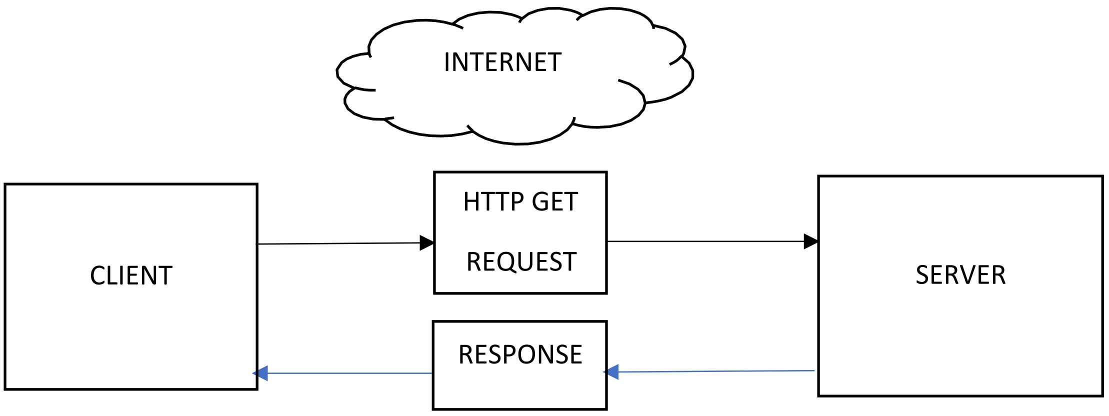
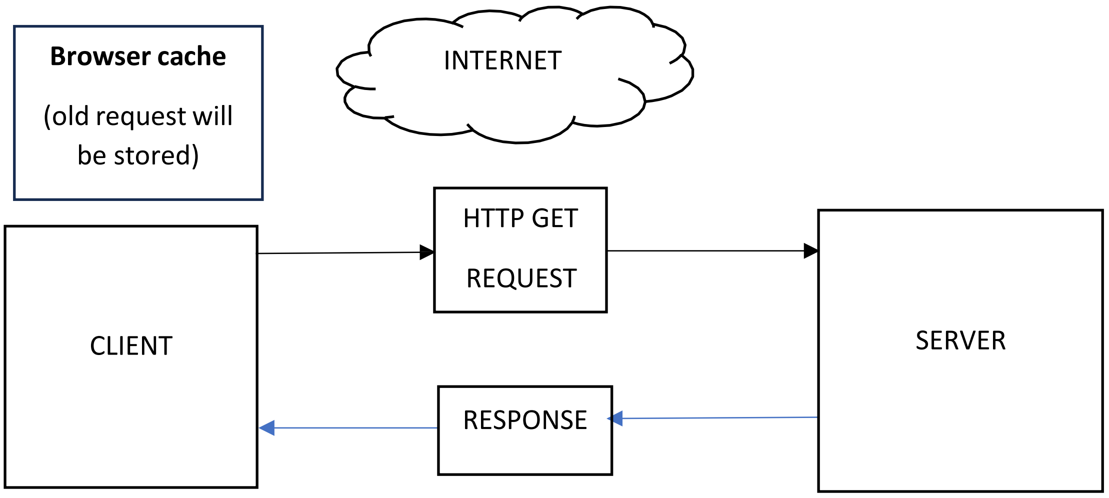
Multi Page Application (MPA)



- In MPA along with the old request new request will be sent to the server.
- Server unable to segregate which one is the old request and which one is the new request.
- So in MPA reloading time will be more.

Single Page Application (SPA)



- In SPA all the old request will be stored inside the browser cache and only a new request will be sent to the server.
- Browser cache is just a memory block.
- Before sending the request to the server, browser will check in the **browser cache** and compare the new request with the old request.
  If the requests are present inside the browser cache, then browser will not send the request to the server. Suppose if the requests are not present inside the browser cache then the request will be send to the server
- So in SPA reloading time will be very very less .

Disadvantage of SPA:

- Search Engine Optimization: Whenever you search some content one browser , ranking will be more for Multi Page Application than Single Page Application based on some of the keywords used in the application.

React JS :

- It is JavaScript library to create interactive user interfaces.
- Using react we can build Single Page Application.

History of ReactJS

- In 2011 facebook announced in the news feed that they will introduced one library to overcome the disadvantages of existing framework and library.
- In 2013 , Jorden Walkie employee of facebook developed ReactJS library.

What was the problem? Or Why facebook introduced React JS library?

- As facebook become bigger, its code base also got massive and when new data flows in, some small changes in the code base needed .
- Even if we do small changes in the code it will re-render the entire app.
- To overcome this React JS was introduced.
- And using ReactJS we can build Single Page Application.

(Important interview Question)

Features of ReactJS:

1)Declarative approach

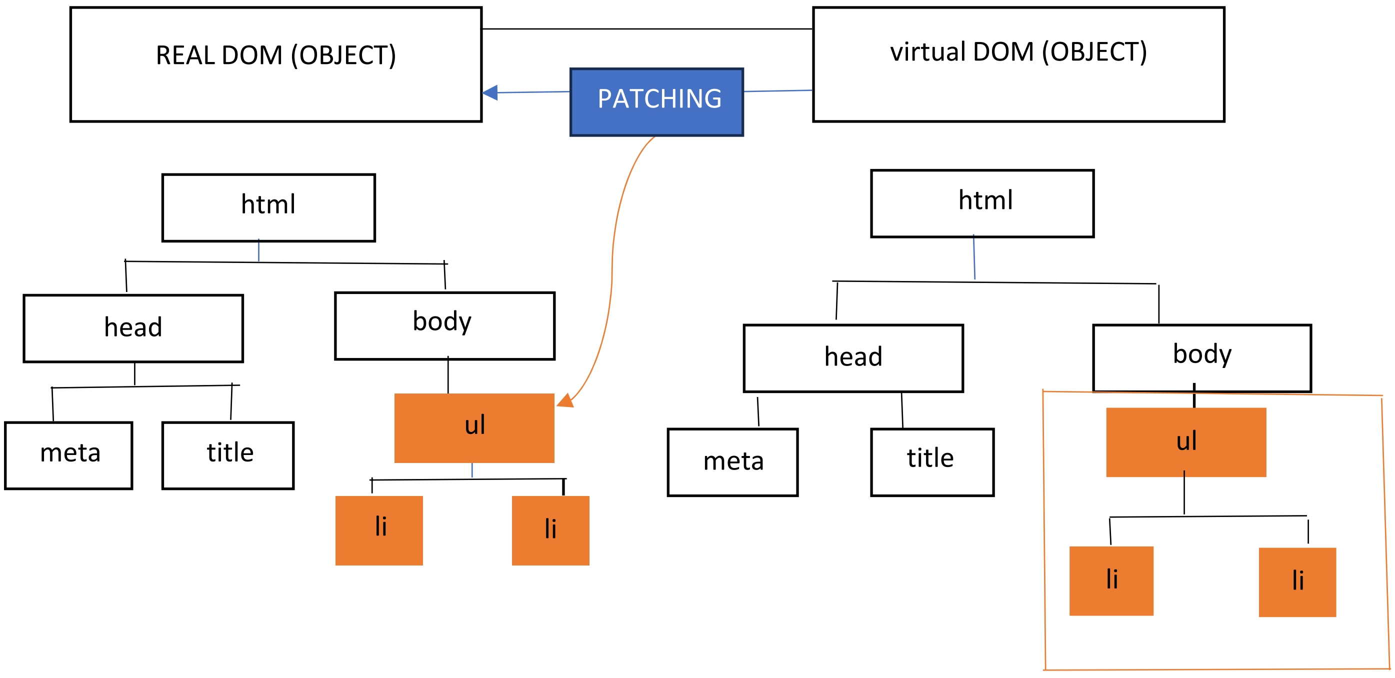
2)Component Based Architecture.

3)Single Page Application

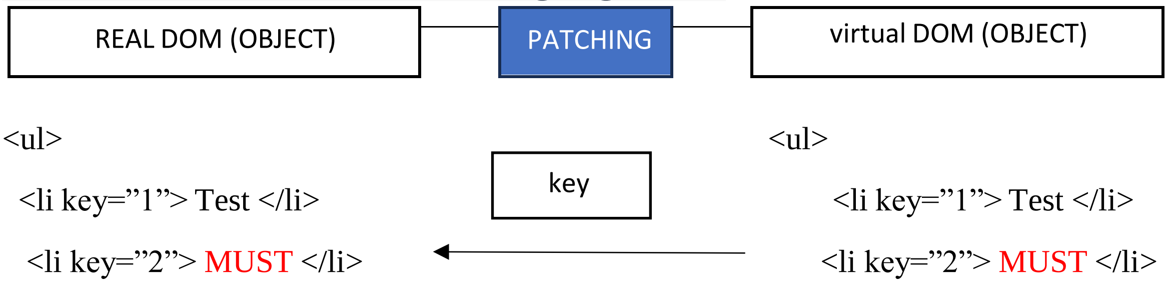
(Important interview Question)

Explain declarative approach / Explain React JS working / Explain virtual DOM / Reconciliation process.

Diffing Algorithm



Diffing Algorithm



```
<ul>                                      <ul>
  <li key="1"> Test </li>                   <li key="1"> Test </li>
   <li key="2"> MUST </li>                 <li key="2"> MUST </li>
    <li key="3"> Test </li>                   <li key="3"> Test </li>
</ul>                                       </ul>
```

- Here first-time structure of RealDOM is copied to a VirtualDOM.
- Whatever changes that we make first it will be updated in Virtual DOM, then Virtual DOM compares with the real DOM. If there is any change then it will be updated or patched in RealDOM using **Diffing Algorithm**.
- Process of updating changes in RealDOM is known as **patching**.
- This entire process is known as **reconciliation process**.

Why we need this key prop ?

- keys helps us to identify which items in the list is changed or added or updated
- key props plays a major role in handling **User Interface updates.**

COMPONENTS:

- React Component are reusable building block of code which associated with logic.
- **Types of Component :**
  1) Class based component
  2) Function based components

Note :

✓ Name of the component should start from uppercase and .jsx extension.

Ex : App.jsx

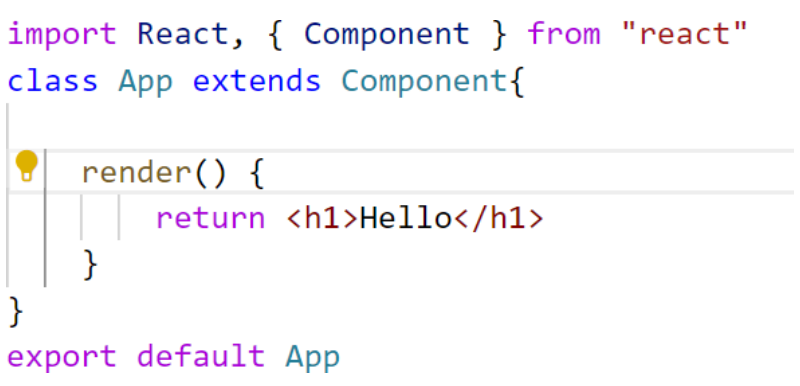
✓ We can represent component in 2 ways

i) < App/>

ii) <App><App/>

Class based component :

App.jsx



```
import React, { Component } from "react"
class App extends Component{

    render() {
        return <h1>Hello</h1>
    }
}
export default App
```

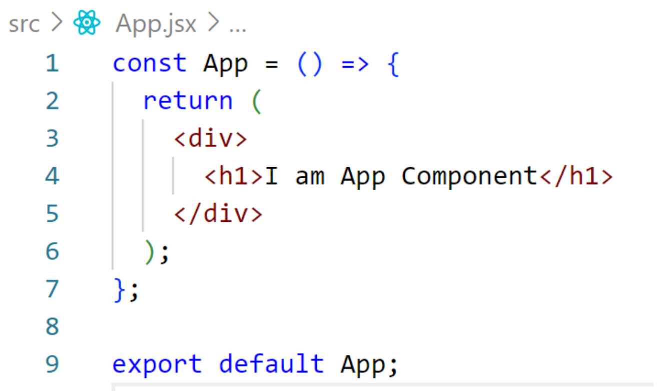
index.js

```
import React from "react"
import ReactDOM from "react-dom/client"
import App from "./App.jsx"

ReactDOM.createRoot(document.getElementById("root")).render(<App/>)
```

<u>Note :</u> In class based component there is one built in property called as state so class based component called as statefull Component.

<u>**Function based component :**</u>

<u>App.jsx</u>



```
const App = () => {
  return (
    <div>
      <h1>I am App Component</h1>
    </div>
  );
};

export default App;
```

<u>index.js</u>

```
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
createRoot(document.getElementById("root")).render(<App/>);
```

<u>Note :</u> In function based component there is no built in property called as state so function based component called as stateless Component.

Difference between Class based and Function based component (important interview question)

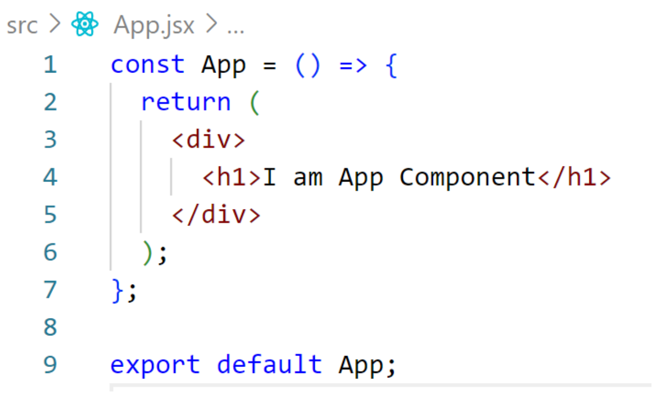
Class Based	**Function Based**
It is called statefull	**It is called stateless**
Hooks are not supported	**Hooks are supported**
Life Cycle methods are present	**Life Cycle methods are not present**

Render method is mandatory	**No render method**
this keyword points to current class based component	**no this keyword**

JSX: (Javascript XML) :

- It is the combination of JavaScript and XML.
- It is an extension to JS syntax allows you to write HTML-like code when working with React.

App.jsx



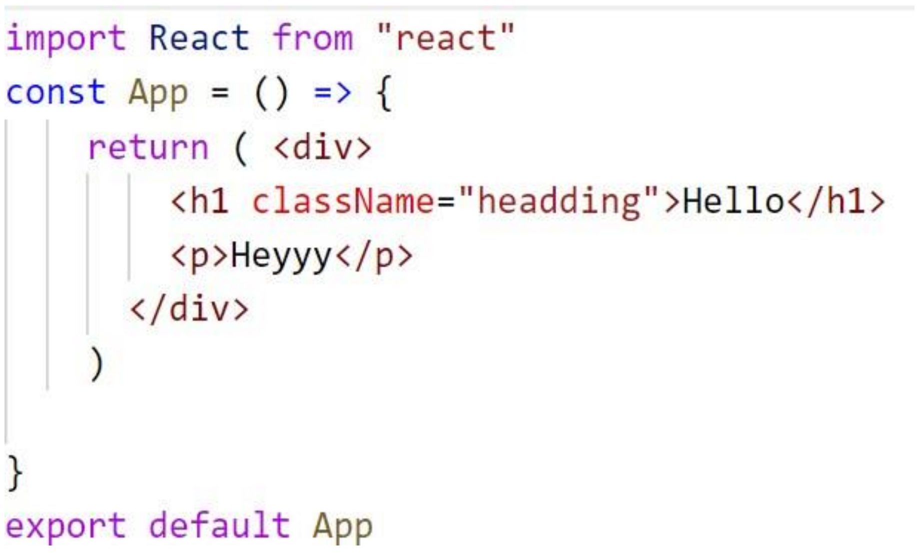
src > App.jsx > ...

```
const App = () => {
  return (
    <div>
      <h1>I am App Component</h1>
    </div>
  );
};

export default App;
```

In this example code inside return (<h1> I am App Component</h1>) statement looks similar to HTML, but it is actually JSX.

JSX RULES :

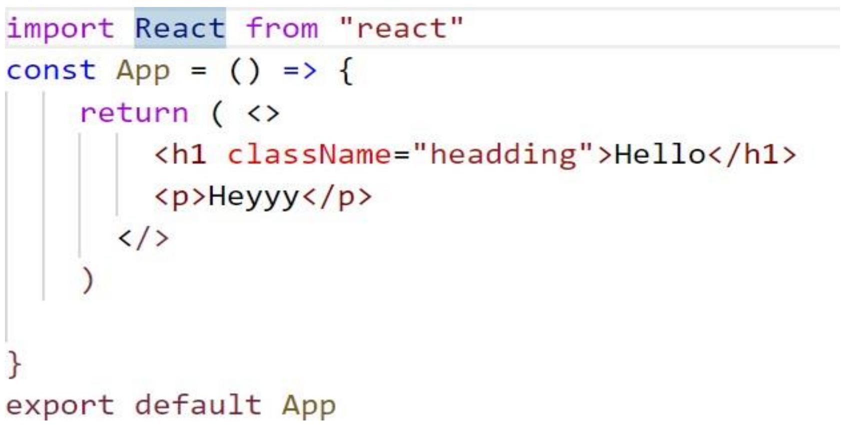
1) **When you want to render more than one element wrap them with one parent and ( )**



```
import React from "react"
const App = () => {
    return ( <div>
        <h1 className="headding">Hello</h1>
        <p>Heyyy</p>
      </div>
    )

}
export default App
```

Note : In the above example we have used <div> </div> as a parent .So in this case <div></div> will create one new node. In order to avoid this we have to use fragments as a parent.

<u>Wrapping with fragments</u>



```
import React from "react"
const App = () => {
    return ( <>
        <h1 className="headding">Hello</h1>
        <p>Heyyy</p>
      </>
    )

}
export default App
```

2) In JSX every element must be closed

- **In JSX all HTML tags must be properly closed.**

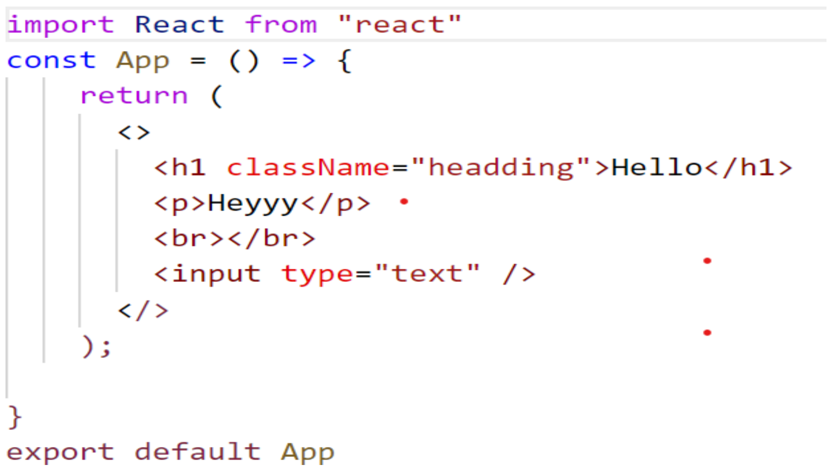
<u>Example:</u>

In HTML	In JSX

<input type = "text ">	`<input type="text" />`
	Or
	`<input type="text"></input>`

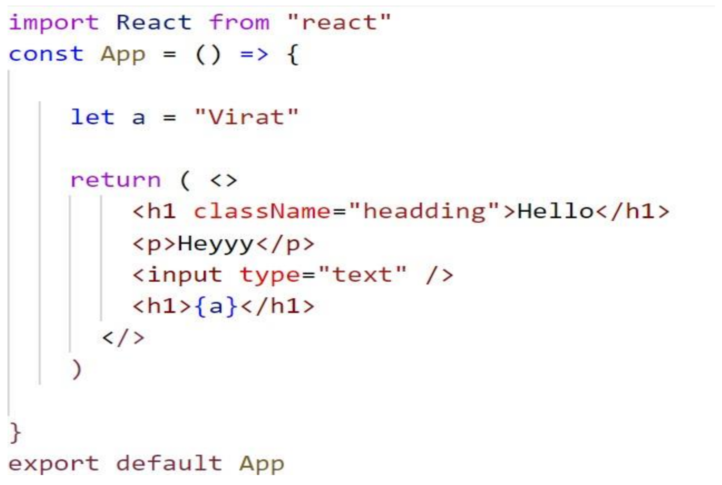
`<br>`	Or `<br />` ` </br>`

App.jsx



```jsx
import React from "react"
const App = () => {
    return (
      <>
        <h1 className="headding">Hello</h1>
        <p>Heyyy</p>
        <br></br>
        <input type="text" />
      </>
    );

}
export default App
```

3) In JSX in order access the variable we need to wrap them with expression { }



```jsx
import React from "react"
const App = () => {

    let a = "Virat"

    return ( <>
        <h1 className="headding">Hello</h1>
        <p>Heyyy</p>
        <input type="text" />
        <h1>{a}</h1>
      </>
    )

}
export default App
```

4) CamelCase Conventions

HTML	JSX
class	className
for	htmlFor

JS Events that we use here should also be in camelCase

onclick----------------------→onClick

State :

- ✓ State is used to store data at component level.
- ✓ In class based component there is a property called as state so class based component is called as Statefull component.
- ✓ In function based component there is no built in property called as state so it is called as StateLess Component.
- ✓ We can make Function based component as statefull by using a hook called as useState()

Hooks :

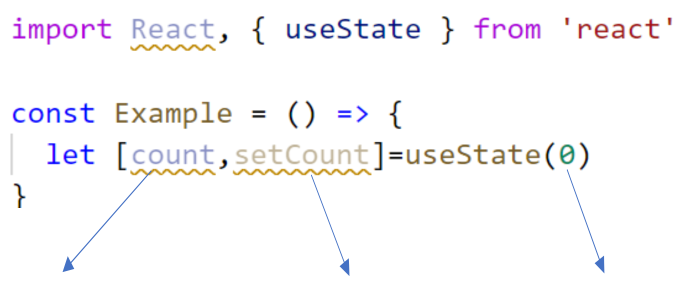
- ✓ Hook is nothing but a function and present only in function based component.
- ✓ Hooks are introduced React 16.8 version onwards.
- ✓ Why Hooks are used in ReactJS?
  - ➢ Hooks allows functional component to have access to state and other React features which present in Class based components.

Basic hooks :

1) useState ( )
2) useEffect ( )
3) useContext ()

useState( ) :

- This hook allows you to add state to FBC.
- It return an array [undefined,f]
- It accepts 2 values
  - a. current state
  - b. Updator function.
- Initialization of useState( )

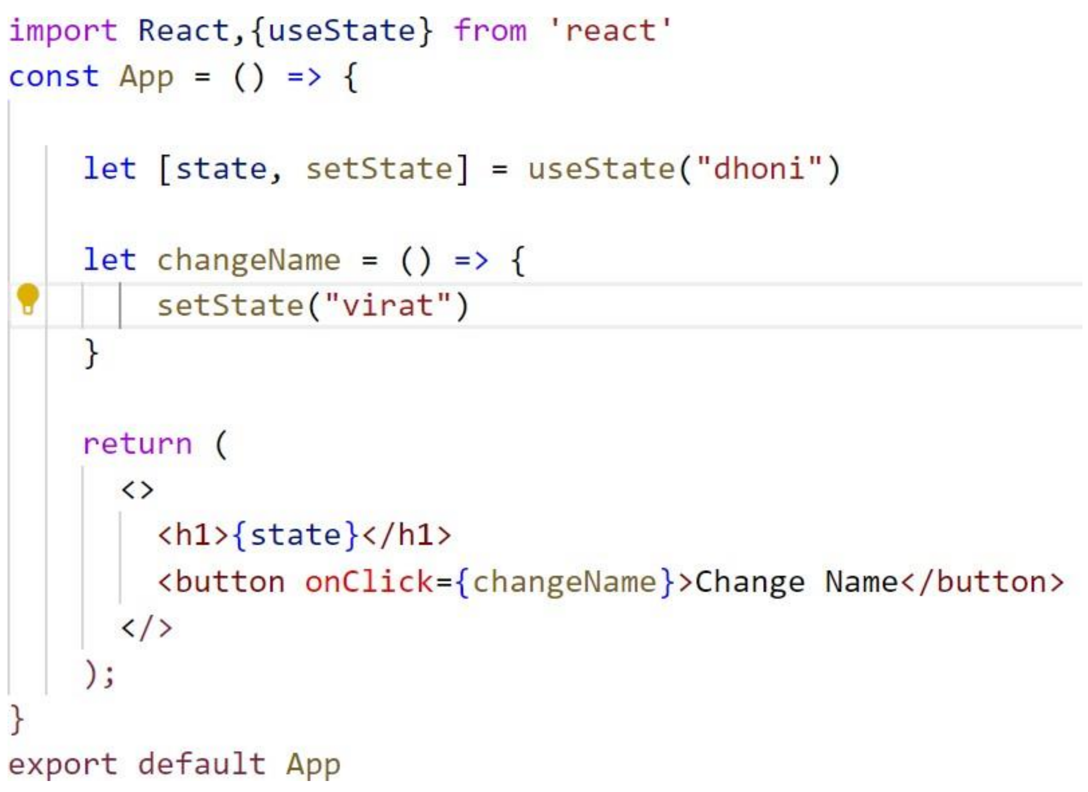


```
import React, { useState } from 'react'

const Example = () => {
  let [count,setCount]=useState(0)
}
```

State variable   setter function   initial value

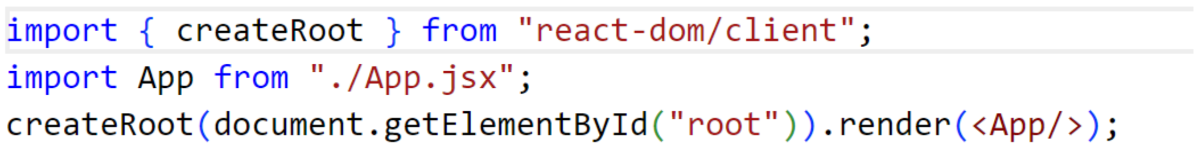
Note : While using useState( ) first we need to import that from React library.Example:

App.jsx



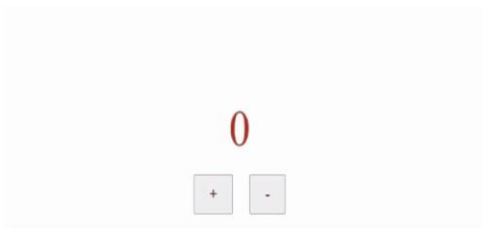
```
import React,{useState} from 'react'
const App = () => {

    let [state, setState] = useState("dhoni")

    let changeName = () => {
        setState("virat")
    }

    return (
        <>
            <h1>{state}</h1>
            <button onClick={changeName}>Change Name</button>
        </>
    );
}
export default App
```

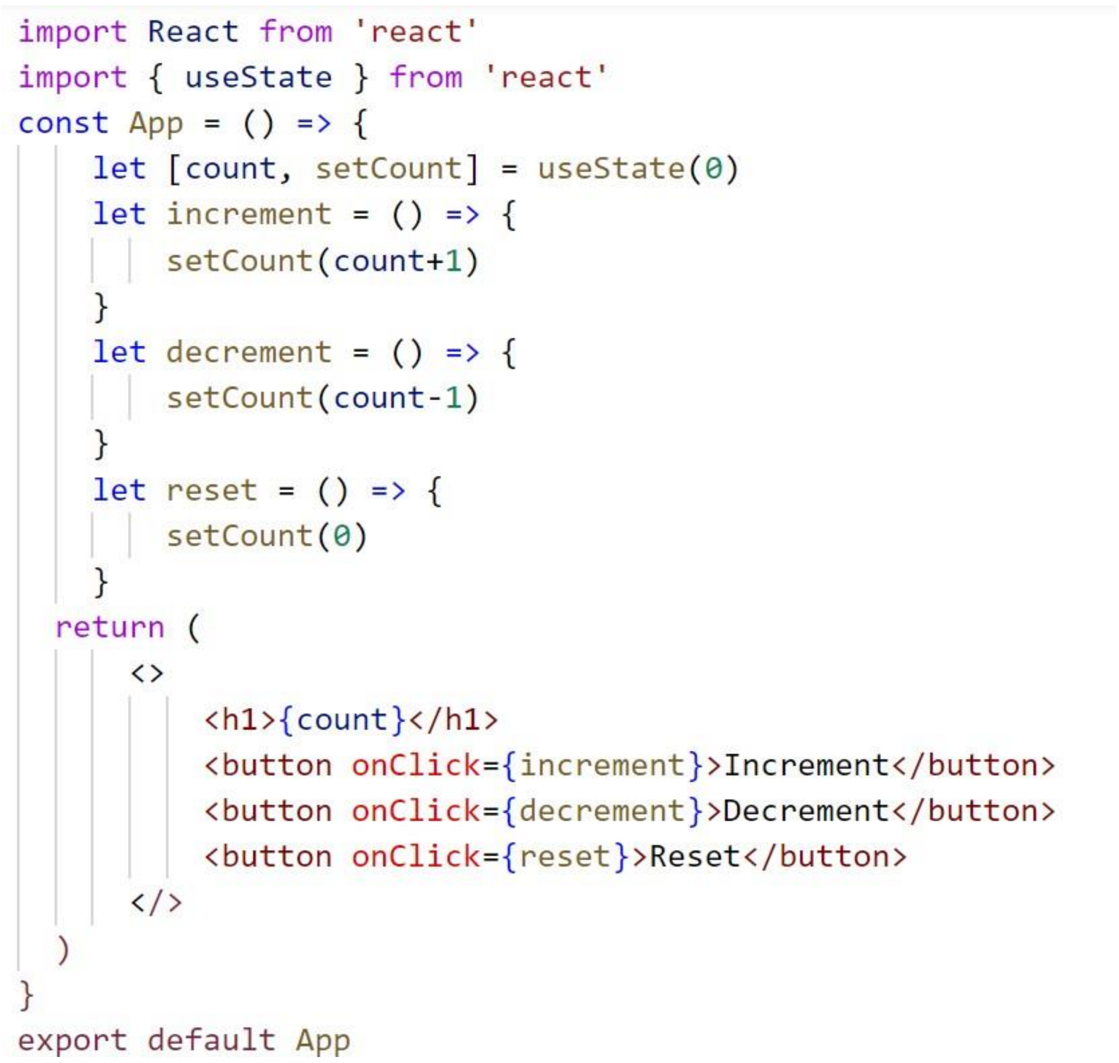
index.js



```
import { createRoot } from "react-dom/client";
import App from "./App.jsx";
createRoot(document.getElementById("root")).render(<App/>);
```

Example : Simple Conter





```
import React from 'react'
import { useState } from 'react'
const App = () => {
    let [count, setCount] = useState(0)
    let increment = () => {
        setCount(count+1)
    }
    let decrement = () => {
        setCount(count-1)
    }
    let reset = () => {
        setCount(0)
    }
  return (
    <>
        <h1>{count}</h1>
        <button onClick={increment}>Increment</button>
        <button onClick={decrement}>Decrement</button>
        <button onClick={reset}>Reset</button>
    </>
  )
}
export default App
```

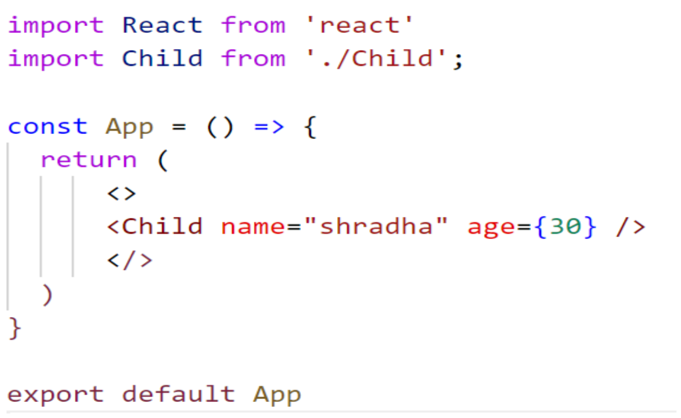
PROPS :

- ✓ Props are shorthand for properties.
- ✓ Props is an inbuilt object.
- ✓ Props is a way to communicate between the component.
- ✓ Props are used to send the data present in parent component into child component.
- ✓ Props are sent html like attribute format.

Example : Sending data present in App.jsx into Child.jsx. Here App.jsx acts as a parent and Child.jsx acts as child.

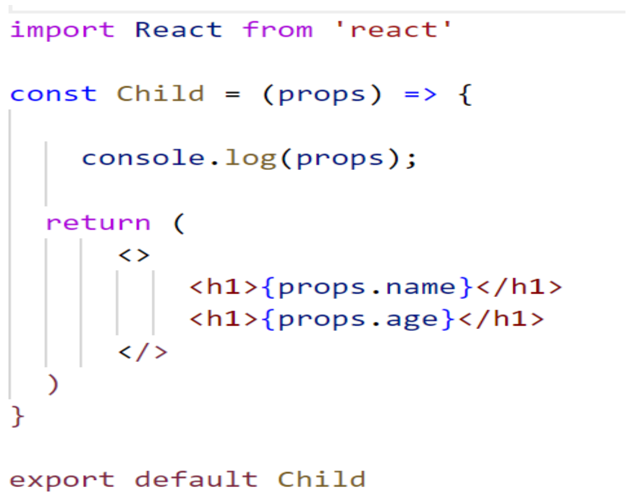
Ex:1

App.jsx :



```
import React from 'react'
import Child from './Child';

const App = () => {
  return (
      <>
      <Child name="shradha" age={30} />
      </>
  )
}

export default App
```

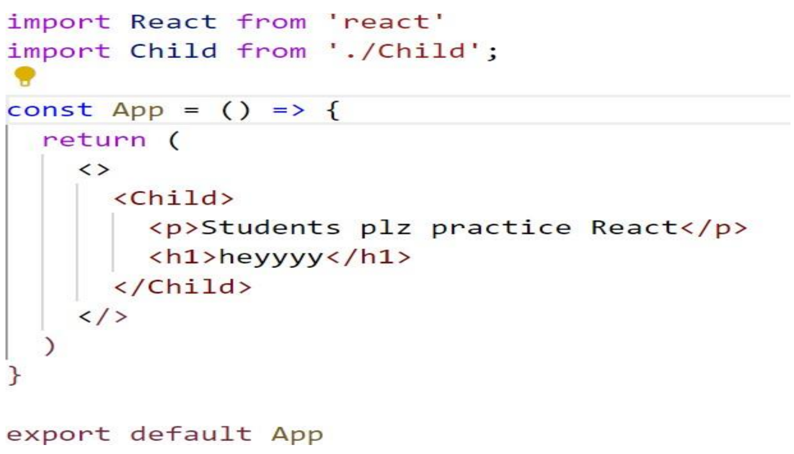
Child.jsx



```
import React from 'react'

const Child = (props) => {

    console.log(props);

    return (
        <>
            <h1>{props.name}</h1>
            <h1>{props.age}</h1>
        </>
    )
}

export default Child
```

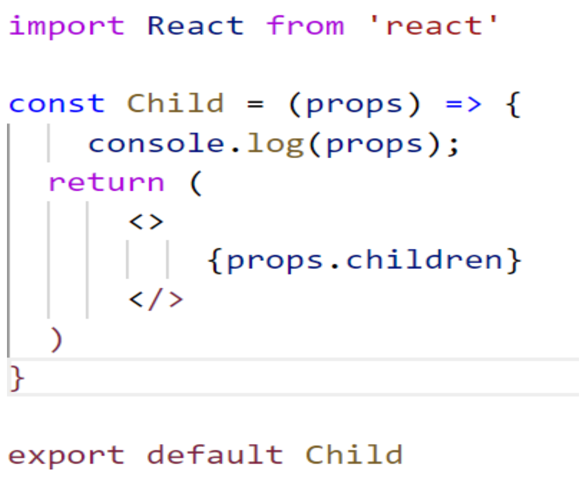
Props Children :

App.jsx :



```
import React from 'react'
import Child from './Child';

const App = () => {
  return (
    <>
      <Child>
        <p>Students plz practice React</p>
        <h1>heyyyy</h1>
      </Child>
    </>
  )
}

export default App
```

Child.jsx

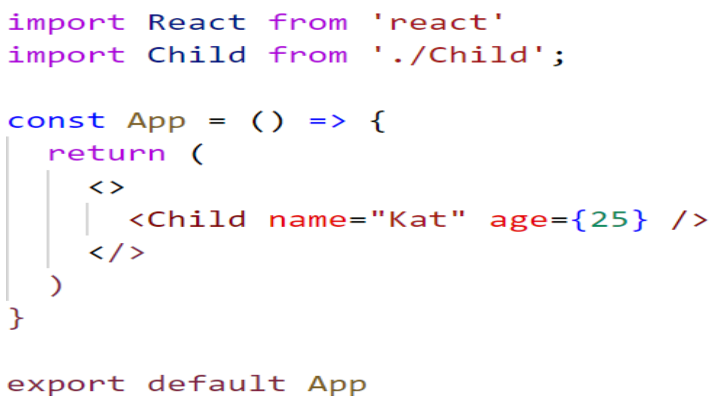


```
import React from 'react'

const Child = (props) => {
    console.log(props);
  return (
    <>
        {props.children}
    </>
  )
}

export default Child
```

Destructuring while receiving props

Ex:1

App.jsx



```
import React from 'react'
import Child from './Child';

const App = () => {
  return (
    <>
      <Child name="Kat" age={25} />
    </>
  )
}

export default App
```

Child.jsx

```
import React from 'react'
const Child = (props) => {

    let { name, age } = props

  return (
    <>
      <h1>{name}</h1>
      <h2>{age}</h2>
    </>
  )
}

export default Child
```

Destructuring while receiving props children

App.jsx

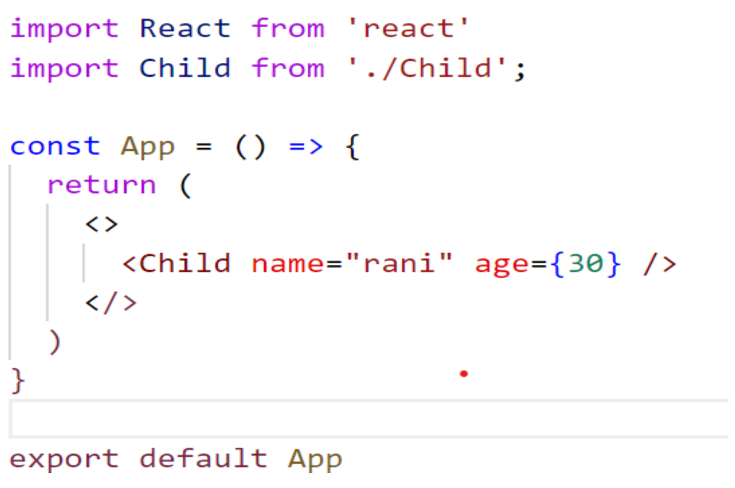
```
import React from 'react'
import Child from './Child';

const App = () => {
  return (
    <>
      <Child>
        <p>plz practice react </p>
      </Child>
    </>
  )
}

export default App
```

Child.jsx

```
import React from 'react'

const Child = (props) => {

    let {children} = props
  return (
    <>
      <h1>{children}</h1>
    </>
  )
}

export default Child
```

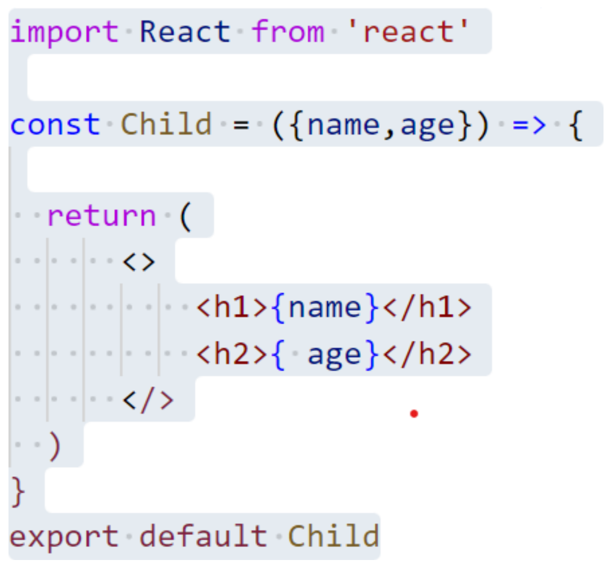
Direct destructuring while receiving props

App.jsx



```jsx
import React from 'react'
import Child from './Child';

const App = () => {
  return (
    <>
      <Child name="rani" age={30} />
    </>
  )
}

export default App
```

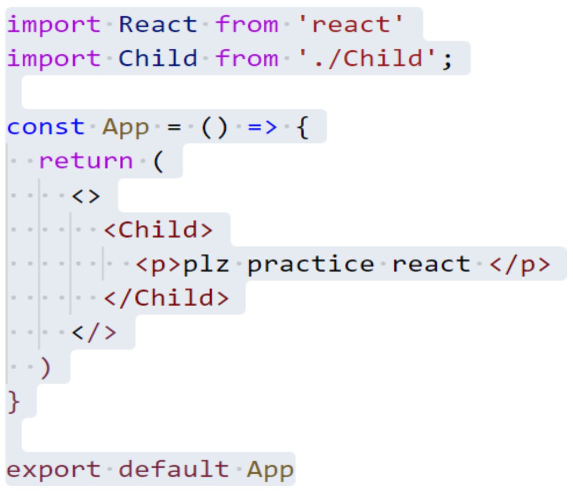
Child.jsx



```jsx
import React from 'react'

const Child = ({name,age}) => {

  return (
      <>
          <h1>{name}</h1>
          <h2>{ age}</h2>
      </>
  )
}
export default Child
```

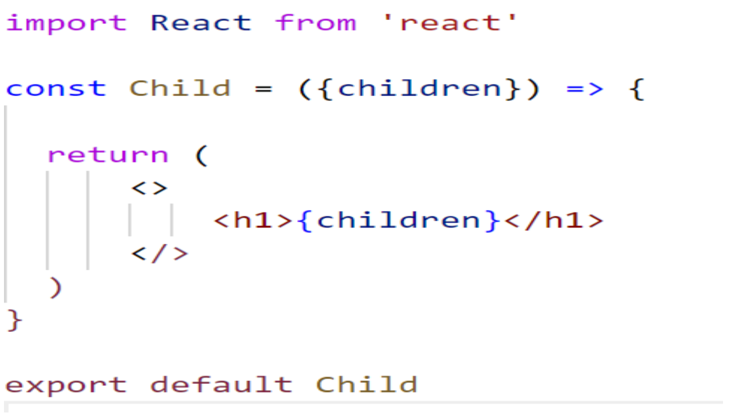
Ex : direct destructuring while receiving props children

App.jsx



```jsx
import React from 'react'
import Child from './Child';

const App = () => {
  return (
    <>
      <Child>
        <p>plz practice react </p>
      </Child>
    </>
  )
}

export default App
```

Child.jsx



```jsx
import React from 'react'

const Child = ({children}) => {

  return (
    <>
      <h1>{children}</h1>
    </>
  )
}

export default Child
```

(Important interview question)

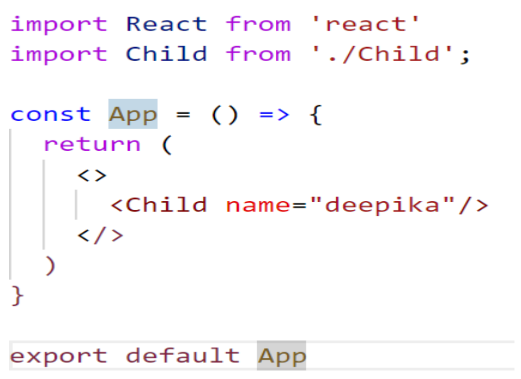
Difference Between state and props

States	Props
State is used to store data at component level	Props are used to communicate between the component, it is a data sent from one component to another component
State are mutable means state value can be changed	Props are immutable means props values cannot be changed

Props Drilling:

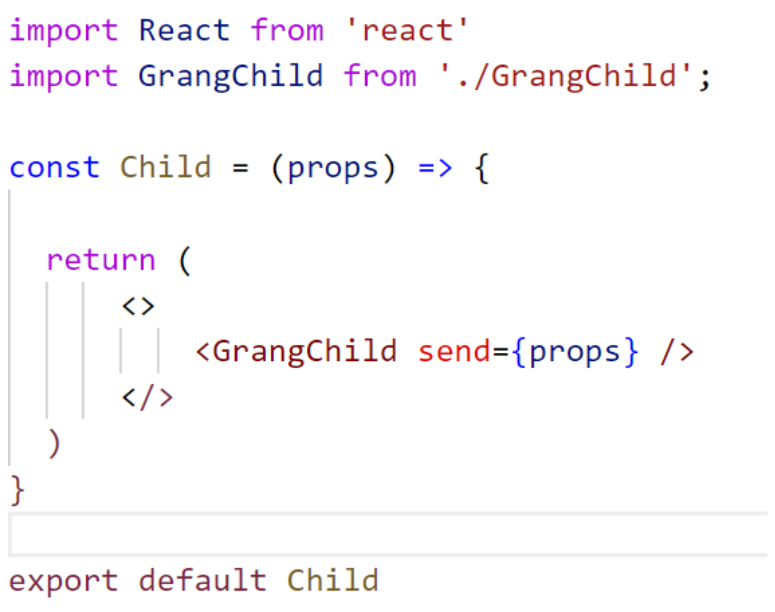
- Props drilling is a process of passing data from one component to another component through multiple layer of component ,even though some of those intermediate component don't directly use the data.

App.jsx



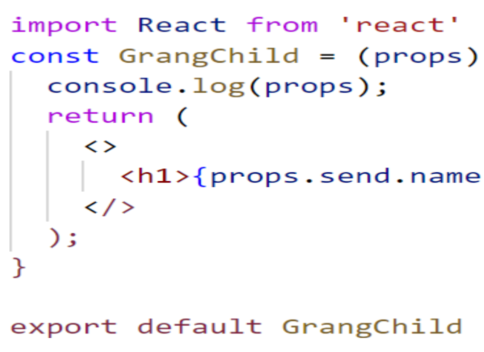
```jsx
import React from 'react'
import Child from './Child';

const App = () => {
  return (
    <>
      <Child name="deepika"/>
    </>
  )
}

export default App
```

Child.jsx



```jsx
import React from 'react'
import GrangChild from './GrangChild';

const Child = (props) => {

  return (
    <>
      <GrangChild send={props} />
    </>
  )
}

export default Child
```

GrandChild.jsx



```jsx
import React from 'react'
const GrangChild = (props) => {
  console.log(props);
  return (
    <>
      <h1>{props.send.name}</h1>
    </>
  );
}

export default GrangChild
```

In this example, we are sending props from App.jsx to GrandChild.jsx through Child.jsx. Where Child.jsx actually don't wanted the data.

NOTE : In order to resolve this props drilling problem we more often go with contextApi

ContextApi:

- ✓ React Hook provides a concept called context
- ✓ React ContextApi allows us to easily access the data at different level of the component tree, without passing prop to every level.
- ✓ We can achieve this by a hook called useContext( )

Parts present in ContextApi:

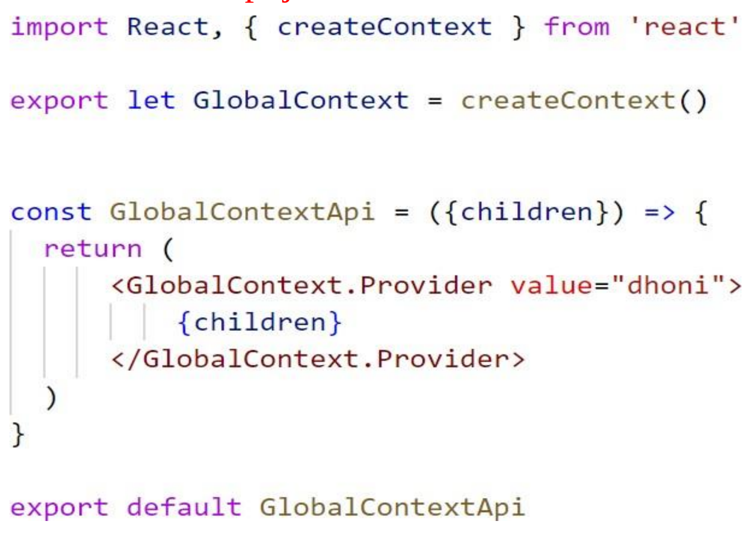
1) We need to create a context by createContext( )
2) Provider-→In order to provide the context we need provider.
3) Consumer→ in order to receive the context we need consumer.

Steps :

1) In src folder create one folder as context
2) Inside this context folder create one component file with name GlobalContextApi.js (here .js represent global)
3) Here if we want to send any props we have to use one special Attribute called as value.

   Note: In context api we have to use value only.

   GlobalContextApi.js

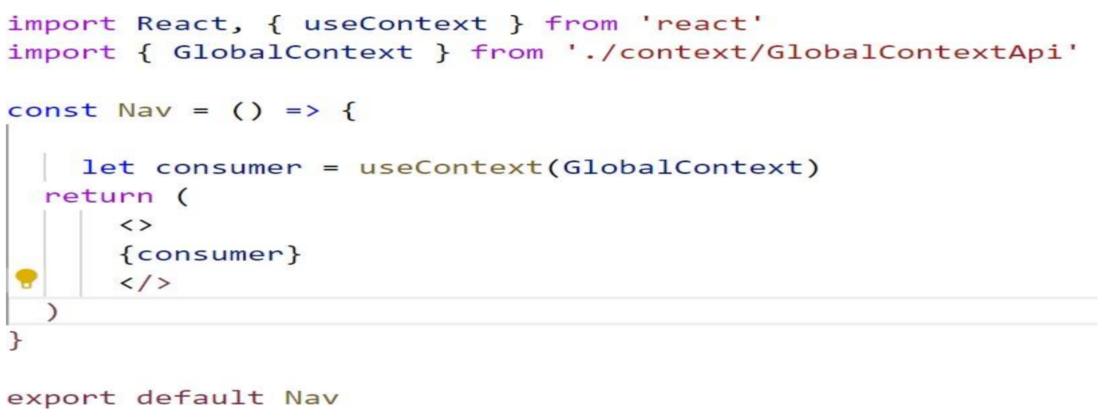


```
import React, { createContext } from 'react'

export let GlobalContext = createContext()


const GlobalContextApi = ({children}) => {
  return (
    <GlobalContext.Provider value="dhoni">
      {children}
    </GlobalContext.Provider>
  )
}

export default GlobalContextApi
```

- In this component we are creating context and exporting it in order to use it in other files.

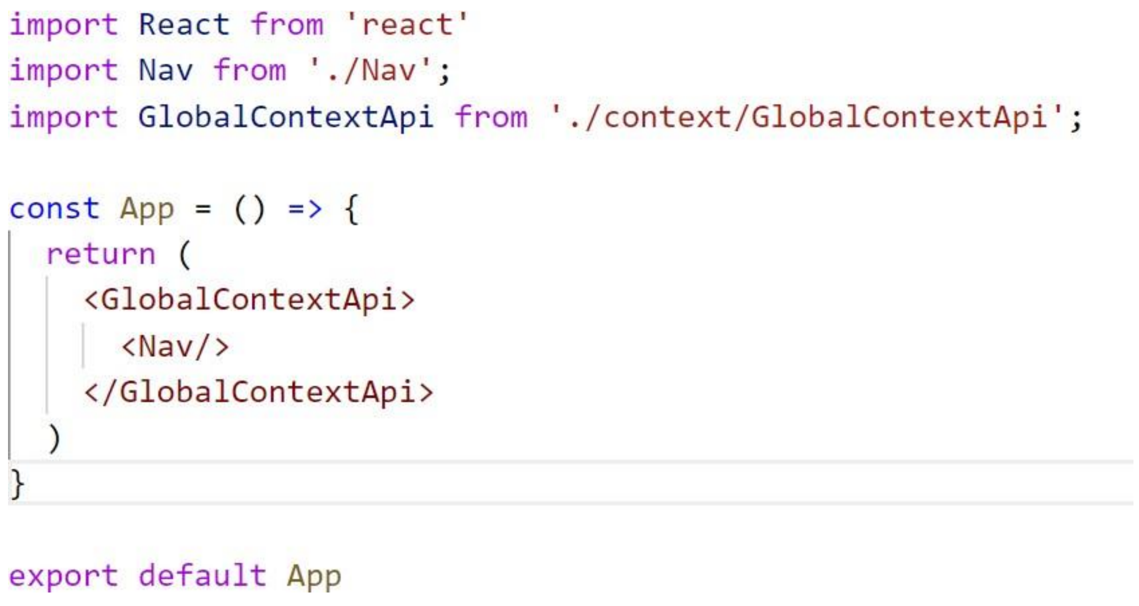
```
export let GlobalContext = createContext()
```

4) Create one more component in src folder let us say Nav.jsx where I want to receive the data. Which acts as a consumer.
Nav.jsx



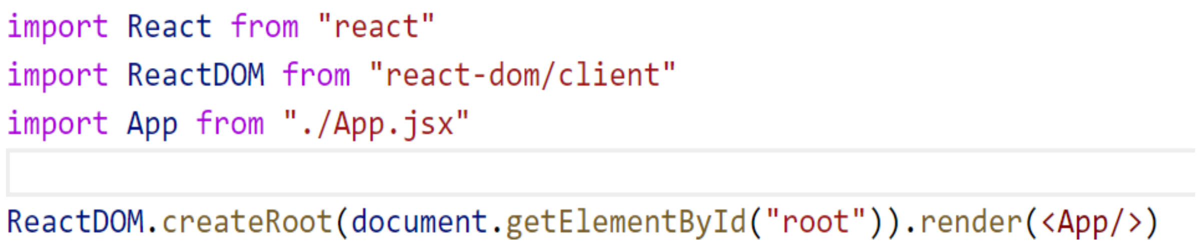
```
import React, { useContext } from 'react'
import { GlobalContext } from './context/GlobalContextApi'

const Nav = () => {

    let consumer = useContext(GlobalContext)
  return (
    <>
    {consumer}
    </>
  )
}

export default Nav
```

App.jsx



```
import React from 'react'
import Nav from './Nav';
import GlobalContextApi from './context/GlobalContextApi';

const App = () => {
  return (
    <GlobalContextApi>
      <Nav/>
    </GlobalContextApi>
  )
}

export default App
```

index.js



```
import React from "react"
import ReactDOM from "react-dom/client"
import App from "./App.jsx"

ReactDOM.createRoot(document.getElementById("root")).render(<App/>)
```

useReducer :

Reducer.jsx

```
import React from "react";
import { useReducer } from "react";
const Reducer = () => {
  let reducerFunc = (state, action) => {
    switch (action.type) {
      case "increment":
        return { count: state.count + 1 };

      case "decrement":
        return { count: state.count - 1 };

      case "reset":
        return { count: (state.count = 0) };
    }
  };
  let initialState = { count: 0 };

  let [state, dispatch] = useReducer(reducerFunc, initialState);

  return (
    <div>
      <h1>Counter = {state.count}</h1>
      <button onClick={() => dispatch({ type: "increment" })}>increment</button>
      <button onClick={() => dispatch({ type: "decrement" })}>increment</button>
      <button onClick={() => dispatch({ type: "reset" })}>increment</button>
    </div>
  );
};

export default Reducer;
```

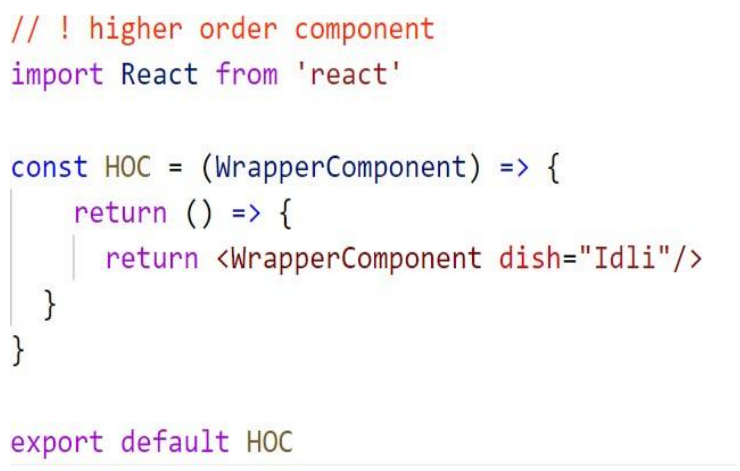
App.jsx

```
 const App = () => {
   return (
     <div>
       <Reducer/>
     </div>
   )
 }

 export default App
```

Higher Order Components :

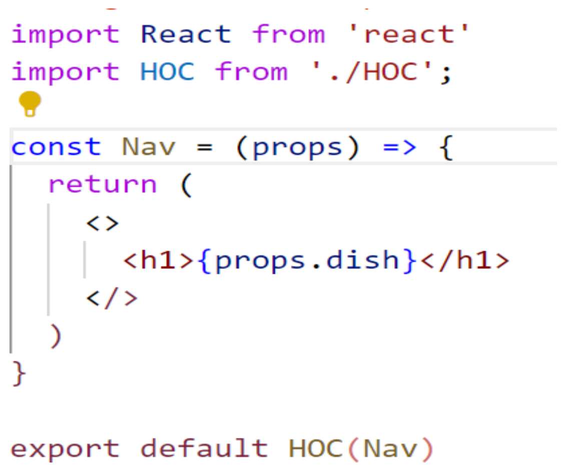
- Higher order component is also one of the solution for props-drilling. But we do not over use it.
- Higher Order Component is a Component which receives another component as an argument and returns a component.

HOC.jsx



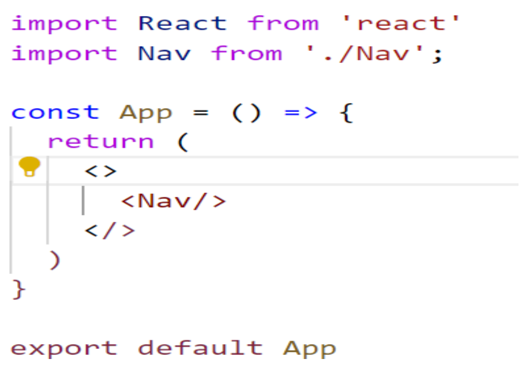
```
// ! higher order component
import React from 'react'

const HOC = (WrapperComponent) => {
    return () => {
      return <WrapperComponent dish="Idli"/>
  }
}

export default HOC
```

Nav.jsx



```
import React from 'react'
import HOC from './HOC';

const Nav = (props) => {
  return (
    <>
      <h1>{props.dish}</h1>
    </>
  )
}

export default HOC(Nav)
```

App.jsx



```
import React from 'react'
import Nav from './Nav';

const App = () => {
  return (
    <>
      <Nav/>
    </>
  )
}

export default App
```

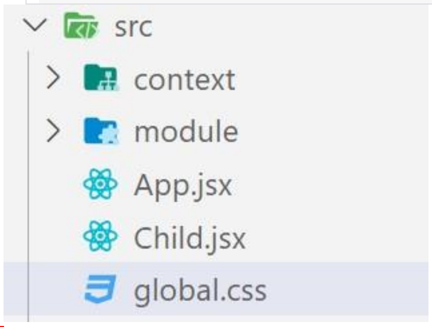
Ways of attaching CSS in React:

1)globalcss:

- Global CSS refers to styles that apply to the entire application and are not scoped to a specific component.
- These styles affect all elements throughout your application.

Steps:

- ✓ In src folder create one css style sheet with name global.css
- ✓ Import it in index.js file



```
src
  context
  module
  App.jsx
  Child.jsx
  global.css
```

App.jsx:

```
import React from 'react'
import Nav from './Nav';

const App = () => {
  return (
    <>
      <h1>App component</h1>
      <Nav/>
    </>
  )
}

export default App
```

global.css

```
h1{
    color: red
}
```

index.js

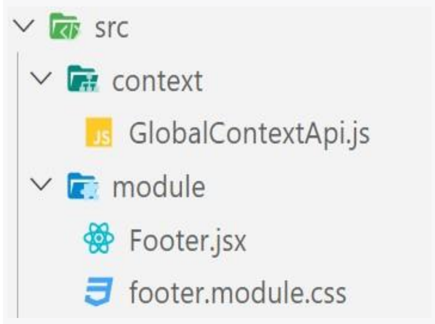
```
import React from "react"
import ReactDOM from "react-dom/client"
import App from "./App.jsx"

import './global.css'

ReactDOM.createRoot(document.getElementById("root")).render(<App/>)
```

2)Module.css:

- Module CSS (also known as CSS Modules) is a way to scope CSS to a specific component.
- It provides local scoping of styles, ensuring that styles only apply to the component they're imported into.

Steps :

- ✓ In src folder create another folder let us say component
- ✓ In component folder create one Component Footer.jsx
- ✓ In component folder create one css file footer.module.css
- ✓ Import footer.module.css file in Footer.jsx.



```
src
  context
    GlobalContextApi.js
  module
    Footer.jsx
    footer.module.css
```

Footer.jsx:

```
import React from 'react'
import Style from './footer.module.css'

const Footer = () => {
  return (
    <>
      <h1 className={Style.heading}>Footer component</h1>
    </>
  )
}

export default Footer
```

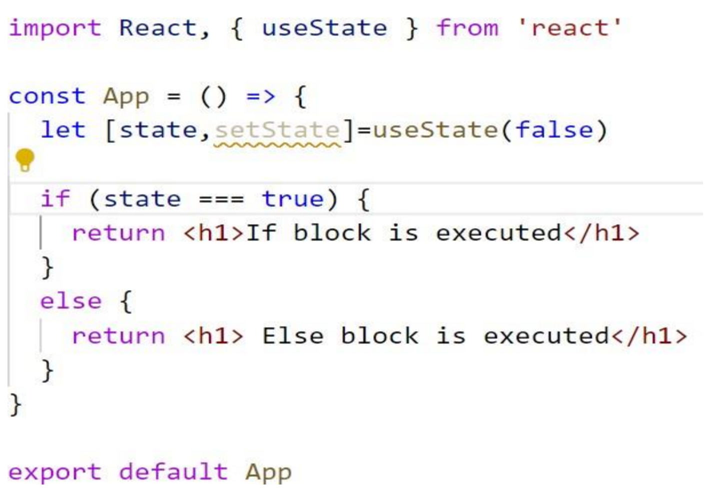
Footer.module.css

```
.heading{
    background-color: □pink;
}
```

Conditional Rendering::

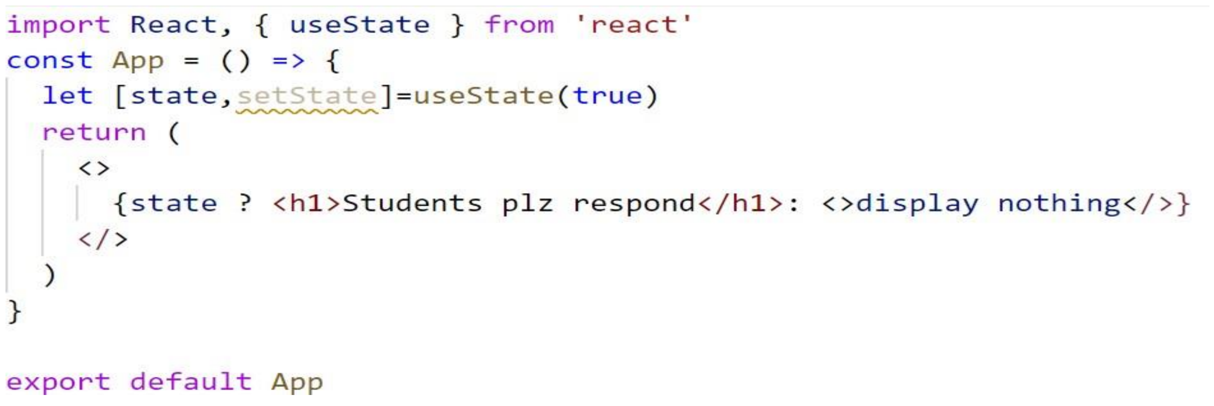
- Suppose if we want to display some content or don not want to display some content on UI based on some condition, then we are going with this conditional Rendering.

1) Using traditional if-else way



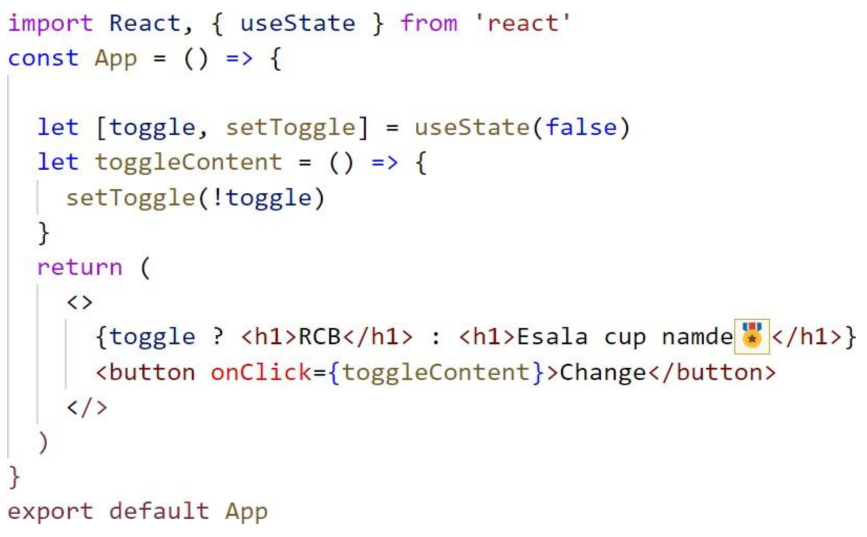
```
import React, { useState } from 'react'

const App = () => {
  let [state,setState]=useState(false)

  if (state === true) {
    return <h1>If block is executed</h1>
  }
  else {
    return <h1> Else block is executed</h1>
  }
}

export default App
```

2) Using Ternary operator



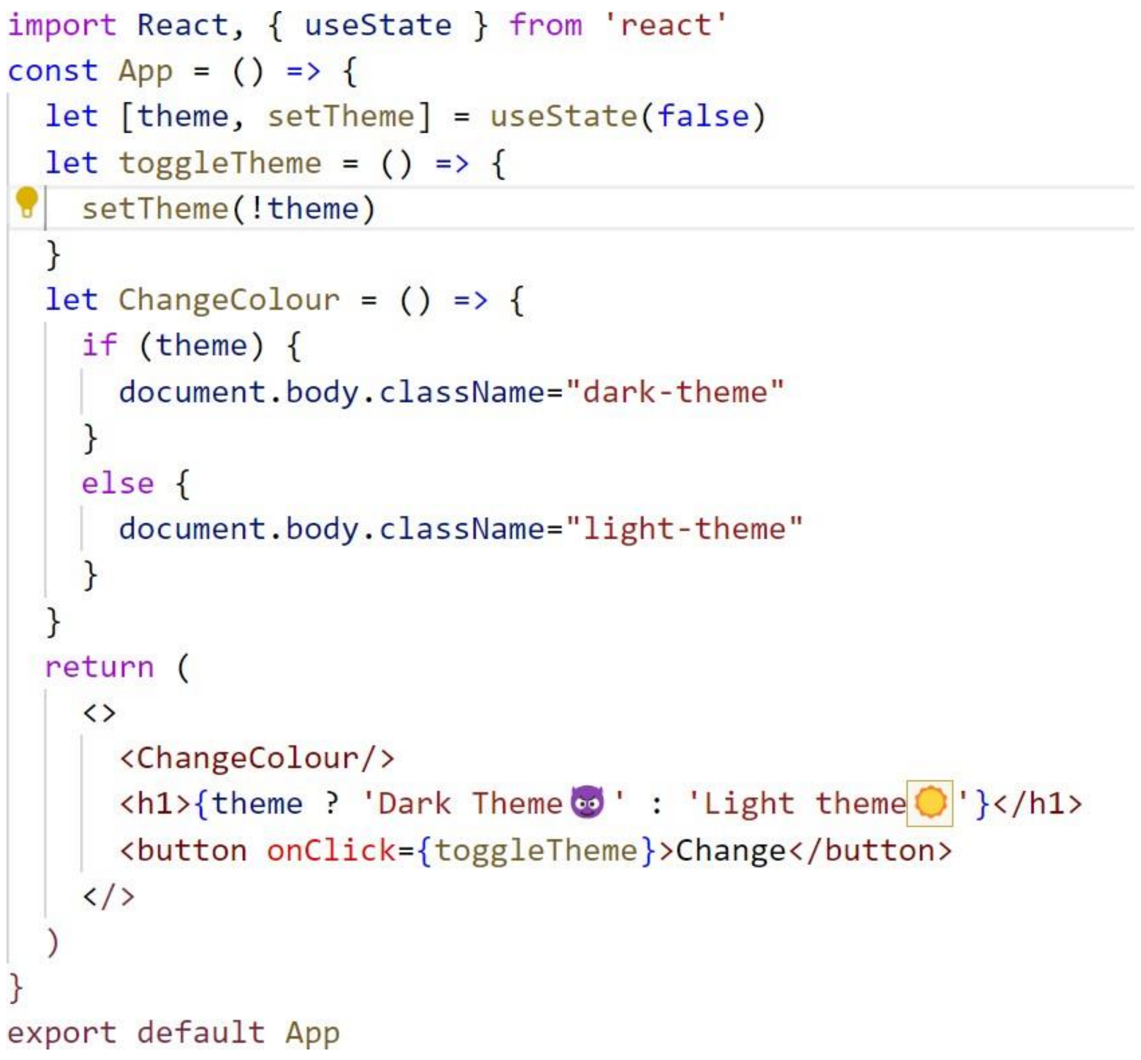
```
import React, { useState } from 'react'
const App = () => {
  let [state,setState]=useState(true)
  return (
    <>
      {state ? <h1>Students plz respond</h1>: <>display nothing</>}
    </>
  )
}

export default App
```

Task: Toggle the content



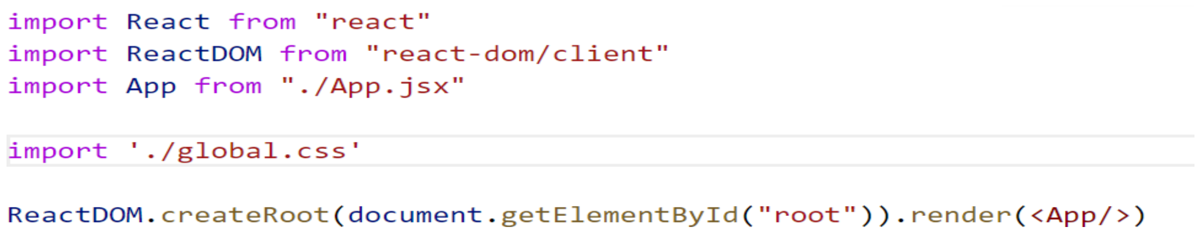
```
import React, { useState } from 'react'
const App = () => {

  let [toggle, setToggle] = useState(false)
  let toggleContent = () => {
    setToggle(!toggle)
  }
  return (
    <>
      {toggle ? <h1>RCB</h1> : <h1>Esala cup namde🏅</h1>}
      <button onClick={toggleContent}>Change</button>
    </>
  )
}
export default App
```

Task:Dark and Light Theme



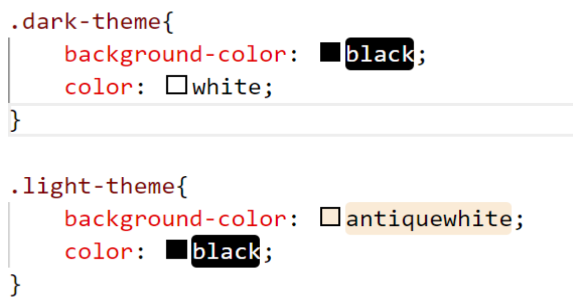
```
import React, { useState } from 'react'
const App = () => {
  let [theme, setTheme] = useState(false)
  let toggleTheme = () => {
    setTheme(!theme)
  }
  let ChangeColour = () => {
    if (theme) {
      document.body.className="dark-theme"
    }
    else {
      document.body.className="light-theme"
    }
  }
  return (
    <>
      <ChangeColour/>
      <h1>{theme ? 'Dark Theme😈' : 'Light theme🌞'}</h1>
      <button onClick={toggleTheme}>Change</button>
    </>
  )
}
export default App
```

index.js



```
import React from "react"
import ReactDOM from "react-dom/client"
import App from "./App.jsx"

import './global.css'

ReactDOM.createRoot(document.getElementById("root")).render(<App/>)
```

global.css



```
.dark-theme{
    background-color: ■black;
    color: □white;
}

.light-theme{
    background-color: □antiquewhite;
    color: ■black;
}
```

Refs:

- ✓ Refs are shorthand for reference.
- ✓ Refs are used to create the reference of a particular element.
- ✓ It is imperative way means we are directly communicating with DOM. So we should not over use refs.
- ✓ Inside ref there is one property called as current.

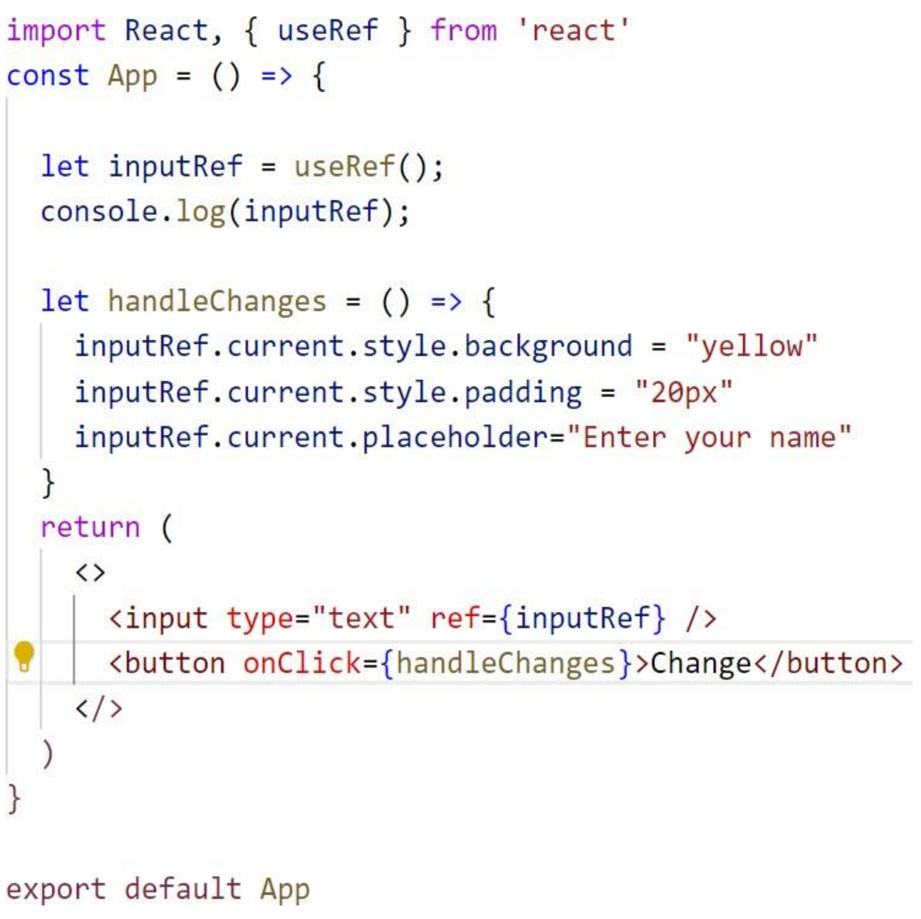
Steps :

1)first create a ref using useRef( ) hook.

2) In order to access the particular element, pass one special attribute called as ref insideparticular element.

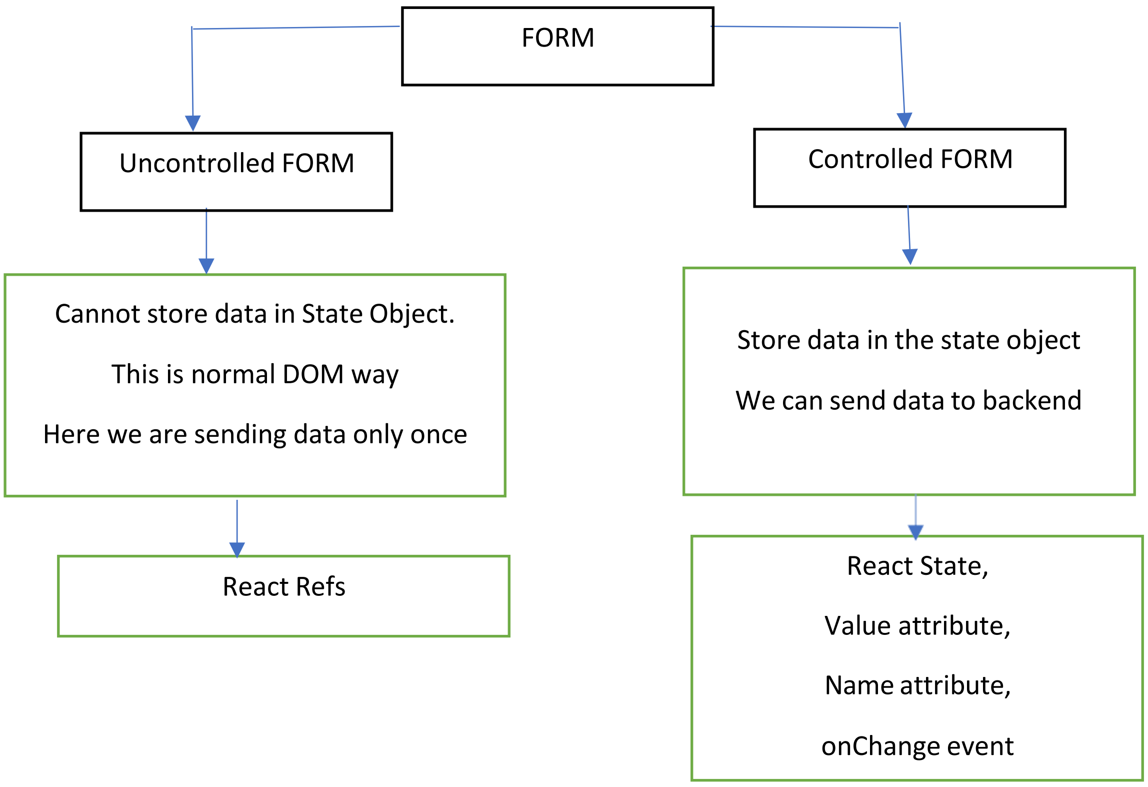
3) Inside the ref pass the created refs, ref={ }. So that we are able to access all the properties of the targeted element.

App.jsx



```
import React, { useRef } from 'react'
const App = () => {

  let inputRef = useRef();
  console.log(inputRef);

  let handleChanges = () => {
    inputRef.current.style.background = "yellow"
    inputRef.current.style.padding = "20px"
    inputRef.current.placeholder="Enter your name"
  }
  return (
    <>
      <input type="text" ref={inputRef} />
      <button onClick={handleChanges}>Change</button>
    </>
  )
}

export default App
```

Forms in React:


FORM
Uncontrolled FORM
Controlled FORM
Cannot store data in State Object.
This is normal DOM way
Here we are sending data only once
Store data in the state object
We can send data to backend
React Refs
React State,
Value attribute,
Name attribute,
onChange event

Controlled Forms

- A controlled form in React is like a form where React controls all the input elements.
- This means that the values of the form elements are stored in the component's state and are updated through React.
- When a user interacts with the form (like typing in an input field), React updates the state, and the value of the input field reflects what's in the component's state.

```
import React, { useState } from 'react'

const App = () => {
  let [state, setState] = useState({
    name: "",
    password: "",
    gender:""
  })

  let [skills, setSkills] = useState([])

  let { name, password,gender } = state;
let handleSubmit = (e) => {
    e.preventDefault();
    console.log(state);
    console.log(skills);
  }

  let handleChange = (e) => {
    console.log(e);
    let { name, value } = e.target;
    setState({...state,[name]:value})
  }

  let handleSkills = e => {
    setSkills([...skills,e.target.value])
  }
  return (
    <>
      <form onSubmit={handleSubmit}>
        <div>
          <input
            type="text"
            placeholder="enter your name"
            value={name}
            name="name"
            onChange={handleChange}
          />
        </div>
        <div>
          <input
            type="password"
            placeholder="enter your password"
            value={password}
            name="password"
            onChange={handleChange}
          />
        </div>
        <div value={gender} name="gender" onChange={handleChange}>
          <label htmlFor="">Gender:</label>
          <input type="radio" name="gender" value="male" />
          Male
          <input type="radio" name="gender" value="female" />
          Female
        </div>
        <div value={skills} name='skills' onChange={handleSkills}>
          <label htmlFor="">Skills</label>
          <input type="checkbox" name='skill' value='html'/>
          Html
          <input type="checkbox" name='skill' value='css'/>
          CSS
          <input type="checkbox" name='skill' value='js'/>
          JS
        </div>
        <div>
          <button>Submit</button>
        </div>
      </form>
    </>
  );
}

export default App
```

Uncontrolled Forms

- An uncontrolled form in React is like a form where React doesn't directly manage the values of the input elements.
- Instead, the values of the form elements are managed by the DOM (the web page itself).

App.jsx

```
import React, { useRef } from 'react'

const App = () => {
  let inputRef = useRef()
  let passwordRef = useRef()

  let handleSubmit = (e) => {
    console.log(e);
    e.preventDefault();
    let name = inputRef.current.value;
    let pass = passwordRef.current.value;
    console.log(name,pass);
  }
  return (
    <>
      <form onSubmit={handleSubmit}>
        <div>
          <input type="text" ref={inputRef}/>
        </div>
        <div>
          <input type="password" ref={passwordRef}/>
        </div>
        <div>
          <button>Submit</button>
        </div>
      </form>
    </>
  )
}

export default App
```

Life cycle methods (CBC)

- ✓ Life cycle methods in class-based components are special methods that are automatically invoked at various points in the lifecycle of a React component. They provide opportunities to perform actions at specific stages, such as when a component is mounted, updated, or unmounted.

- ✓ In simple words it is cycle between the component's birth to death.

In life cycle methods there are several phases

1. **Mounting**
2. **updation**
3. **unmounting(death)**

- ➢ **Each and every phases of life cycle has its own methods.**

1) Mounting Phase (birth)

Whenever an instance of an component is created and we want to insert that in to the dom tree, then this mounting phase is executed

Method of mounting phase :

- **constructor---**it will execute only once whenever the instance of a component is created.
- **render**---it will execute whenever we want to render something on UI, or any modification or updation happens.
- **componentDidMount**--it will similar to constructor it executes only once, after creating instance and rendering it on UI in order to insert into dom tree this is exected.

2)Updation---

Suppose if I want to make any changes or updation in the instance of a component, like state is there and we want to make changes in the state using setState and sending data through props.

Method of updation phase :

- **render**--whenever we do any modification in a components like state and that changes should display on UI
- **componentDidUpdate**--after doing updation that particular part should get updated to dom tree.It executes only once.

3)Unmounting(death)—

whenever we want to remove an instance of an component from the dom tree then this unmounting phase will execute.

Method of unmounting phase :

componentWillUnmount--suppose if we want to remove the particular component.

NOTE: In FBC components we can achieve these life cycle methods using a hook called as useEffect

useEffect

- ✓ useEffect are used to perform side effects operations in functional components.
- ✓ It acts as a alternate for comopnentDidMount( ), componentDidUpdate( ), componentWillUnmount( )
- ✓ useEffect accepts 2 arguments.

Syntax

useEffect(< functions >, < dependency >)

It can be an asynchronous function.

Performs asynchronous operations like data fetching inside it.

This array of dependency tells when to re-run the effect.

It is optional.

App.jsx

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

const App = () => {

    let [dish,setDish]=useState("dosa")

    useEffect(() => {
        console.log("render method");
    })

    //!empty dependency--->componentDidMount
    useEffect(() => {
        console.log("Component mounting");
    }, [])

    //!componentDidUpdate
    useEffect(() => {
        console.log("update");
    },[dish])

    let changeDosa = () => {
        setDish("Dahibara")
    }
  return (
      <>
        <h1>{dish}</h1>
        <button onClick={changeDosa}>Change the dish</button>
      </>
  )
}

export default App
```

Ex: unmounting

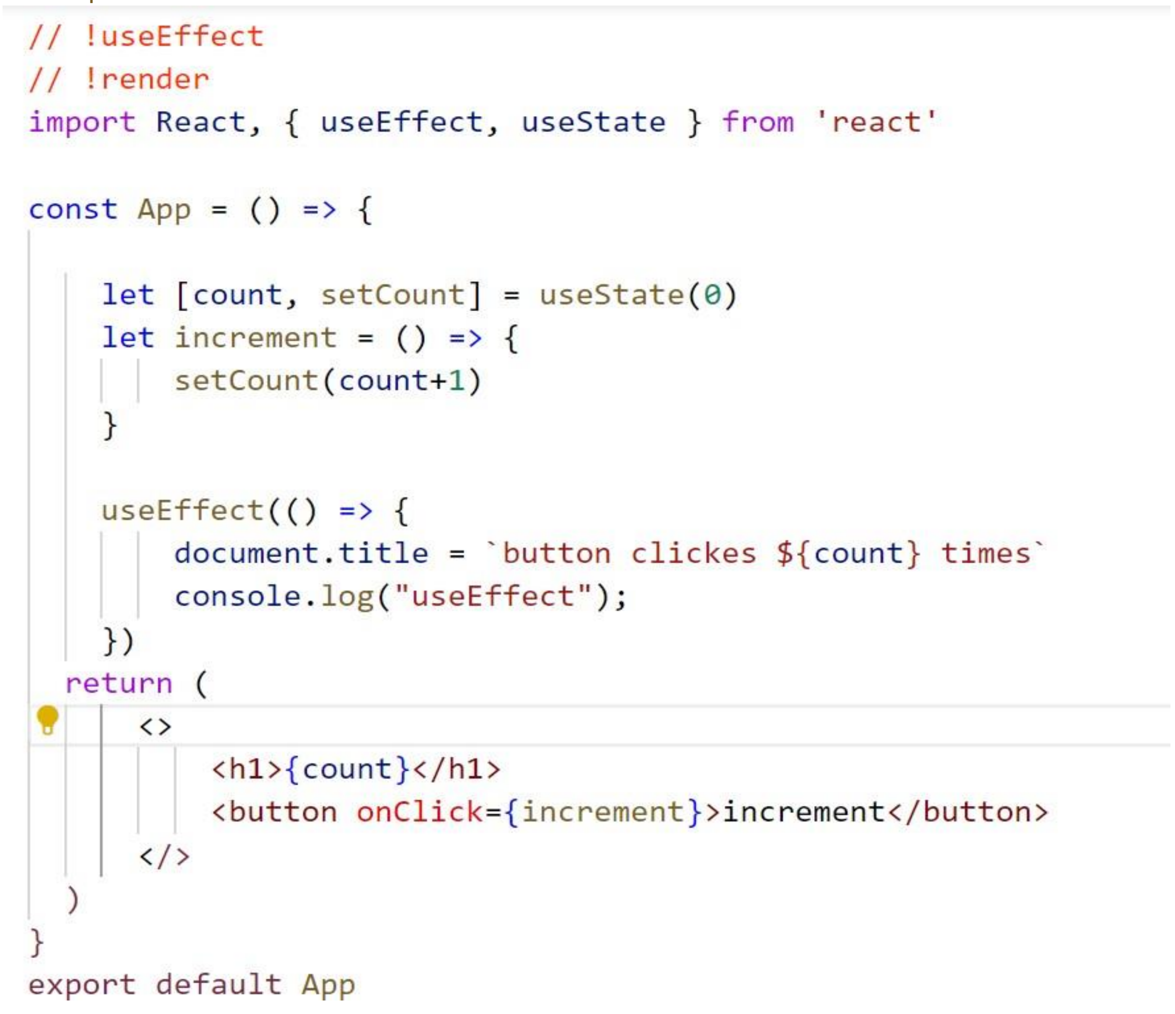
App.jsx

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

const App = () => {

    let [dish,setDish]=useState("dosa")

    useEffect(() => {
        console.log("render method");
    })

    //!empty dependency--->componentDidMount
    useEffect(() => {
        console.log("Component mounting");
    }, [])

    //!componentDidUpdate
    useEffect(() => {
        console.log("update");
    },[dish])

    let changeDosa = () => {
        setDish("Dahibara")
    }
  return (
      <>
          <h1>{dish==="dosa"?<Nav/>:<></>}</h1>
          <button onClick={changeDosa}>Change the dish</button>
      </>
  )
}

export default App
```

Nav.jsx

```
import React, { useEffect } from 'react'

const Nav = () => {
    //!componentwillumount--->cleanup function

    useEffect(() => {
        return () => {
            console.log("umounting");
        }
    })

  return (
    <div>Unmounting Example</div>
  )
}

export default Nav
```

Ex: equivalent to render



```
// !useEffect
// !render
import React, { useEffect, useState } from 'react'

const App = () => {

    let [count, setCount] = useState(0)
    let increment = () => {
        setCount(count+1)
    }

    useEffect(() => {
        document.title = `button clickes ${count} times`
        console.log("useEffect");
    })
  return (
      <>
        <h1>{count}</h1>
        <button onClick={increment}>increment</button>
      </>
  )
}
export default App
```

JavaScript Concept

Promises:

- It is an object.
- It will always see the event you are performing whether it is completed or not.
- It will keep track on what we are doing

3 Stages:

1) **Pending state :**
- ✓ This is the initial state, neither fulfilled nor rejected. This means that the asynchronous operation hasn't completed yet.

2) **Resolved state :**
- ✓ This means that the operation completed successfully and the Promise now has a value. This value is often referred to as the fulfillment value.

3) **Rejected state:**
- ✓ This means that an error occurred during the operation, and the Promise is rejected with a reason (usually an Error object) explaining what went wrong.

Note :

- Whenever we have promise we need to resolve it.
- We can resolve it by either using async and await or then and catch

How to create a promise ?

1) **Create a Promise constructor**

```
let pobj = new Promise( )
```

2) **Whenever we create a promise using new keyword we need to execute a function**

```
let pobj = new Promise( ( ) =>{
})
```

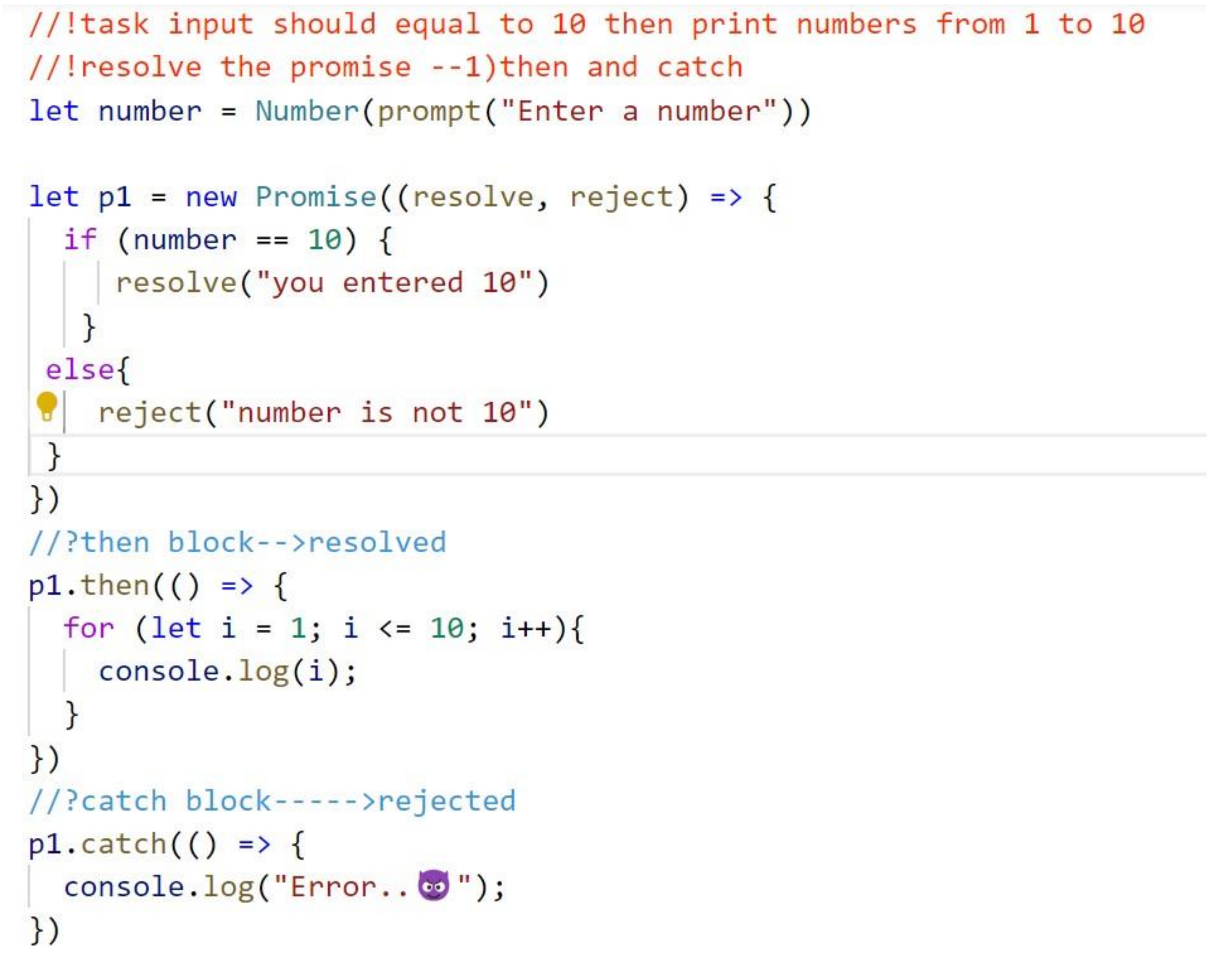
3) **In this execute function we have to pass 2 arguments as function.**
   let pobj = new Promise((resolve, reject) =>{

   })

   Final syntax:
   let pobj = new Promise((resolve, reject) =>{

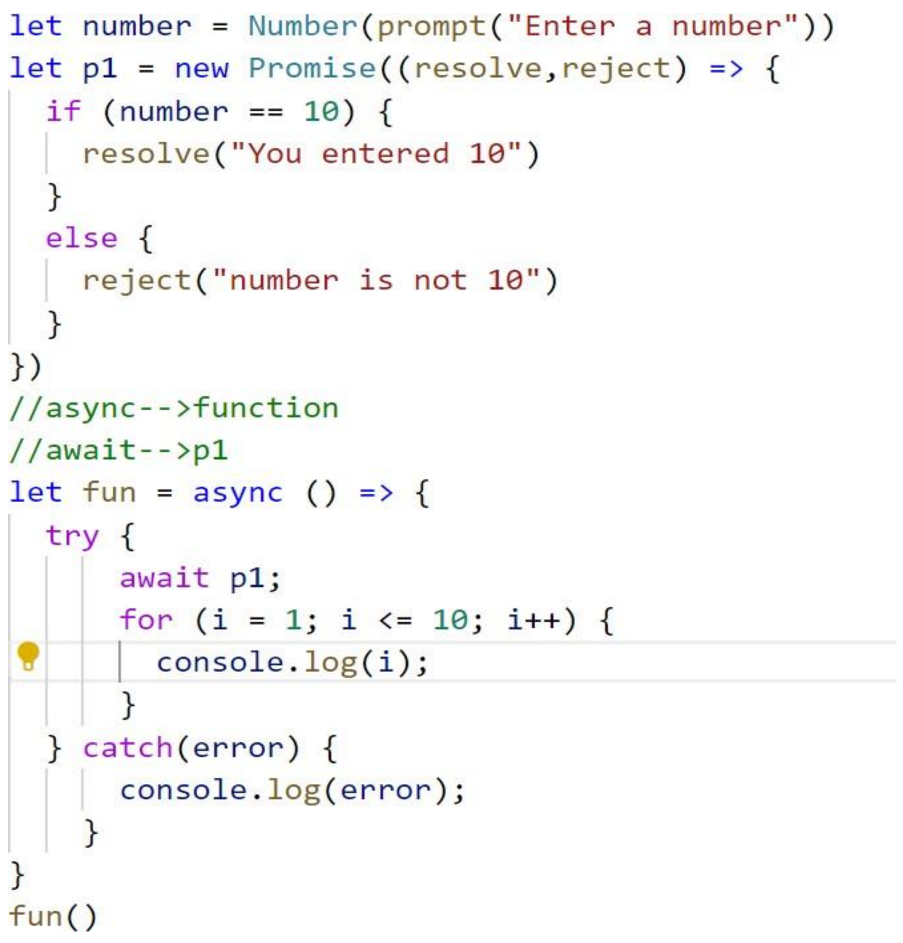
   })

Ex 1: resolving using then and catch



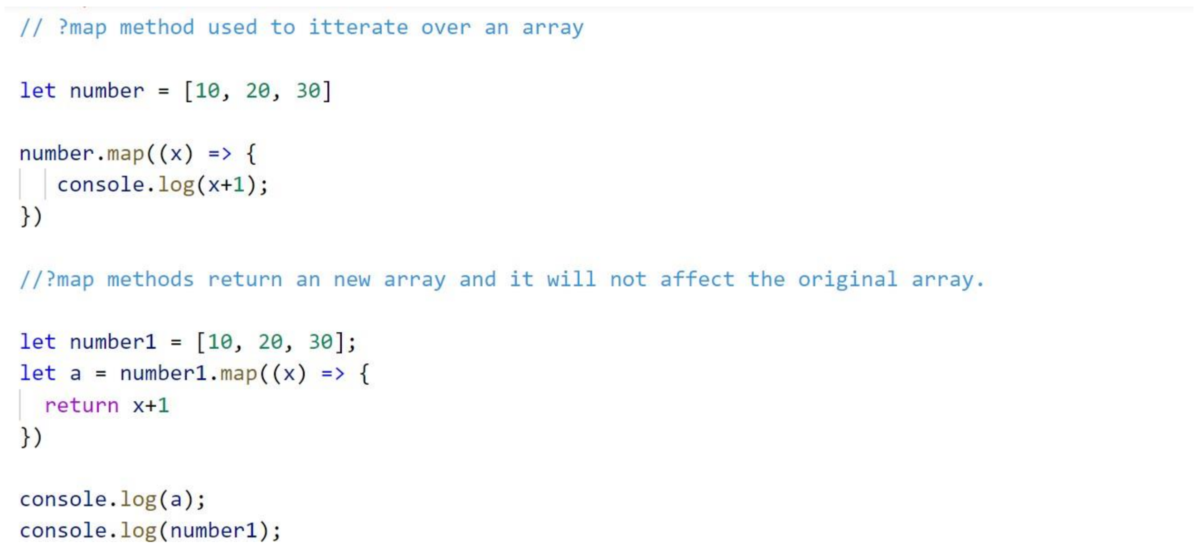
```
//!task input should equal to 10 then print numbers from 1 to 10
//!resolve the promise --1)then and catch
let number = Number(prompt("Enter a number"))

let p1 = new Promise((resolve, reject) => {
  if (number == 10) {
      resolve("you entered 10")
   }
 else{
    reject("number is not 10")
 }
})
//?then block-->resolved
p1.then(() => {
  for (let i = 1; i <= 10; i++){
    console.log(i);
  }
})
//?catch block----->rejected
p1.catch(() => {
  console.log("Error..👿");
})
```

Ex 2: resolving using async and await



```
let number = Number(prompt("Enter a number"))
let p1 = new Promise((resolve,reject) => {
  if (number == 10) {
    resolve("You entered 10")
  }
  else {
    reject("number is not 10")
  }
})
//async-->function
//await-->p1
let fun = async () => {
  try {
      await p1;
      for (i = 1; i <= 10; i++) {
        console.log(i);
      }
  } catch(error) {
      console.log(error);
    }
}
fun()
```

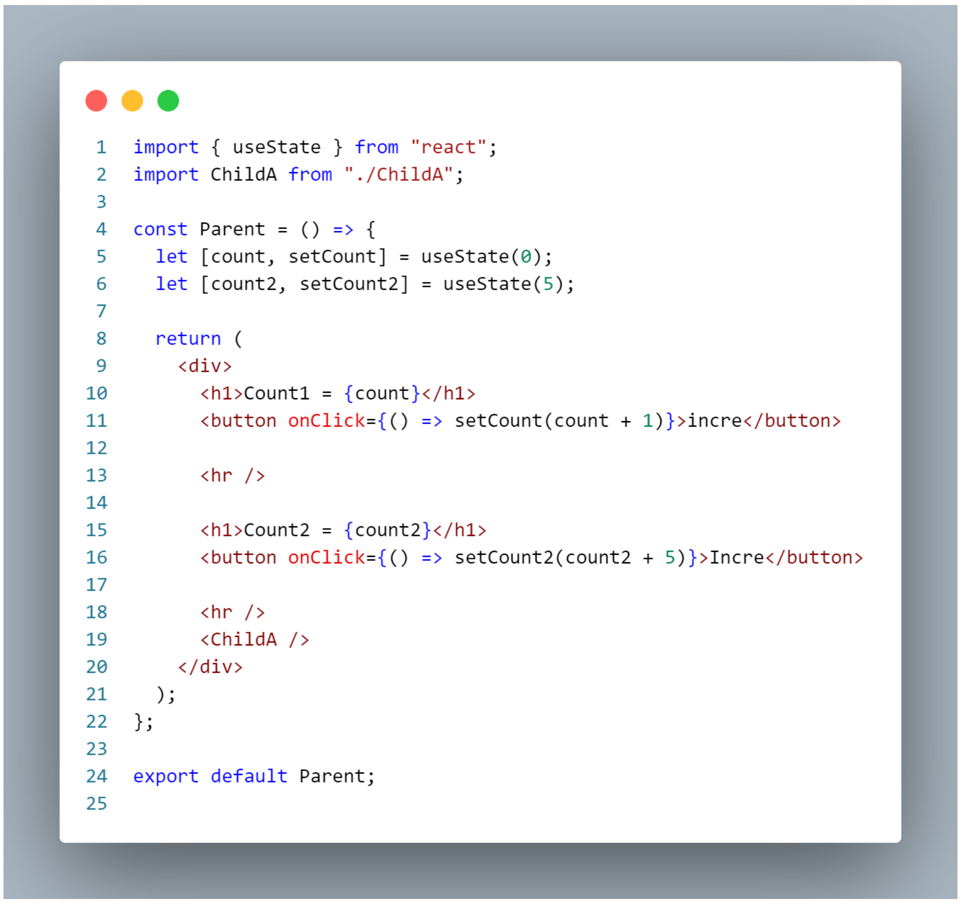
Map method



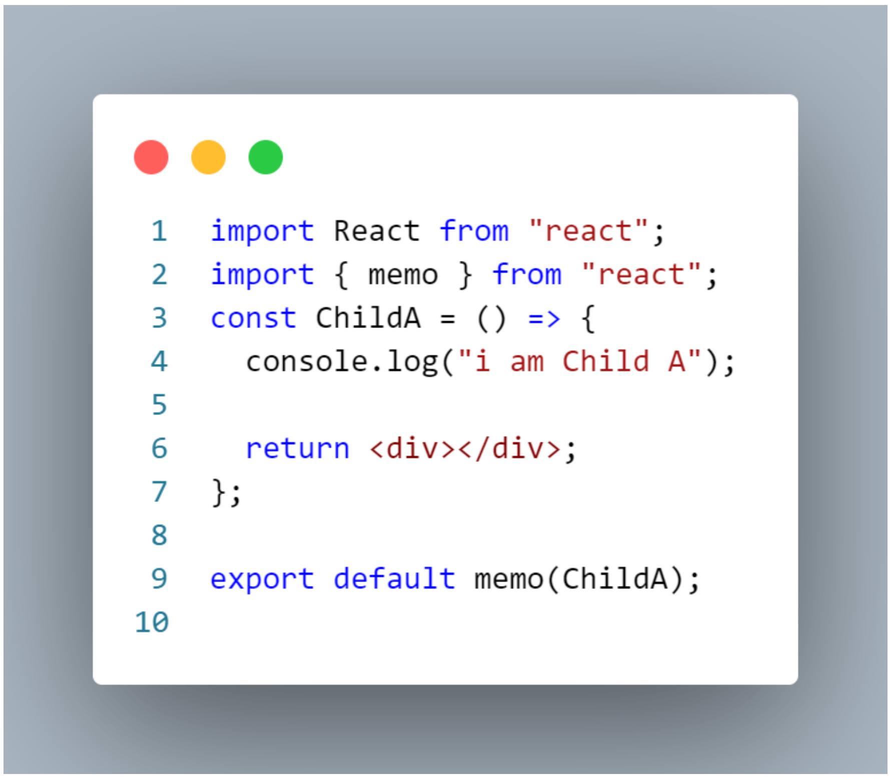
```
// ?map method used to itterate over an array

let number = [10, 20, 30]

number.map((x) => {
    console.log(x+1);
})

//?map methods return an new array and it will not affect the original array.

let number1 = [10, 20, 30];
let a = number1.map((x) => {
  return x+1
})

console.log(a);
console.log(number1);
```

React Memo

- ✓ Memo will memorize the entire component due to which if a state having non-primitive data is not changing actually then component will not re-render.
- ✓ Memo used to skip the re-rendering of child components when the parent components renders in FBC.
- ✓ The component which you don’t want re-render simply wrap those componets with React.memo
- ✓ Memo is one Higher Order Component.



```jsx
import { useState } from "react";
import ChildA from "./ChildA";

const Parent = () => {
  let [count, setCount] = useState(0);
  let [count2, setCount2] = useState(5);

  return (
    <div>
      <h1>Count1 = {count}</h1>
      <button onClick={() => setCount(count + 1)}>incre</button>

      <hr />

      <h1>Count2 = {count2}</h1>
      <button onClick={() => setCount2(count2 + 5)}>Incre</button>

      <hr />
      <ChildA />
    </div>
  );
};

export default Parent;
```



```
import React from "react";
import { memo } from "react";
const ChildA = () => {
  console.log("i am Child A");

  return <div></div>;
};

export default memo(ChildA);

```

useMemo( ) :

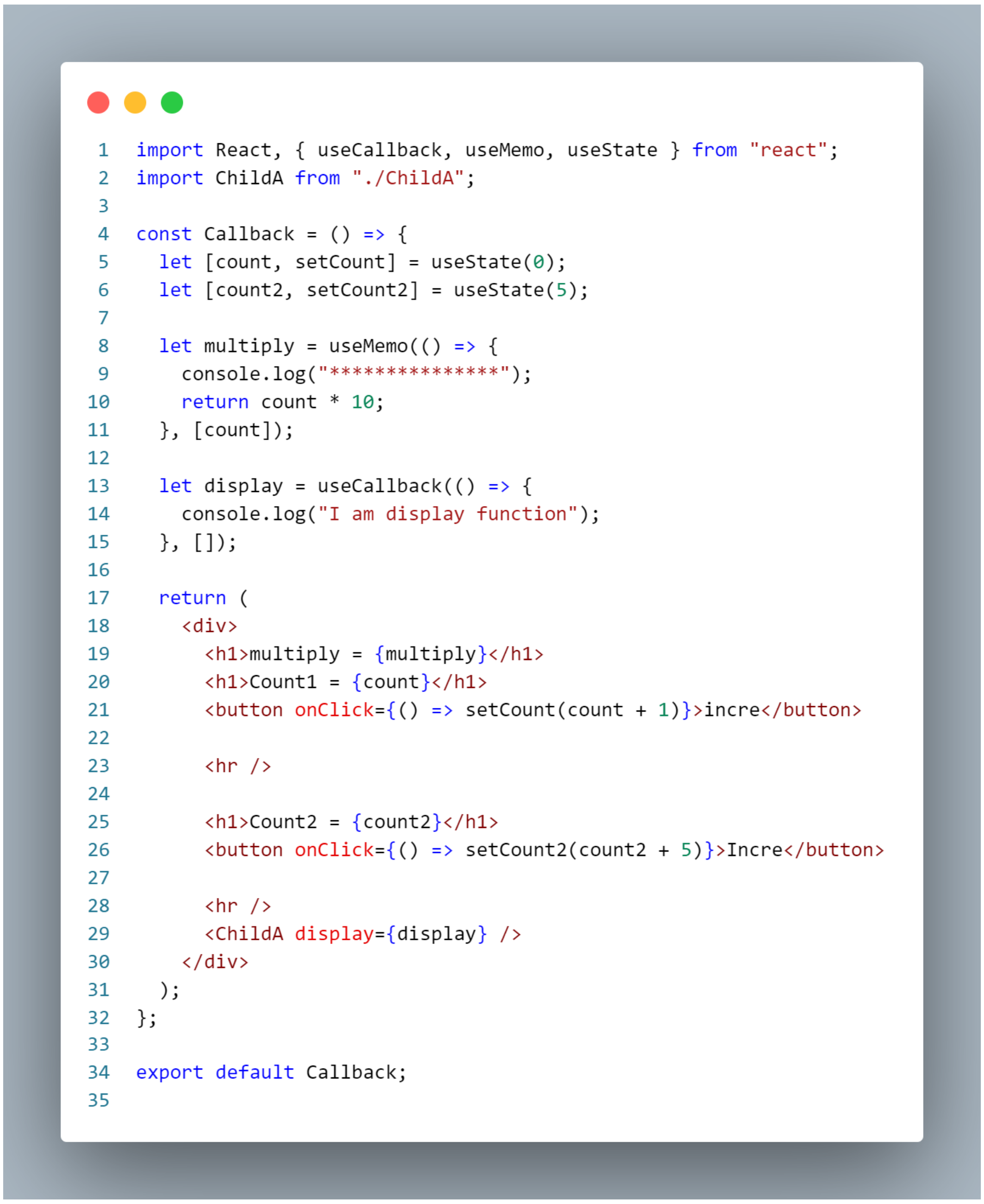
- ✓ useMemo it is a hook.
- ✓ It will memorize a particular value in an outcome of a function or resultsof a function. So that other functions will not be effected.
- ✓ useMemo () accepts 2 values
  1) A function that performs the expensive computation.
  2) An array of dependencies. The memorized value will only be recalculatedif any of the dependencies have changed.

```
import { useMemo, useState } from "react";

const Parent = () => {
  let [count, setCount] = useState(0);
  let [count2, setCount2] = useState(5);

  let multiply = useMemo(() => {
    console.log("***************");
    return count * 10;
  }, [count]);

  return (
    <div>
      <h1>multiply = {multiply}</h1>
      <h1>Count1 = {count}</h1>
      <button onClick={() => setCount(count + 1)}>incre</button>

      <hr />

      <h1>Count2 = {count2}</h1>
      <button onClick={() => setCount2(count2 + 5)}>Incre</button>
    </div>
  );
};

export default Parent;
```

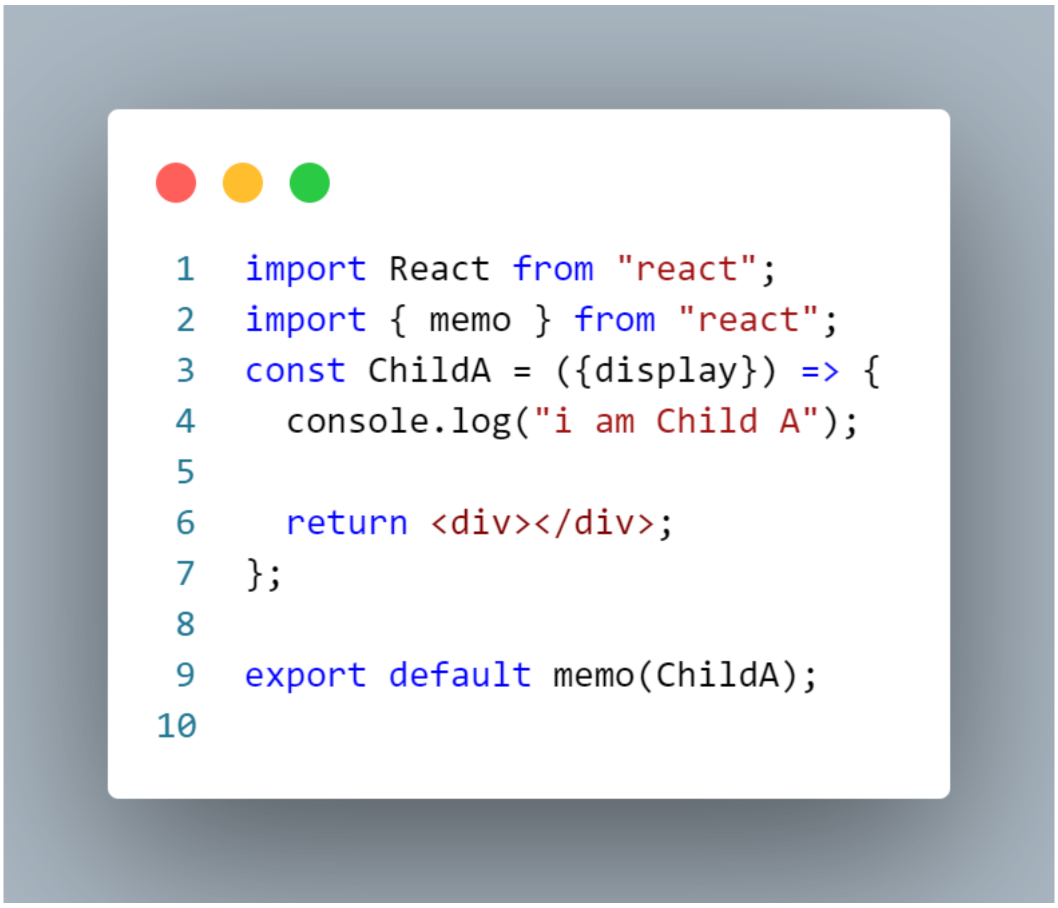
- ✓ In the above example heavy function is applicable only for state count1. So we wrapped that function with useMemo( ) so it will memorized the value of that particular function.
- ✓ Here we have to pass the dependency to indicate that it should be applicable for only count1 state variable

useCallback( )

- ✓ useCallback( ) it is a hook.
- ✓ It will memorized the entire function instead of the value returned by the function.



```jsx
import React, { useCallback, useMemo, useState } from "react";
import ChildA from "./ChildA";

const Callback = () => {
  let [count, setCount] = useState(0);
  let [count2, setCount2] = useState(5);

  let multiply = useMemo(() => {
    console.log("***************");
    return count * 10;
  }, [count]);

  let display = useCallback(() => {
    console.log("I am display function");
  }, []);

  return (
    <div>
      <h1>multiply = {multiply}</h1>
      <h1>Count1 = {count}</h1>
      <button onClick={() => setCount(count + 1)}>incre</button>

      <hr />

      <h1>Count2 = {count2}</h1>
      <button onClick={() => setCount2(count2 + 5)}>Incre</button>

      <hr />
      <ChildA display={display} />
    </div>
  );
};

export default Callback;

```



```jsx
import React from "react";
import { memo } from "react";
const ChildA = ({display}) => {
  console.log("i am Child A");

  return <div></div>;
};

export default memo(ChildA);
```

Routing :

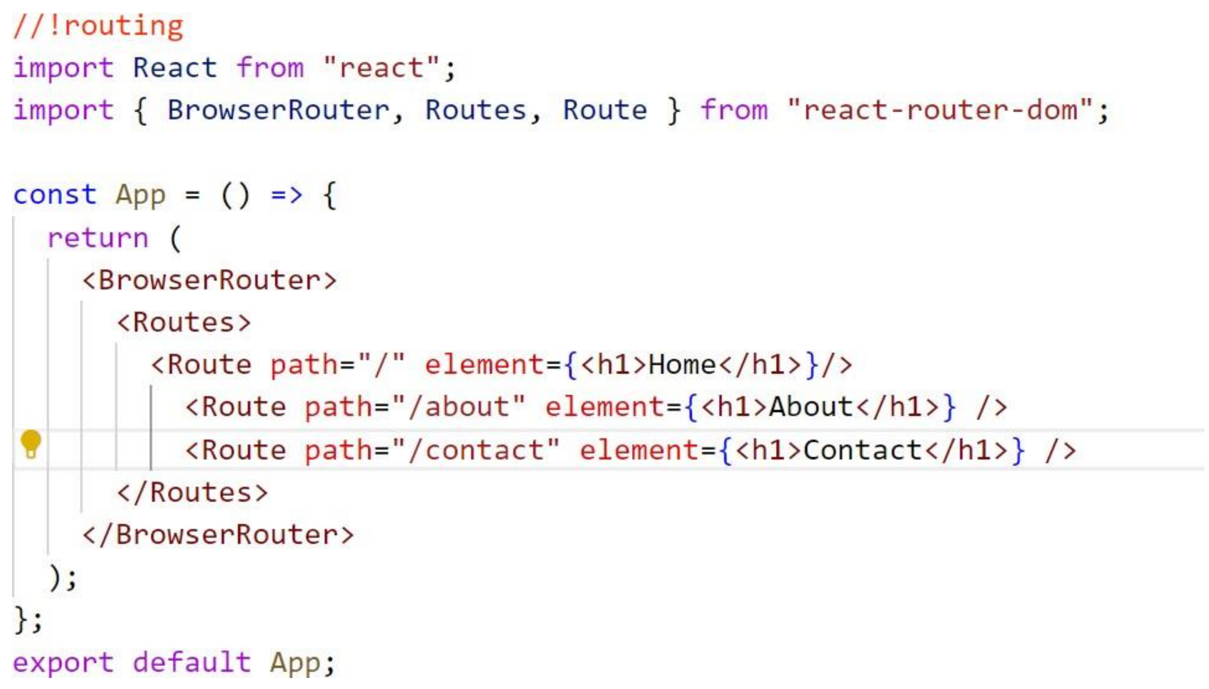
- ✓ Routing in React allows you to build single-page applications (SPAs) by handling navigation and rendering different components based on the URL.
- ✓ This helps create a seamless user experience without the need to refresh the page.

Steps:

1) Install React Router:
   - ✓ React Router is a popular library for handling routing in React. You can install it using npm:

     npm install react-router-dom
2) Import the React Router Dom and SetUp the Routes:

App.jsx



```
//!routing
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";

const App = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<h1>Home</h1>}/>
          <Route path="/about" element={<h1>About</h1>} />
          <Route path="/contact" element={<h1>Contact</h1>} />
      </Routes>
    </BrowserRouter>
  );
};
export default App;
```

❖ **BrowserRouter :**

- ✓ It is used to connect our application URL with the URL of the browser address bar.
- ✓ It should wrap your entire application to enable routing.

❖ **Routes :**

- ✓ **Routes** is a component provided by React Router that is used to define the routes in your application.

- ✓ It renders the first **Route** that matches the current location.

❖ **Route:**

- ✓ **Route** is a component provided by React Router that renders some UI when its path matches the current URL.
- ✓ It takes a **path** prop to specify the URL path to match, and an **element** prop to define what should be rendered when the route matches.

❖ **path:**

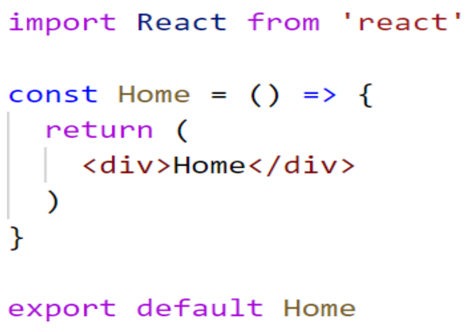
- ✓ **path** is a prop that you give to a **Route** component to specify the URL path it should match. It is a string that represents the path you want to route to.

❖ **element:**

- ✓ **element** is a prop that you give to a **Route** component to specify what should be rendered when the route matches. This can be a component, an element, or any valid React JSX.

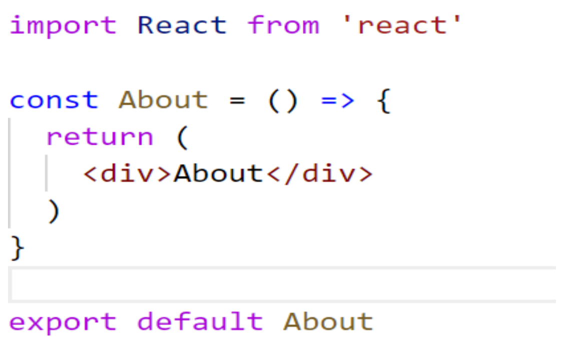
3) Create separate Components render that.

Home.jsx



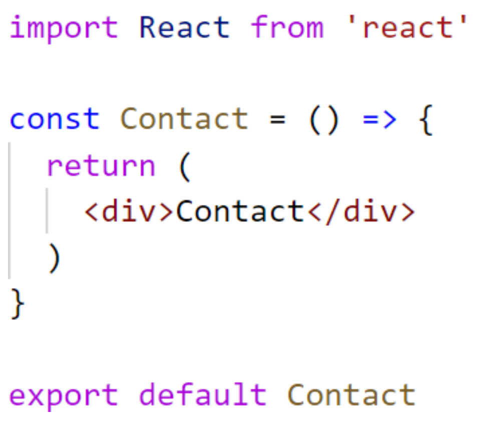
```jsx
import React from 'react'

const Home = () => {
  return (
    <div>Home</div>
  )
}

export default Home
```

About.jsx



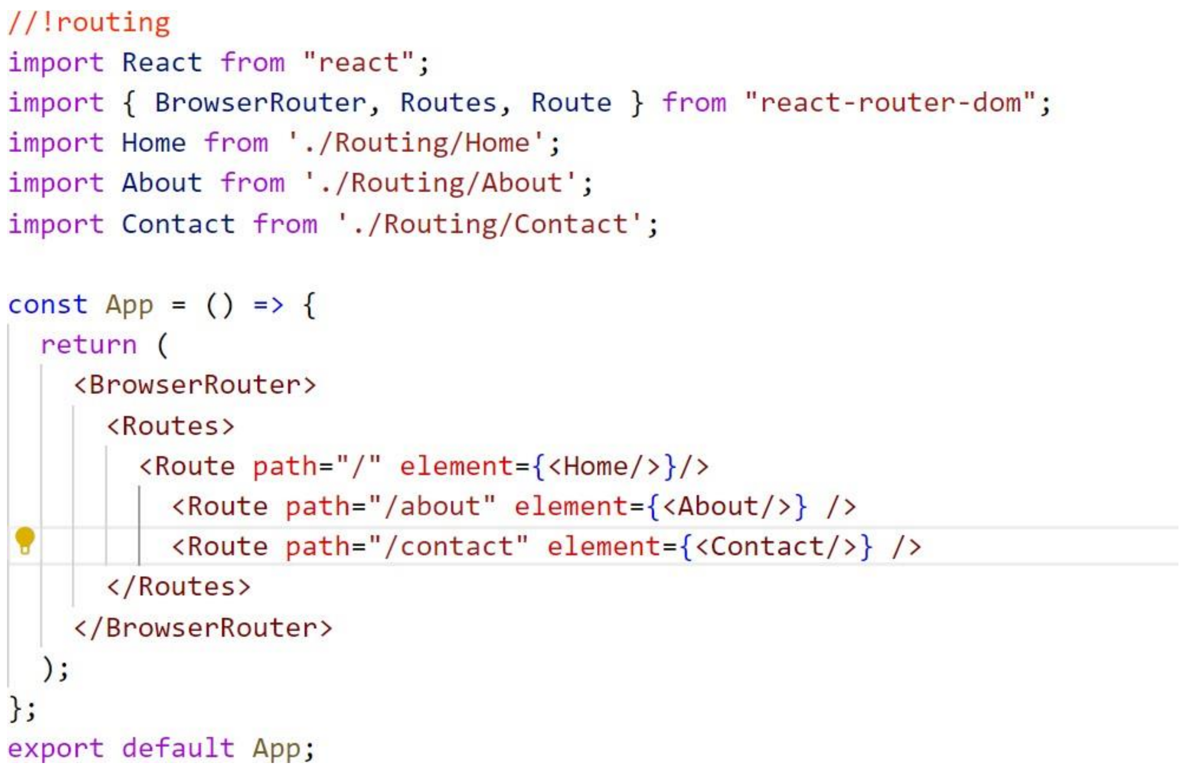
```jsx
import React from 'react'

const About = () => {
  return (
    <div>About</div>
  )
}

export default About
```

Contact.jsx



```jsx
import React from 'react'

const Contact = () => {
  return (
    <div>Contact</div>
  )
}

export default Contact
```

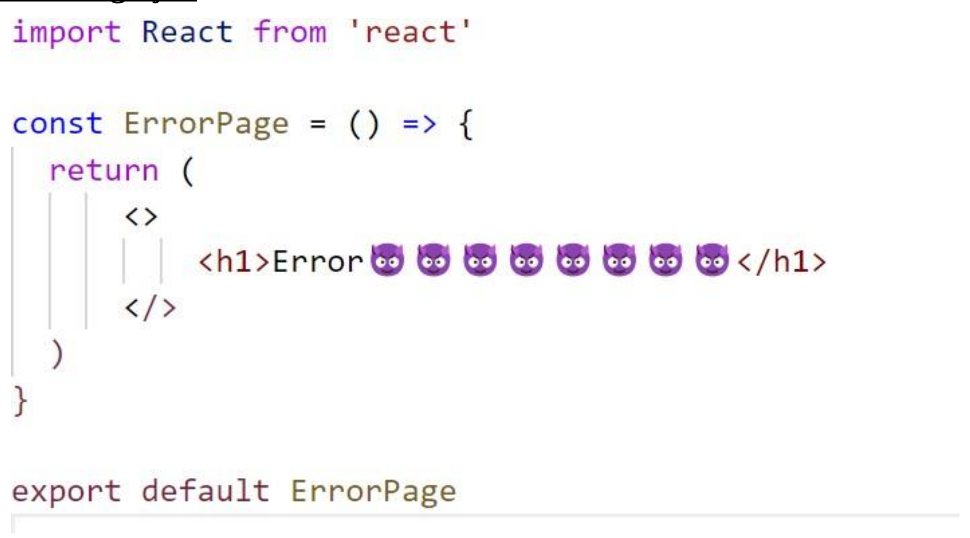
App.jsx



```jsx
//!routing
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from './Routing/Home';
import About from './Routing/About';
import Contact from './Routing/Contact';

const App = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home/>}/>
          <Route path="/about" element={<About/>} />
          <Route path="/contact" element={<Contact/>} />
      </Routes>
    </BrowserRouter>
  );
};
export default App;
```

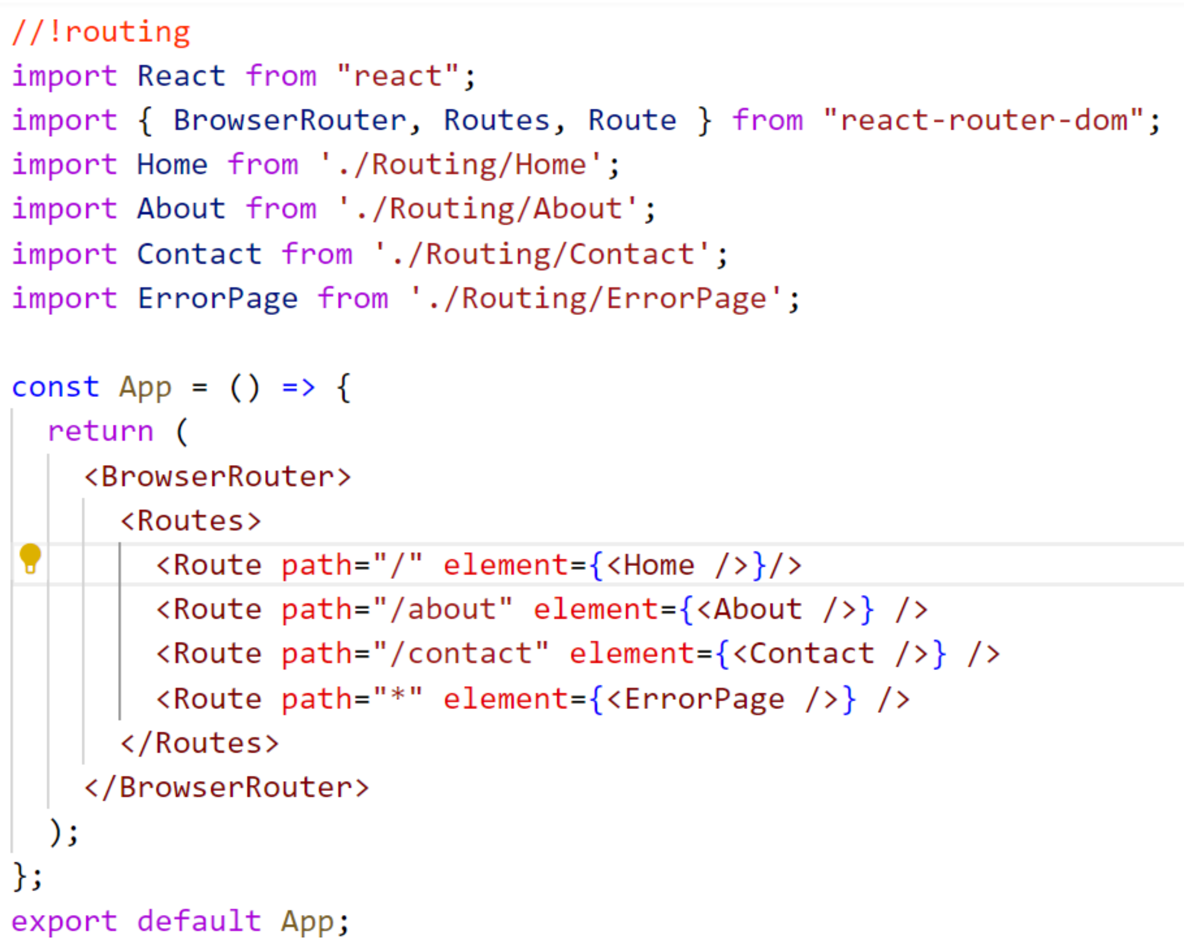
4) Inorder to display the error page for wrong URL create one Component and render it.

ErrorPage.jsx



```
import React from 'react'

const ErrorPage = () => {
  return (
    <>
      <h1>Error😈😈😈😈😈😈😈😈</h1>
    </>
  )
}

export default ErrorPage
```

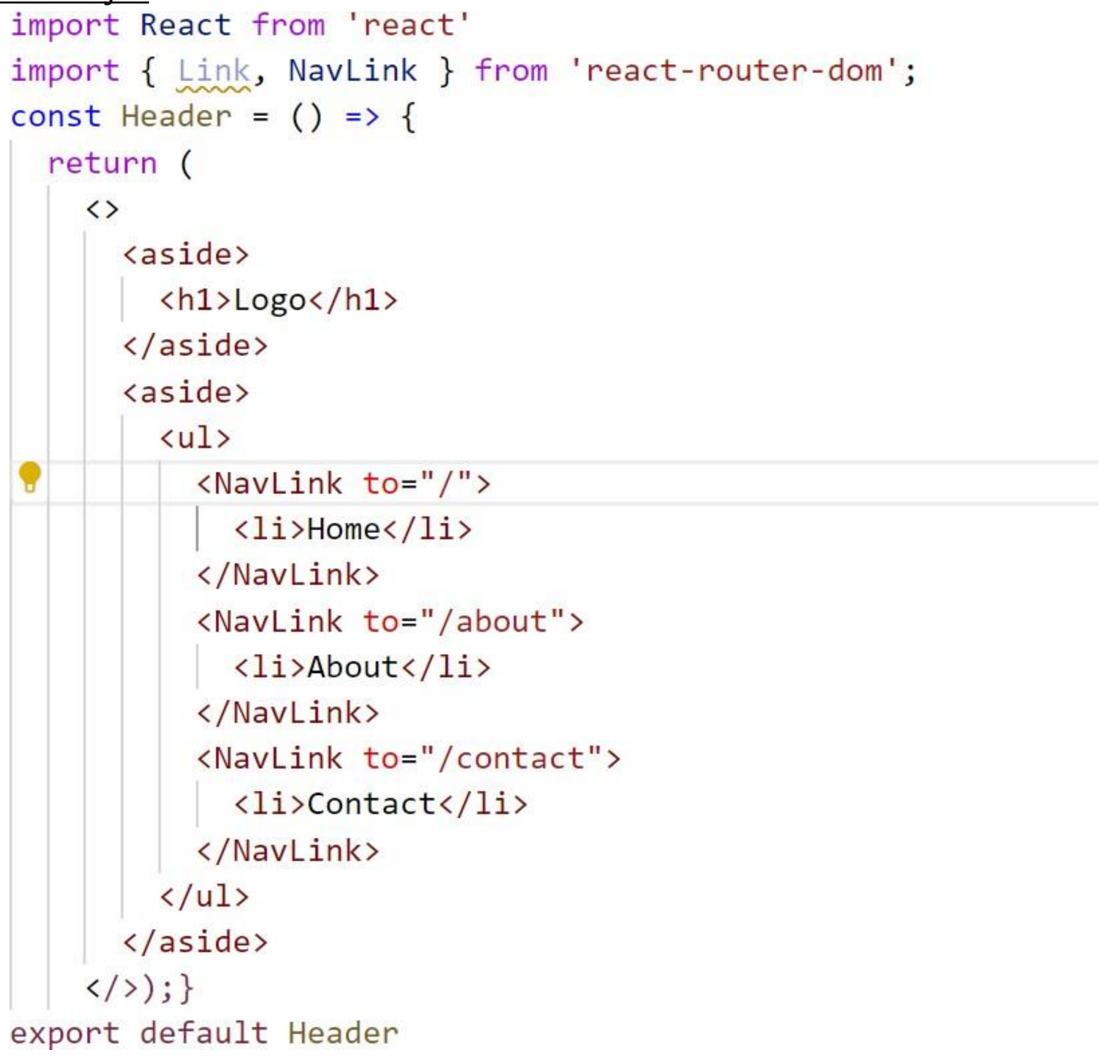
App.jsx



```
//!routing
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from './Routing/Home';
import About from './Routing/About';
import Contact from './Routing/Contact';
import ErrorPage from './Routing/ErrorPage';

const App = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />}/>
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="*" element={<ErrorPage />} />
      </Routes>
    </BrowserRouter>
  );
};
export default App;
```

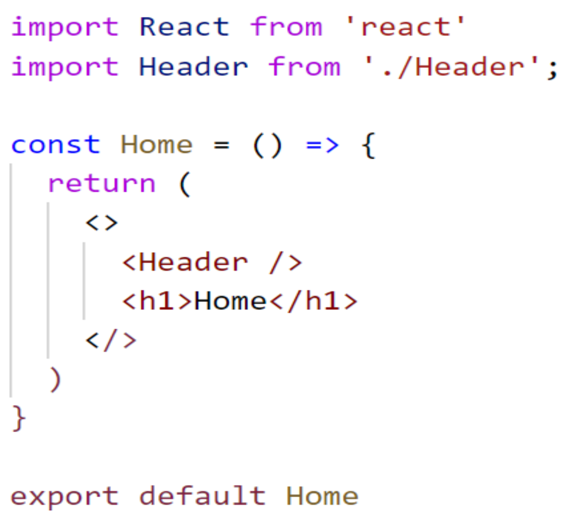
5) Inorder to display the navbar in every page create Header component call in each component.

Header.jsx



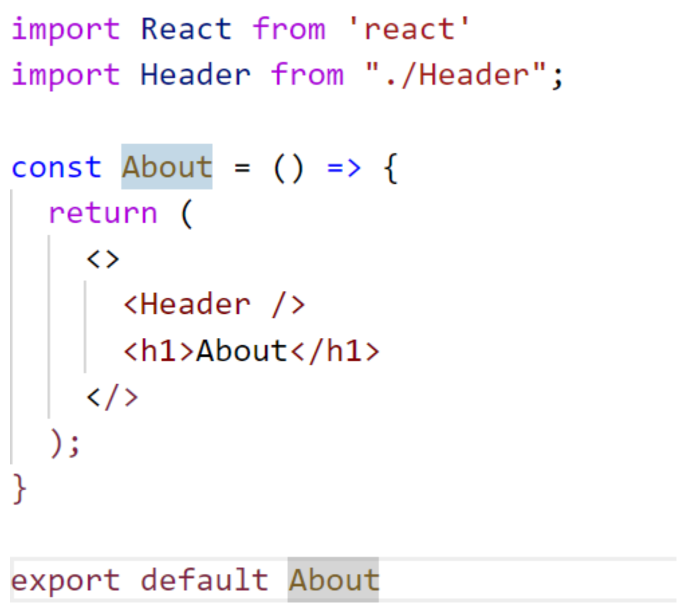
```
import React from 'react'
import { Link, NavLink } from 'react-router-dom';
const Header = () => {
  return (
    <>
      <aside>
        <h1>Logo</h1>
      </aside>
      <aside>
        <ul>
          <NavLink to="/">
            <li>Home</li>
          </NavLink>
          <NavLink to="/about">
            <li>About</li>
          </NavLink>
          <NavLink to="/contact">
            <li>Contact</li>
          </NavLink>
        </ul>
      </aside>
    </>);}
export default Header
```

Home.jsx



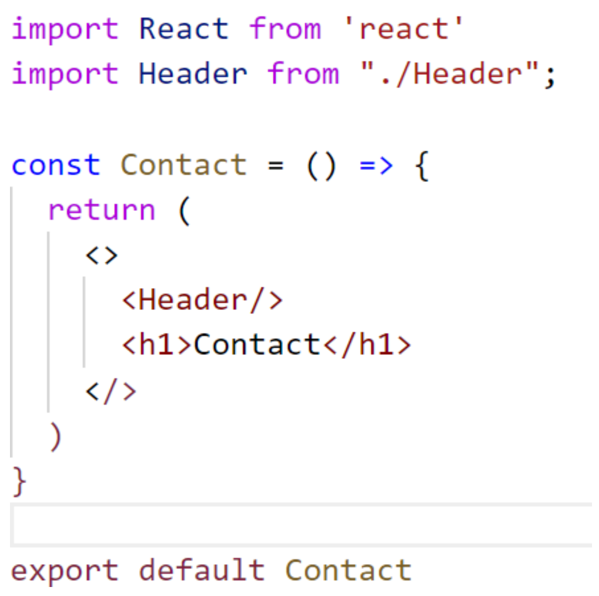
```
import React from 'react'
import Header from './Header';

const Home = () => {
  return (
    <>
      <Header />
      <h1>Home</h1>
    </>
  )
}

export default Home
```

About.jsx



```jsx
import React from 'react'
import Header from "./Header";

const About = () => {
  return (
    <>
      <Header />
      <h1>About</h1>
    </>
  );
}

export default About
```

Contact.jsx



```jsx
import React from 'react'
import Header from "./Header";

const Contact = () => {
  return (
    <>
      <Header/>
      <h1>Contact</h1>
    </>
  )
}

export default Contact
```

App.jsx

- ✓ Link :
  - The **Link** component is a part of the React Router library. It is used to create a hyperlink to navigate between different routes in a React application. It renders an anchor **<a>** tag with an **href** attribute that points to the specified route.
- ✓ NavLink:
  - **NavLink** is a component provided by React Router that is used for navigation in a React application. It's similar to **Link**, but it provides additional styling and behavior options for the active link. Specifically, **NavLink** allows you to apply a specific style to a link when it matches the current route.
  - It provides a class attribute with active value like class=”attribute” to activated component.
- ✓ **to :**
  - The **to** prop is used with the **Link** component to specify the target route. It indicates the URL path that the link should navigate to when clicked.

6) Nested Routing and Shared Routing.

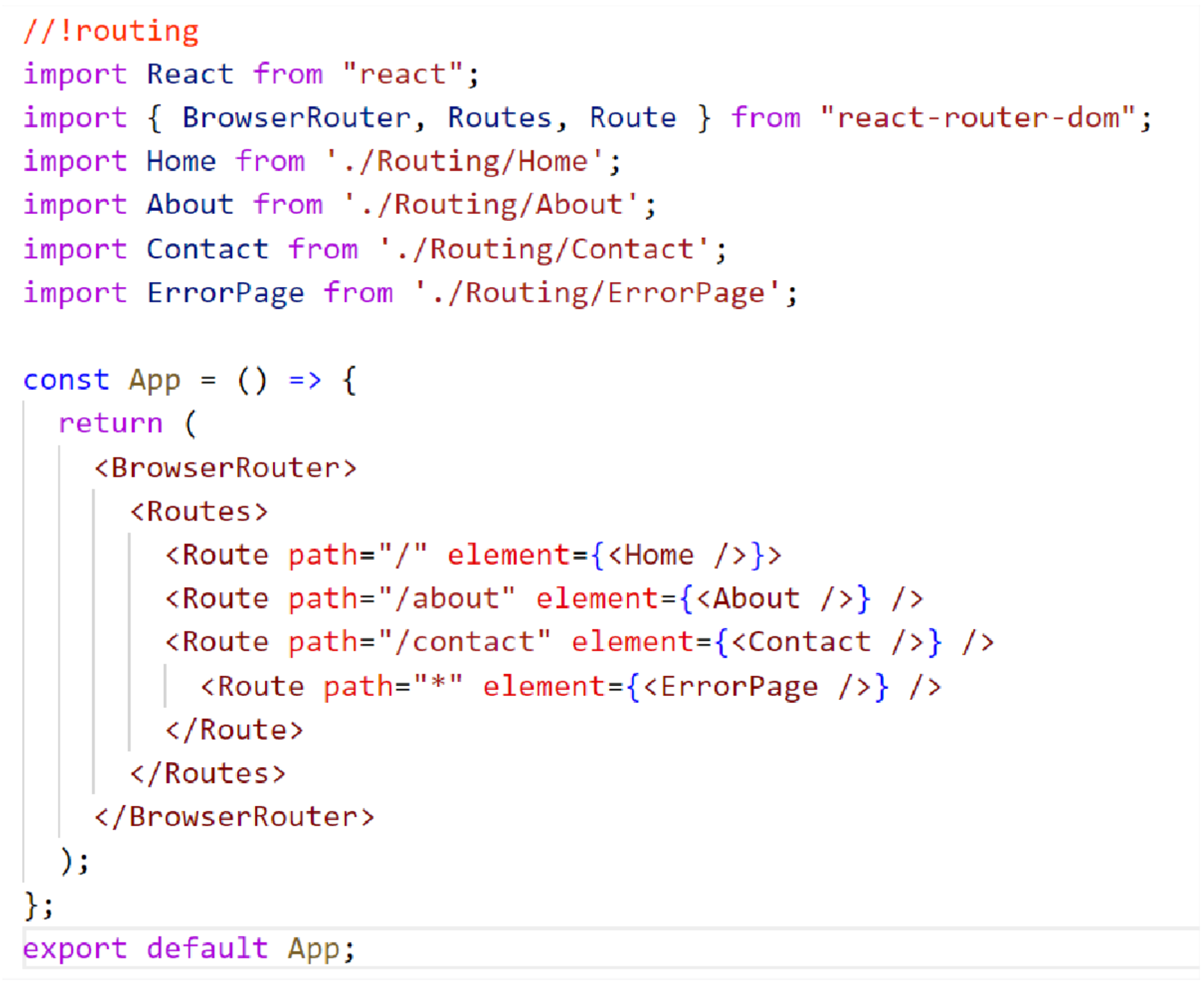
Nested Routing

Nested routing in React refers to the practice of defining routes within other routes. This allows you to have a hierarchical structure of components that correspond to different URL paths.

Shared Routing

- ✓ Shared routing refers to a routing approach in React where multiple components share the same route.
- ✓ We need to mention <Outlet/> in parent component.

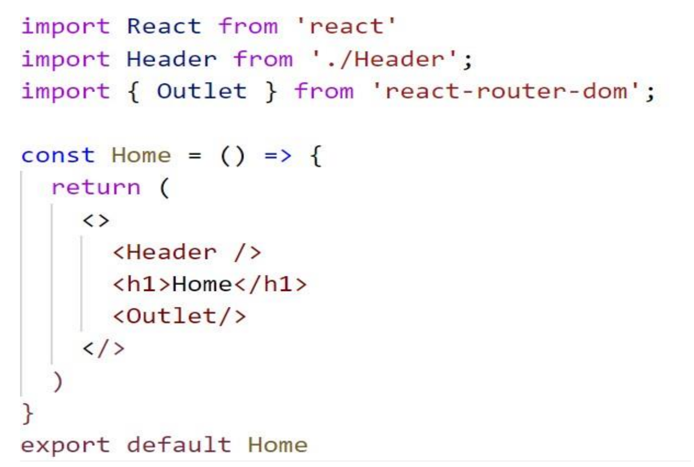
App.jsx



```
//!routing
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from './Routing/Home';
import About from './Routing/About';
import Contact from './Routing/Contact';
import ErrorPage from './Routing/ErrorPage';

const App = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />}>
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
          <Route path="*" element={<ErrorPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );
};
export default App;
```

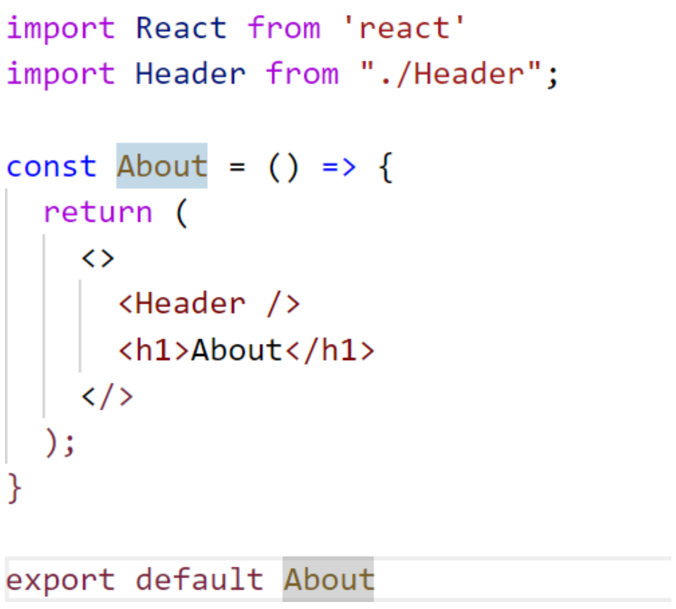
- ✓ Here Home component is acting as a parent component.

Home.jsx

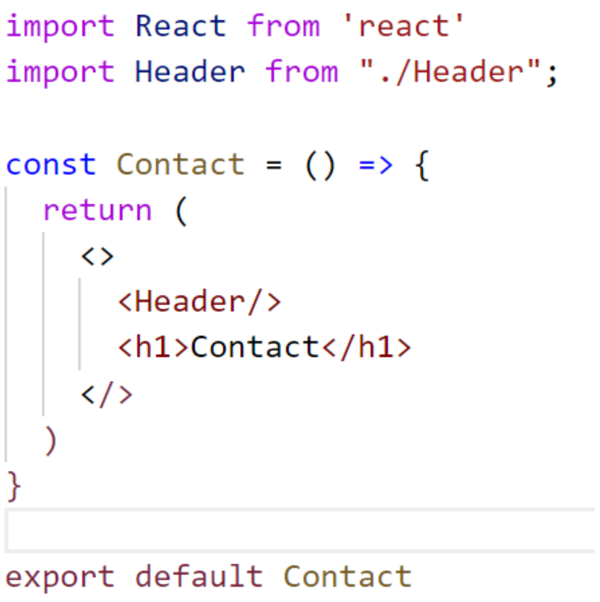


```
import React from 'react'
import Header from './Header';
import { Outlet } from 'react-router-dom';

const Home = () => {
  return (
    <>
      <Header />
      <h1>Home</h1>
      <Outlet/>
    </>
  )
}
export default Home
```

About.jsx


```
import React from 'react'
import Header from "./Header";

const About = () => {
  return (
    <>
      <Header />
      <h1>About</h1>
    </>
  );
}

export default About
```

Contact.jsx


```
import React from 'react'
import Header from "./Header";

const Contact = () => {
  return (
    <>
      <Header/>
      <h1>Contact</h1>
    </>
  )
}

export default Contact
```

- ✓ But here there is one problem that is parent content will also display with child components.
- ✓ So in order to tackle this we have to use index props.

index props

- ✓ Create new component let us say Layout.jsx
- ✓ Whatever the content you want to write in Home component write the content in Layout.

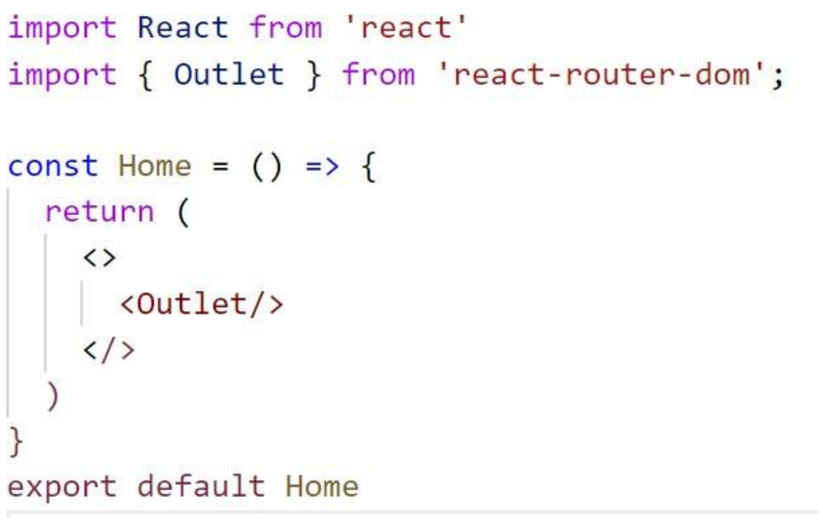
App.jsx

```
//!routing
import React from "react";
import { BrowserRouter, Routes, Route } from "react-router-dom";
import Home from './Routing/Home';
import About from './Routing/About';
import Contact from './Routing/Contact';
import ErrorPage from './Routing/ErrorPage';
import Layout from "./Routing/Layout";

const App = () => {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/" element={<Home />}>
        <Route index element={<Layout/>} />
        <Route path="/about" element={<About />} />
        <Route path="/contact" element={<Contact />} />
        <Route path="*" element={<ErrorPage />} />
        </Route>
      </Routes>
    </BrowserRouter>
  );};
export default App;
```

Layout.jsx

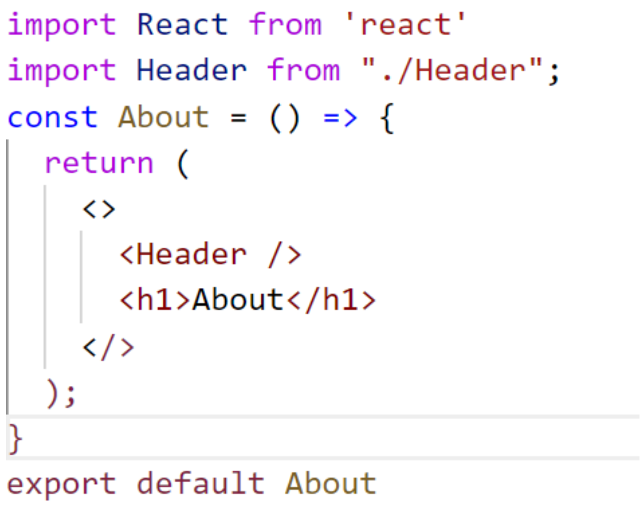
```
import React from 'react'
import Header from './Header';

const Layout = () => {
  return (
    <>
      <Header/>
      <h1>Home page</h1>
    </>
  );
}

export default Layout
```

Home.jsx



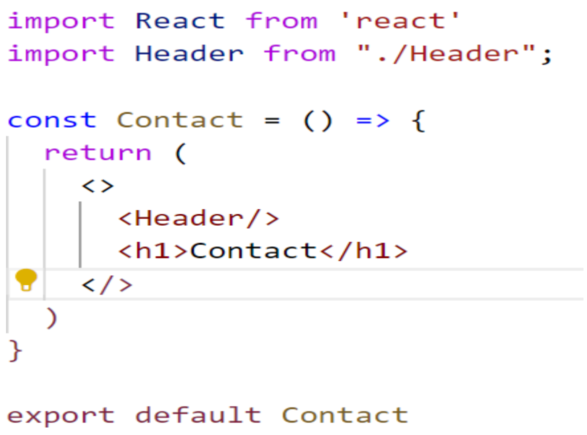
```
import React from 'react'
import { Outlet } from 'react-router-dom';

const Home = () => {
  return (
    <>
      <Outlet/>
    </>
  )
}
export default Home
```

About.jsx



```
import React from 'react'
import Header from "./Header";
const About = () => {
  return (
    <>
      <Header />
      <h1>About</h1>
    </>
  );
}
export default About
```

Contact.jsx



```
import React from 'react'
import Header from "./Header";

const Contact = () => {
  return (
    <>
      <Header/>
      <h1>Contact</h1>
    </>
  )
}

export default Contact
```

Important interview question :

1) Difference between NavLink and Link

1. **Usage**:
   - `Link`: It is a basic component used to create hyperlinks in a React application. It is similar to an `<a>` tag in HTML.
   - `NavLink`: It is a special version of `Link` that is aware of the current URL and can apply a specific styling or class when the link's route matches the current URL. It is often used for navigation menus where you want to highlight the active link.
2. **Active Style/Class**:
   - `Link`: It does not have any built-in way to apply specific styles or classes based on the current route.
   - `NavLink`: It has a built-in `activeClassName` and `activeStyle` props that allow you to apply a specific style or class to the link when it's active.
3. **Active Behavior**:
   - `Link`: It doesn't have any special behavior related to the active route.
   - `NavLink`: It automatically applies the specified styles or classes when the route matches the link's `to` prop.

AXIOS

- Axios are JS library used to make HTTP request from browser.
- Since it is a library we need to install it.

Command to install axios:

```
npm install axios
```

✓ Axios will return promise. So we need to resolve it
✓ Axios returns promise so we need to handled in two ways.

1. Using then and catch 2. Using async and await

```
!axios
import React from 'react'
import { useEffect } from 'react';
import axios from 'axios'
import { useState } from 'react';
import "./global.css"

const App = () => {

    let [d, setData] = useState([])

    let getApi = async () => {
        let {data} = await axios.get("https://api.github.com/users");
        setData(data)
        console.log(d);
    }
    useEffect(() => {
        getApi()
    },[])
  return (
    <section>
      <article>
        {d.map(x => {
          return (
            <ul key={x.id}>
              <h1>{x.id}</h1>
              <img src={x.avatar_url} alt="" />
            </ul>
          );
        })}
      </article>
    </section>
  );
}

export default App



!then catch

import React, { useEffect, useState } from 'react'
import axios from "axios"
```

```
const App = () => {

   let [user, setUser] = useState([])

   useEffect(() => {
      axios.get("https://api.github.com/users").then(x => {

         setUser(x.data)
         console.log(user);
      })
   },[])

  return (
     <div>
        {
           user.map(x => {
              return (
                 <ul>
                    <li>{x.id}</li>
                    <li>{x.login}</li>
                    <li>
                       <img src={x.avatar_url} alt="" />
                    </li>
                 </ul>
              )
           })
        }
     </div>
  )
}

export default App
```

useParams ():

React JS useParams Hook helps to access the parameters of the current route to manage the dynamic routes in the URL. The react-router-dom package has useParams hooks that let you access and use the parameters of the current route as required.

useNavigate ():

The useNavigate() hook is introduced in the React Router v6 to replace the useHistory() hook. In the earlier version, the useHistory() hook accesses the React Router history object and navigates to the other routers using the push or replace methods. It helps to go to the specific URL, forward or backward pages. In the updated version, the React Router's new navigation API provides a useNavigate() hook which is an imperative version to perform the navigation actions with better compatibility

CREATE BROWSER ROUTER

```
import React from "react";
import "./global.css";
import { createBrowserRouter, RouterProvider } from "react-router-dom";
import Home from "./components/Home";
import About from "./components/About";
import Layout from "./components/Layout";
import Register from "./components/Register";
import Login from "./components/Login";
import Profile from "./components/Profile";
import Allusers from "./components/Allusers";
import EditUser from "./components/EditUser";

const App = () => {
  let routes = createBrowserRouter([
    {
      path: "/",
      element: <Layout />,
      children: [
        {
          path: "/",
          element: <Home />,
        },
        {
          path: "/about",
          element: <About />,
        },
        {
          path: "/register",
          element: <Register />,
        },
        {
          path: "/login",
          element: <Login />,
        },
        {
          path: "/profile",
          element: <Profile />,
        },
        {
          path: "/allusers",
          element: <Allusers />,
        },
        {
          path:"/edituser/:userID",
          element:<EditUser/>
        }
      ],
    },
  ]);
  return (
    <div>
      <RouterProvider router={routes}></RouterProvider>
    </div>
  );
};

export default App;
```