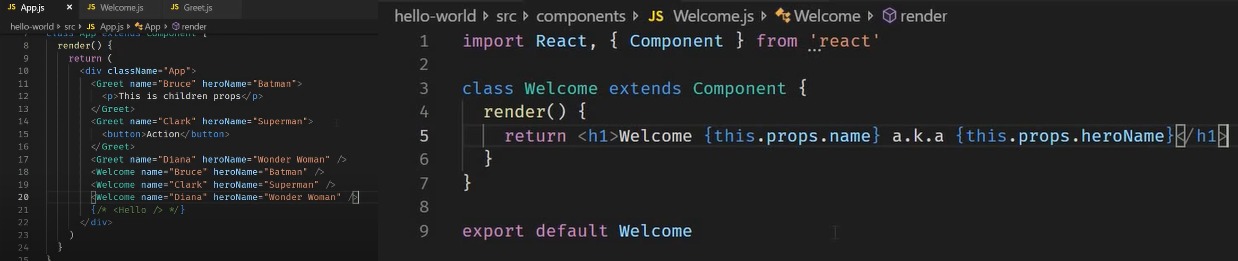


**Note: Props are immutable how do we maintain components data that might change over time?**

Sometimes it is also possible that you might not have an idea as to what content is being passed in but we want the component to render that unknown content we can do the specifying the content between opening and closing tag of the component and retrieving it using the reserved children property in the props object.



**Props using in class component:**

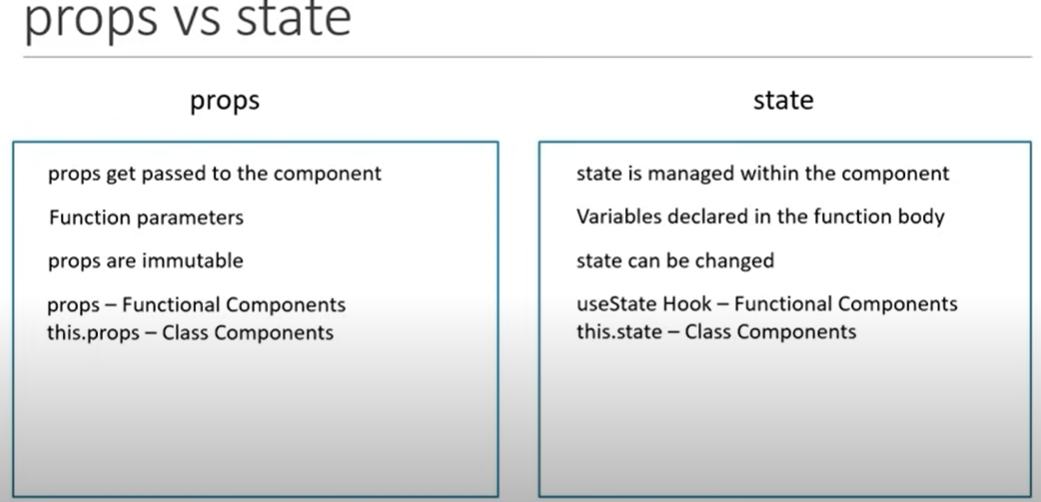


**State:**

What is rendered on the screen using props as it turns out there is a second way to influence what is rendered on the screen and that is the state of the component.

Create state object and initialize it and usually done inside the **class constructor**. Within the constructor we call the **super method**. This is required because we extend react component class and a call have to be made to the base class constructor. Then we create out state object.

State is an object is privately maintained inside a component a state can influence what is rendered in the browser. State can change within the component



**setState:**

When we have to change the state of the component then we use setState. What will happen if we don’t use it and try to change state directly. React will not re-render the component

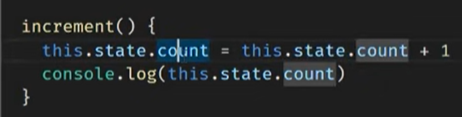
**Example:** I have counter button clicking on that button I need to increment 1 value. I will print that value in console clicking that button increment 1 will working in the console but UI didn’t change. What this means is that the UI is not re-rendering whenever the state changing and this the main reason, we should never modify the state directly.

The only place where you can assign this.state is the constructor any other time to change the state setSate method has to be used.

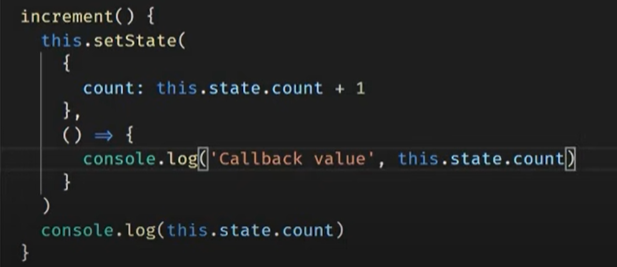
I’m updating the state value directly

**Do’s Don’ts:**

State never modifies the state directly instead make use of setState. When we modify the state directly react will not re-rendered the component. SetState on the other hand will let react now it has to re-render the component.



Note: in the count value clicking increment button 1 increment is working fine but in console value is 0, console value 1 less than the rendered value and this is calls the setState are asynchronous.

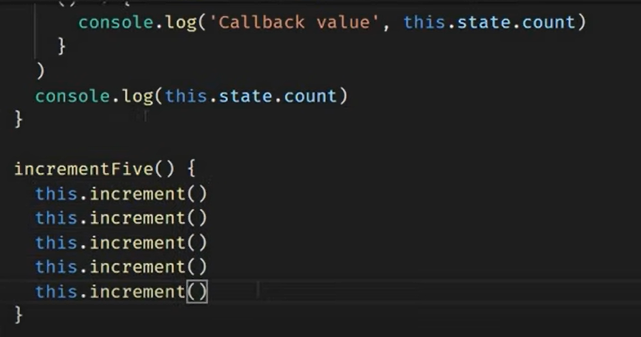


What is happening is console here is being called before the state is actually set many times in your application you might want to execute some code only after the state has been updated to handle such a situation you can pass in a callback function as the second parameter to the set state method setSate method have two parameters 1st parameter is state object 2nd parameter is call back function

0 is synchronous console.log statement.

1 is the callback function statement.

Whenever you need excute some code after the state has been changed do not place the code right after the setState method instead place the code within the callback function that is passed as 2nd parameter to the setstate method

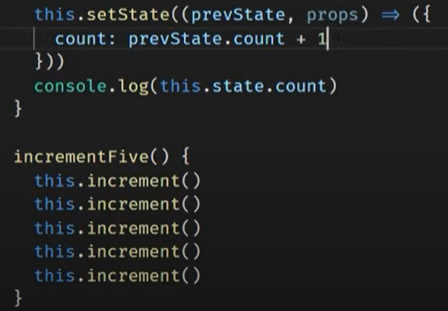


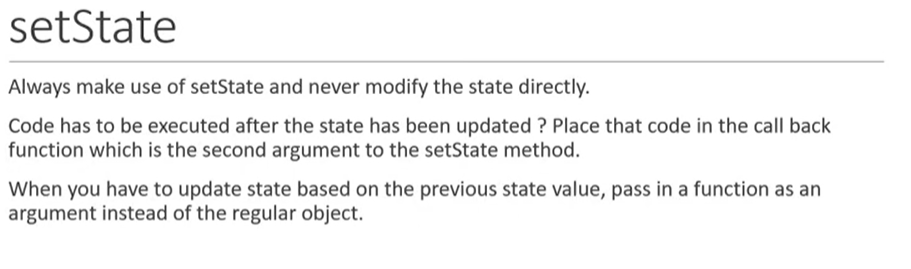
Used to current state to calculate the new state value.

Clicking on the button value changes 1 instead of 5. in the console 0 prints 5times and callback value 1 print 5 times, because react grouped multiple setState call into a single update for better performance.

All the five setState calls are done in one single go and the updated value does not carryover between different calls so whenever you wants update the state based on the pervious state, we need to pass a function as an argument to setState method instead of passing in an object.

When we have to update the state based on the previous state make sure to pass in a function as an argument instead of the regular object the function access to the previous state which can be used to calculate the new state. As it turns out the second parameter to this function is the props object.





**Destructuring props and state**

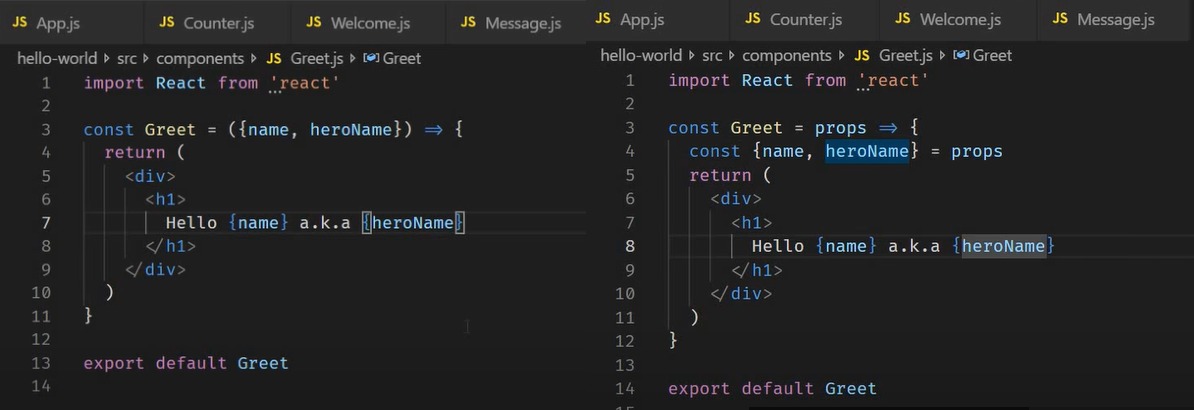
Destructuring is ES6 feature that makes it possible unpack values from arrays or properties from objects into distinct variables. In react destructuring props and state improves code readability.

**Destructuring props and state using in functional component:**

Ther are two ways to destructure props in functional component.

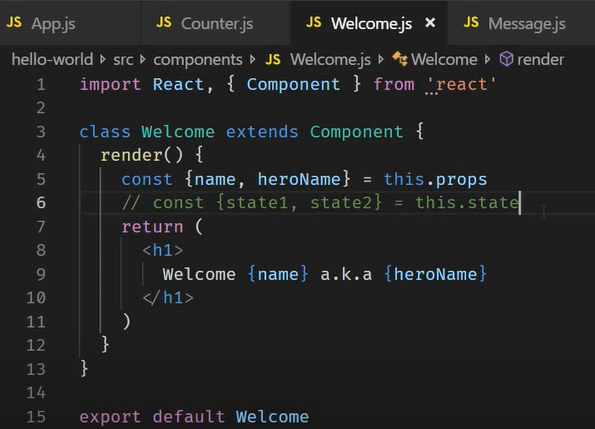
1st way destructure it in the function parameter itself.

2nd way destructure in the function body.



**Destructuring props and state using in class component:**

Destructure props and state we use in the render method

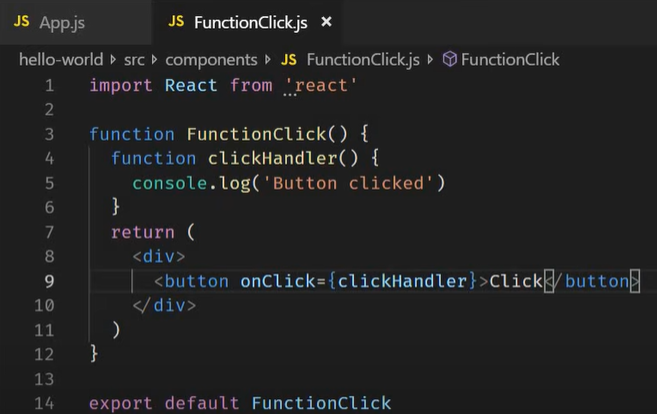


**Event Handling**

User interaction with our application events fired Example: mouse clicks mouse over key press change event etc. Application must handle such event and execute the necessary code.

1. React events are named using camel case rather than lowercase in plan HTML and vanilla js you would specify **onclick** in react it has to be camel cased **onClick.**
2. With JSX you pass a function as the event handler rather than string

**Event Handling using in functional component:**



OnClick we pass the function as the event handler close attention to the absence of parentheses if we add parentheses, **clickHnandler()** it become its function call. We want the handler to be a function and not a function call.

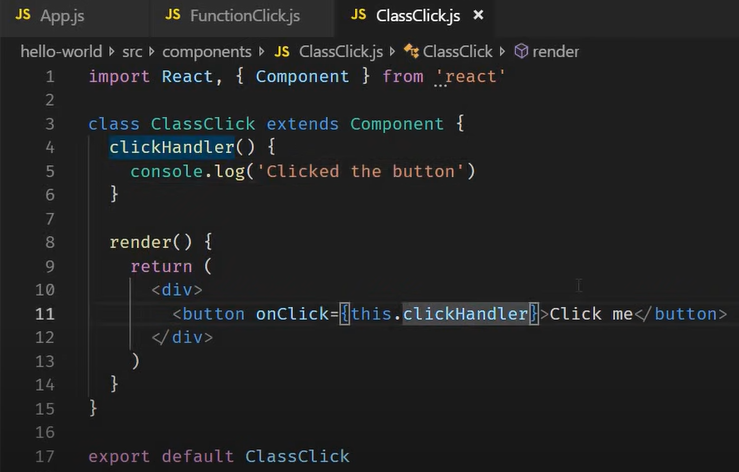
**clickHnandler()** in the console already that message been displayed after clicking the button, we didn’t see any message

**clickHnandler** if we do leave the parentheses this correct way.

Now this scenario become reverse in class components when click handler change the state of the component the component constantly re-render.

Event handler is a function and not a function call do not add parentheses.

**Event Handling using in class component:**

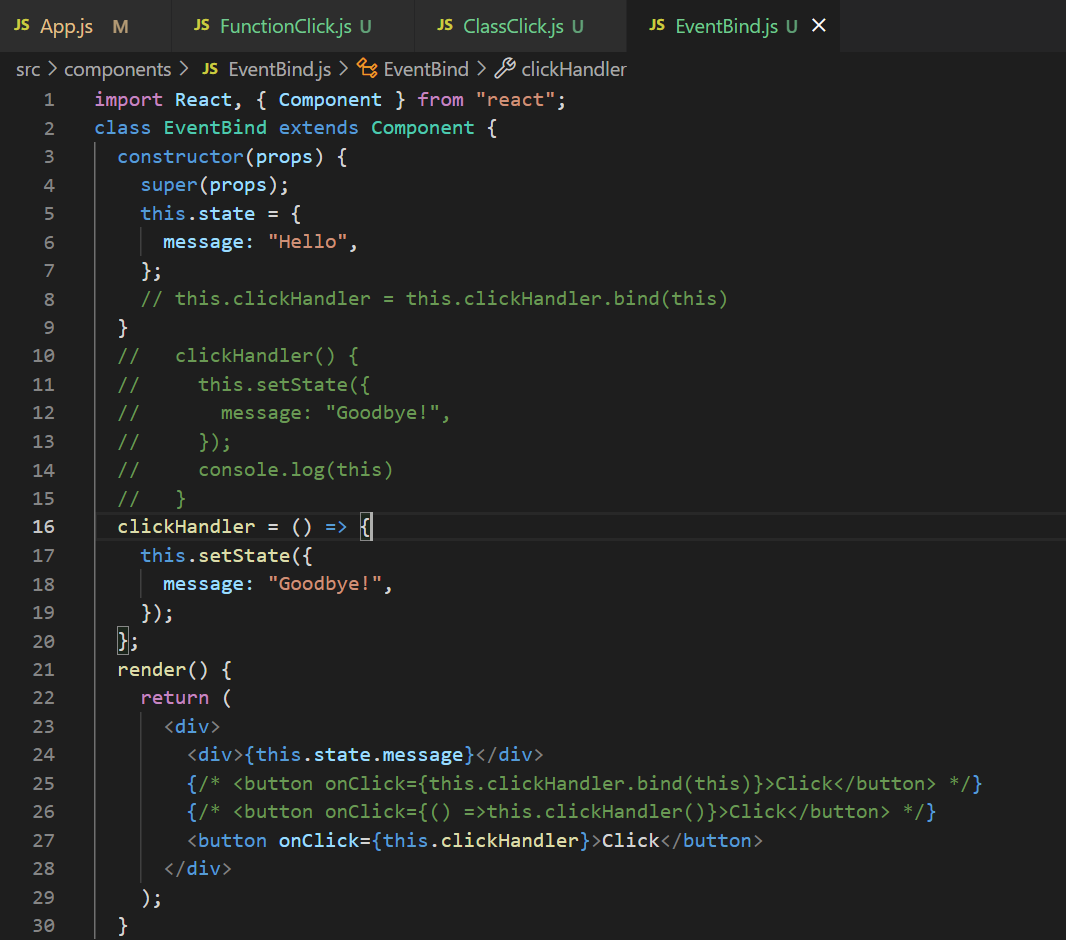


**Binding Event Handlers:**

Why need to bind event handlers in react. Purely because of the way this keyword in js it is not because of how react works

We have four ways bind methods.

1. Bind in the render method.
2. Arrow function in the render method simply calling the event handler in the arrow function body.
3. Binding event handler in the constructor.
4. Using arrow function in the class property.



**Methods as props:**