PURE COMPONENT

In React, a pure component is a class component that optimizes rendering performance by implementing a shallow comparison of props and state. It is an optimization technique provided by React to prevent unnecessary re-renders of components.

Pure components in React help improve rendering performance by preventing unnecessary re-renders when props and state remain unchanged. They are a useful tool for optimizing the performance of your React applications.

Let see an example
e.g.

```
import React from 'react';
class App extends React.Component {
constructor()
  {
super()
this.state={
       count:1
    }
  }
render() {
    console.log("render")
    return (
<div>
<h1> Pure Comonent in React  {this.state.count}</h1>
          {/* <button onClick={()=>{this.setState({count:this.state.count+1})}} >Increase </button> */}
<button onClick={()=>{this.setState({count:1})}} >Increase </button>

</div>
    );
  }
}
export default App;
```

create pure compoent

To create a pure component in React, you can extend the ***React.PureComponent*** class or use the React.memo higher-order component.

```
import React from 'react';

class MyComponent extends React.PureComponent {
render() {
   // Component rendering logic
  }
}
```

Solution

```
import React from 'react';
class App extends React.PureComponent {
constructor()
  {
super()
this.state={
       count:1
    }
  }
render() {
    console.log("render")
    return (
<div>
<h1> Pure Comonent in React  {this.state.count}</h1>
<button onClick={()=>{this.setState({count:this.state.count+1})}} >Increase </button>
          {/* <button onClick={()=>{this.setState({count:1})}} >Increase </button> */}

</div>
    );
  }
}
export default App;
```

Exmple2 :
User.js

```
import React from 'react';
class User extends React.PureComponent {
render() {
    console.log("render")
    return (
<div>
<h1> User Comonent in React {this.props.count} </h1>
</div>
    );
  }
}

export default User;
```

App.js

```
import React from 'react';
import User from "./User"
class App extends React.Component {
constructor()
  {
super()
this.state={
       count:1
    }
  }
render() {
    console.log("render")
    return (
<div>
<User count ={this.state.count}/>
          {/* <button onClick={()=>{this.setState({count:this.state.count+1})}} >Increase </button> */}
<button onClick={()=>{this.setState({count:1})}} >Increase </button>

</div>
    );
  }
}
export default App;
```

USEMEMO HOOK

In React, the useMemo hook is used to memoize the result of a function or computation, preventing unnecessary recalculations on every render. It allows you to optimize performance by caching the result of an expensive computation and reusing it when the dependencies haven't changed.

Problem
App.js

```
import React, { useState } from "react";
function App() {
 const [count, setCount] = useState(0);
 const [item, setItem] = useState(10);
function  change()
 {
  console.log("change")
  return count * 2
 }

 return (
<>
<h1> useMemo hook </h1>
<h2>count : {count} </h2>
<h2>item : {item} </h2>
<h2> {change()}</h2>
<button
onClick={() => {
setCount(count + 1);
```

```
      }}
>
Increase{" "}
</button>

<button
onClick={() => {
setItem(item * 10);
      }}
>
update{" "}
</button>
</>
  );
}

export default App;
```

Solution

```
import React, { useState, useMemo } from "react";
function App() {
  const [count, setCount] = useState(0);
  const [item, setItem] = useState(10);

  const changeMemo = useMemo(function change() {
    console.log("change");
    return count * 2;
  }
,[count]);

  return (
<>
<h1> useMemo hook </h1>
<h2>count : {count} </h2>
<h2>item : {item} </h2>
<h2> {changeMemo}</h2>

<button
onClick={() => {
setCount(count + 1);
      }}
>
Increase{" "}
</button>

<button
onClick={() => {
setItem(item * 10);
      }}
>
update{" "}
</button>
</>  );}export default App;
```