EXPERIMENT - 22

Explain React Hooks with an Example

React Hooks are special functions that let you "hook into" React features, such as state and lifecycle methods, in functional components. Before Hooks were introduced, you could only use state and other React features in class components. With Hooks, you can use state and other features in functional components, making them more powerful and versatile.

1] UseState Hook:

The useState hook is one of the most commonly used hooks in React. It allows you to add state to functional components, enabling them to hold and manage data that can change over time, such as user input, dynamic content, or UI interactions.

Program:

Index.js

```
import React, { useState } from 'react';
import ReactDOM from 'react-dom/client';
function Car() {
  const [car, setCar] = useState({
    name: "Toyota",
    model: "Glanza",
    color: "Cafe White",
    year: 2024
  });
  const changeColor = () => {
    setCar(prevState => ({
      ...prevState,
      color: "Blue"
    }));
  };
  return (
    <>
      <h1>
        My {car.name} of model {car.model} of color {car.color} which is manufactured in {car.year}
      </h1>
      <button onClick={changeColor}>Update Color</button>
```

```
    </>
  );
}
const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(<Car />);
```

OUTPUT:

My Toyota of model Glanza of color Blue which is manufactured in 2024

Update Color

2|UseEffect Hook:

useEffect is a Hook in React that allows you to perform side effects in your functional components. Side effects can include things like fetching data from an API, subscribing to a service, or manually updating the DOM. The useEffect hook enables you to run this code after the component has rendered and manage it throughout the component's lifecycle.

Program:

Index.js

```
import React, { useEffect, useState } from 'react';
import ReactDOM from 'react-dom/client';
function Timer() {
  const [count, setCount] = useState(0);
  useEffect(() => {
    const timer = setTimeout(() => {
      setCount((count) => count + 1);
    }, 1000);
    return () => clearTimeout(timer);
  }, [count]);
  return (
    <>
      <h1>I have rendered {count} times</h1>
    </>
  );
}
const root = ReactDOM.createRoot(document.getElementById("root"));
```

root.render(<Timer />);

OUTPUT:

I have rendered 7 times

3]UseContext Hook:

The useContext hook in React is used to access the value of a context within a functional component. Context provides a way to pass data through the component tree without having to pass props down manually at every level. It's particularly useful when you have global data or functionality that needs to be accessed by many components at different levels of the tree.

Program:

```
import React, { useState } from "react"; import ReactDOM from "react-dom/client";

function Component() {   const [user, setUser] = useState("Kumar");   return (
   <div>
     <h1>Hello {user} from component 1</h1>
     <Component2 name={user} />
   </div>
  );
}
function Component2(props) {   const user = props.name;   return (
   <div>
     <h1>Hello  from component 2</h1>
     <Component3 name={user} />
   </div>
  );
}
function Component3(props) {   const user = props.name;   return (
   <div>
     <h1>Hello from component 3</h1>
     <Component4 name={user} />
   </div>
  );
}
```

```
function Component4(props) {   const user = props.name;   return (
   <div>
     <h1>Hello {props.name} from component 4</h1>
   </div>
  );
}
Const   root=ReactDOM.createRoot(document.getElementById("root"));
root.render(<Component />);
```

OUTPUT:

Hello kumara from Component 1

Hello from Component 2

Hello from Component 3

Hello kumara from Component 4

4]UseRef Hook:

The useRef hook in React is used to persist values across renders without causing a re-render when the value changes. It can also be used to directly access and manipulate DOM elements.

Program:

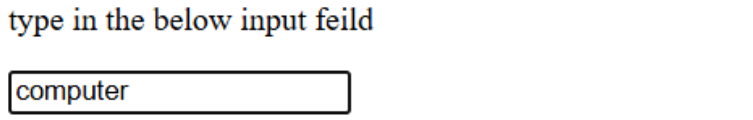
index.js

```
import { useState, useRef, useEffect, use } from "react";
import { createRoot } from "react-dom/client";

function App() {
   const [inputValue, setInputValue] = useState("");
   const count=useRef(0);
   useEffect(() => {
      count.current=count.current+1;
   })
   return (
      <>
      <p>type in the below input feild</p>
      <input type="text" value={inputValue} onChange={(e) => setInputValue(e.target.value)}
/>
      <h1> Render/ charecterCount={count.current}</h1>
      </>
   );
```

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

OUTPUT:



Render/ charecterCount=8

EXPERIMENT - 23

React Router Examples

Index.js

```
import {createRoot} from 'react-dom/client';
import { BrowserRouter, Routes,Route,Link } from 'react-router-dom';
function Home()
{
   return <h1> Home page</h1>;
}
function About()
{
   return <h1> About page</h1>;
}
function Contact()
{
   return<h1> Contact page</h1>;
}
function App()
{
   return (
      <BrowserRouter>
      <nav>
```

```
        <Link to='/'>Home</Link>|{""}
        <Link to='/About'>ABOUT</Link>|{""}
        <Link to='/Contact'>CONTACT</Link>
        </nav>
        <Routes>
          <Route path="/" element={<Home/>}/>
          <Route path="/About" element={<About/>}/>
          <Route path="/Contact" element={<Contact/>}/>
        </Routes>
        </BrowserRouter>
    );
}
createRoot(document.getElementById("root")).render(<App/>);
```

OUTPUT:

EXPERIMENT-24

In react, Create a form that should contain Present Address And Permanent Address in two Text Area Box. When the User Enters any text in any of the one text area box itshould dynamically reflect in another textarea box.implement using react.

Index.js

```
import ReactDom from 'react-dom/client';
import React, { Component } from 'react';

class ParentComponent extends Component {
constructor(props){
super(props);
this.state={name:''};
}
```

```
nameChanged=(val)=>{
this.setState({name:val});
}
render() {
return (
<div>
<PresentAddress name={this.state.name} onNameChange={this.nameChanged}/>
<PermanentAddress name={this.state.name} onNameChange={this.nameChanged}/>
</div>
)
}
}
class PresentAddress extends Component {

handleChange=(e)=>{
this.props.onNameChange(e.target.value);
}
render() {
return (
<div>
<form>
<label>

Present Address:<textarea rows='3' cols='20'
value={this.props.name} onChange={this.handleChange}/>
</label>
</form>
</div>
)
}
}
```

```
class PermanentAddress extends Component {
handleChange=(e)=>{
this.props.onNameChange(e.target.value);
}
render() {
return (
<div>
<form>
<label>
Permanent Address:<textarea rows='3' cols='20'
value={this.props.name} onChange={this.handleChange}/>
</label>
</form>
</div>
)
}
}
const root=ReactDom.createRoot(document.getElementById('root'));
root.render(<ParentComponent />);
```

OUTPUT: