WEEK-1

1. Introduction to Modern JavaScript and DOM

A) **Program Statement: Write a JavaScript program to link JavaScript file with the HTML page**

Description:

This is the foundational step. Students learn to connect an external .js file to an HTML document using<script src="path/to/file.js"></script> thetag, promoting clean code separation.

- **The** <script> **Tag:** This is the HTML element used to embed or reference executable code, most commonly JavaScript.
- src **(source) Attribute:** This attribute of the <script> tag is used to specify the path (URL) to an external JavaScript file.
- **Execution Order:** The browser parses the HTML document from top to bottom. Where you place the <script> tag significantly impacts the performance and behavior of your page.

Source code:

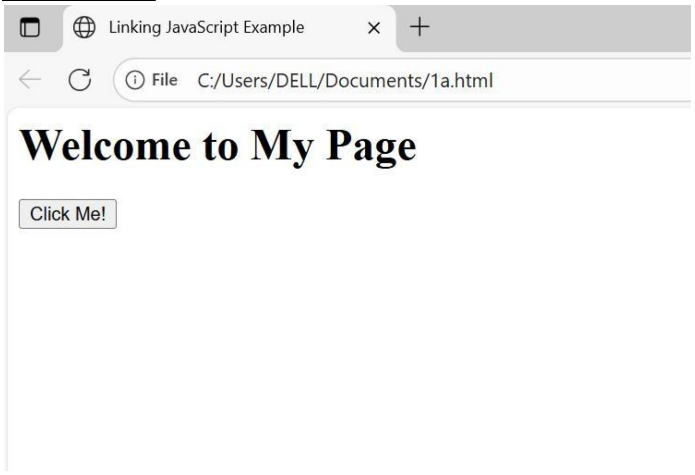
```
<html>

<head>

<title>Linking JavaScript Example</title>

</head>

<body>

<h1>Welcome to My Page</h1>

<button id="myButton">Click Me!</button>

<p id="message"></p>

<!-- Linking the external JavaScript file —>

<script src="script.js"></script>
```

```
</body> </html>
```

Script.js:

```
document.getElementById("myButton").addEventListener("click", function() {

document.getElementById("message").innerText = "Button clicked! Hello from

JavaScript!";

});
```

OUTPUT:


Linking JavaScript Example
File C:/Users/DELL/Documents/1a.html
Welcome to My Page
Click Me!

B) Program Statement: Write a JavaScript program to select the elements in HTML page using selectors.

Description: Selection means identifying specific elements (like a heading, paragraph, button, or image) so that we can apply changes to them.

JavaScript provides different ways to select elements using selectors. Each selector has its own purpose and is useful in different situations:

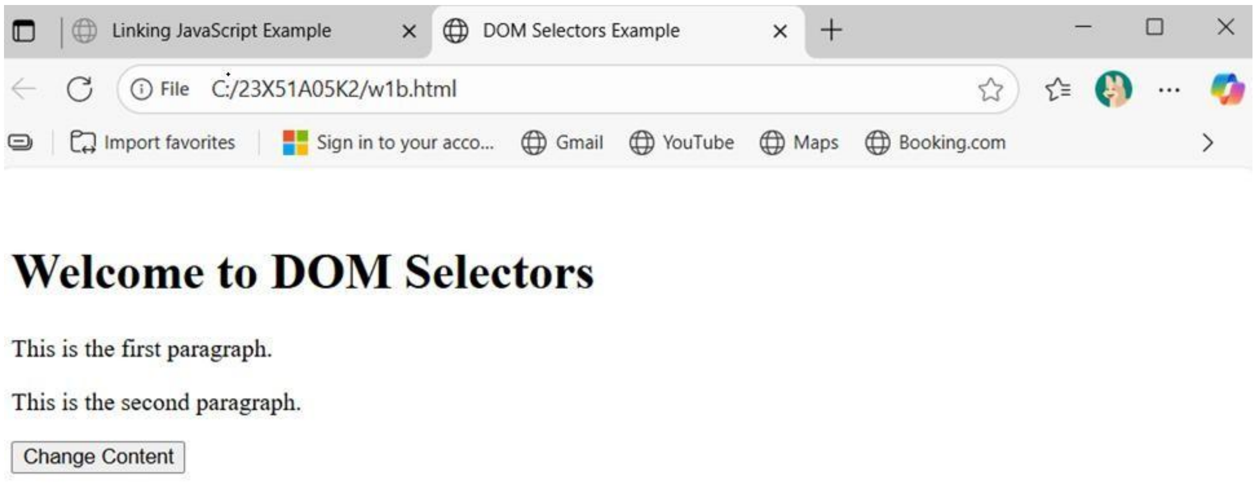
1. getElementById("idName") – Selects one element with the given ID (IDs must be unique).
2. getElementsByClassName("className") – Selects all elements that have the same class name.
3. getElementsByTagName("tagName") – Selects all elements that share the same tag (like all tags).
4. querySelector("CSS_Selector") – Selects the first element that matches a CSS-style selector.
5. querySelectorAll("CSS_Selector") – Selects all elements that match a CSS- style selector.

Source Code:

```
<html>
<head>
<title> DOM selectors Example</title>
<style> .highlight {
background-color: yellow;}
</style>
</head><body>
<h1 id="mainHeader">Welcome to DOM Selectors</h1>
<p class="text">This is the first paragraph.</p>
<p class="text">This is the second paragraph.</p>
<button id="changeButton">Change Content</button>
```

```
<script src="selectors.js"></script>
</body></html>
selectors.js
// Select elements when the button is clicked
document.getElementById("changeButton").addEventListener("click", function() {
// 1. Select by ID const header =
document.getElementById("mainHeader");
header.innerText = "Header Changed!";
// 2. Select by Class (first element) using querySelector const
firstParagraph = document.querySelector(".text");
firstParagraph.innerText = "First paragraph updated!"; // 3.
Select all elements with class 'text' using querySelectorAll
const allParagraphs = document.querySelectorAll(".text");
allParagraphs.forEach((p, index) => {
p.classList.add("highlight"); // Add highlight class
p.innerText = `Paragraph ${index + 1} updated!`;
});
});
```

OUTPUT:



Welcome to DOM Selectors

This is the first paragraph.

This is the second paragraph.

C)Program Statement: Write a JavaScript program to implement the event listeners.

Description: In JavaScript, event listeners are used to handle user interactions such as clicking a button, hovering over text, typing into an input box, or pressing a key. An event listener "listens" for a specific action (event) to occur on an HTML element and then executes a function in response. The most commonly used method is add Event Listener (), which attaches an event to an element. Unlike using inline events (e.g., onclick="..."), event listeners provide more flexibility.

Types of Event Listeners

1. Mouse Events

- click → when the user clicks an element
- dblclick → double click
- mouseover → mouse enters an element
- mouseout → mouse leaves an element
- mousedown / mouseup → pressing and releasing the mouse button
- mousemove → moving the mouse pointer

2. Keyboard Events

- keydown → key is pressed down
- keyup → key is released
- keypress → key is pressed (deprecated in modern JS, prefer keydown)

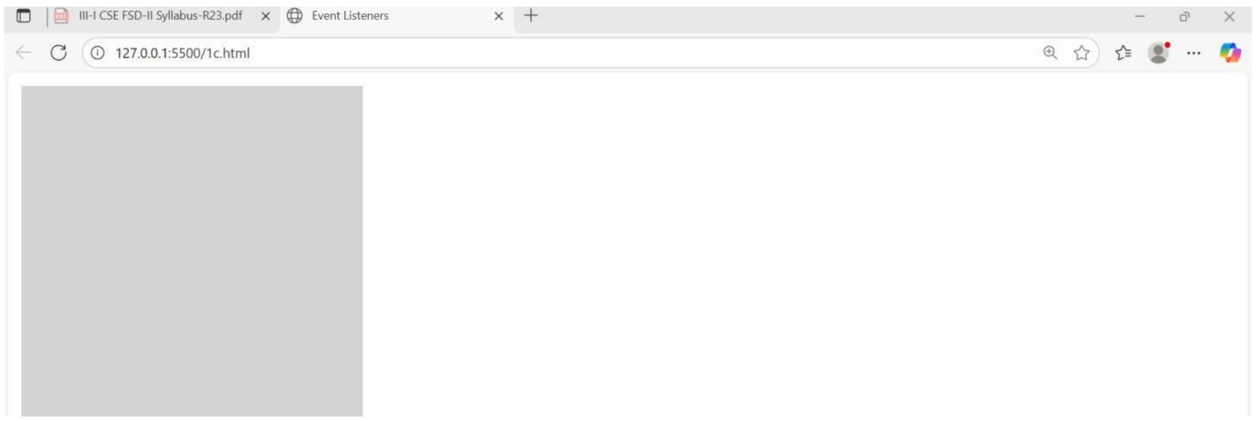
3. Form Events

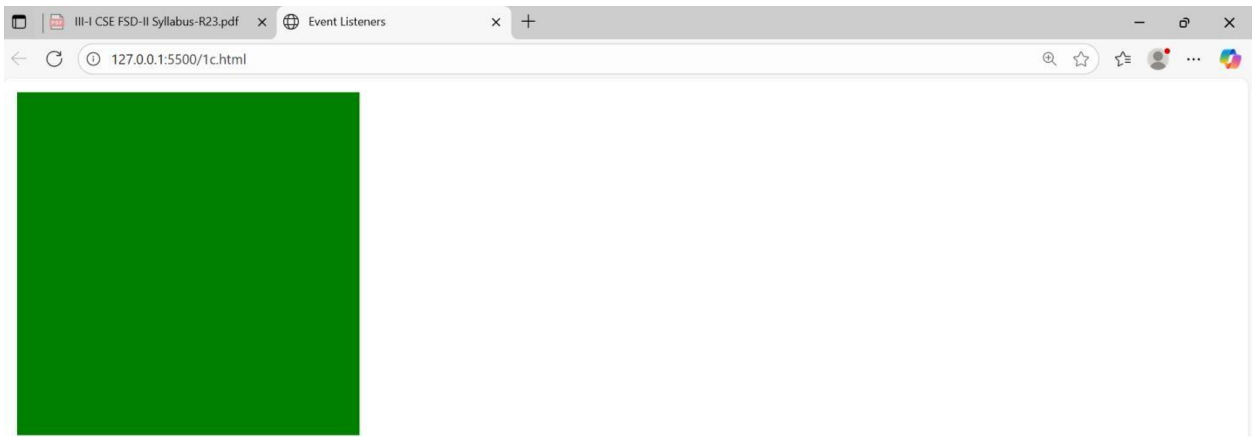
- submit → form submitted
- change → input value changes
- focus → input gains focus
- blur → input loses focus
- input → every time a user types in an input field **Source Code:**

```
<!DOCTYPE html>

<head>
   <title>Event Listeners</title>

   <style>

      .my-element{ width: 200px;

         height: 200px; background-

         color: lightgray;

      }

   </style>

</head>

<body>

   <div id="myDiv" class="my-element"></div>

   <script> function

      changeBackgroundColor(elementId,color){ const element

      = document.getElementById(elementId); if(element){
```

```
    element.addEventListener('mouseenter',()=>{
    element.style.backgroundColor = color;
    });}}
  changeBackgroundColor('myDiv','green');
  </script>
  </body></html>
```

OUTPUT:





D)Program Statement: Write a JavaScript program to handle the click events for the HTML button elements.

Description: In JavaScript, click events are among the most commonly used events in web development. A click event is triggered when a user clicks on an HTML element, most often a button. By handling click events, we can make web pages interactive— for example, displaying messages, changing styles, hiding elements, or performing calculations. Click events can be handled in two main ways:

1. Inline event handling – using onclick directly in the HTML element (not recommended for large projects).

2.Event listeners (addEventListener()) – preferred way because it keeps HTML and JavaScript separate and allows multiple functions to be attached to the same button.

For **buttons**, this is one of the most common events, used to:

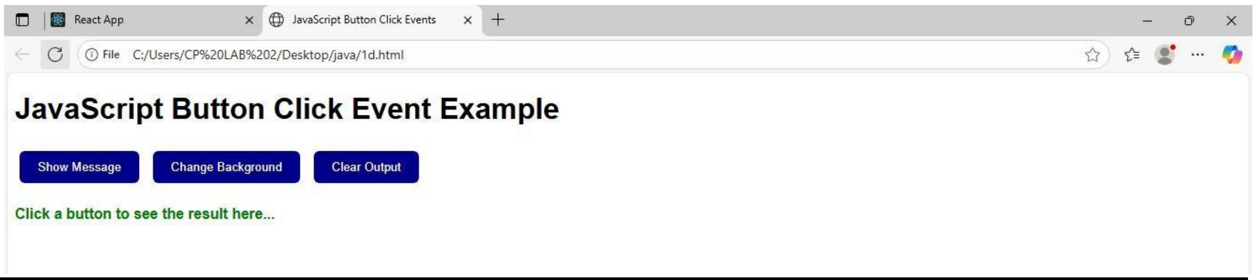
- Submit forms
- Show/hide content
- Trigger calculations
- Call APIs

Source Code:

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>JavaScript Button Click Events</title>
<style>
body {
font-family: Arial, sans-serif;
} button { padding:
10px 20px; margin:
```

```
5px;    border-radius:
6px;
border: none; background-
color: darkblue; color:
white; cursor: pointer; }
button:hover { background-
color: navy;
}
 #output {
margin-top: 20px;
font-weight: bold;
color: green;
}
</style> </head>
<body>
<h1>JavaScript Button Click Event Example</h1>
<button id="btn1">Show Message</button>
<button id="btn2">Change Background</button>
<button id="btn3">Clear Output</button>
<p id="output">Click a button to see the result here...</p
<!-- Linking external JavaScript -->
<script src="clickEvents.js"></script>
</body> </html>
```

OUTPUT:


React App
JavaScript Button Click Events
File C:/Users/CP%20LAB%202/Desktop/java/1d.html

JavaScript Button Click Event Example

Show Message
Change Background
Clear Output

Click a button to see the result here...

E) Program Statement: Write a JavaScript program to With three types of functions

i.Function declaration

ii.Function definition

iii.Arrow functions

Description:

In JavaScript, a function is a block of reusable code that performs a specific task. Functions are very important because they help to avoid repeating the same code again and again, and they make programs easier to read, maintain, and reuse.

The first type is called a **Function Declaration**. This is the traditional way of creating a function by using the function keyword followed by the name of the function. The main feature of function declaration is that it is hoisted, which means we can call the function even before we have written its definition in the program. This makes it simple and flexible for use in many cases.

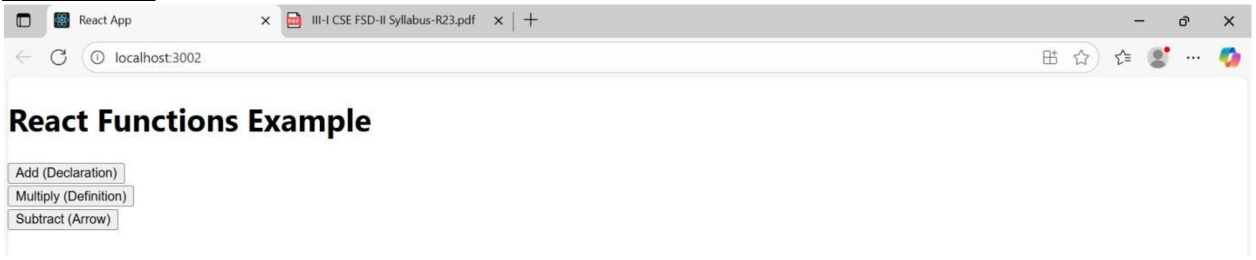
The second type is called a **Function Definition or Function Expression**. In this method, we create a function and store it inside a variable. This gives more control to the programmer and also allows us to create anonymous functions (functions without names) that can be passed around like normal data values.

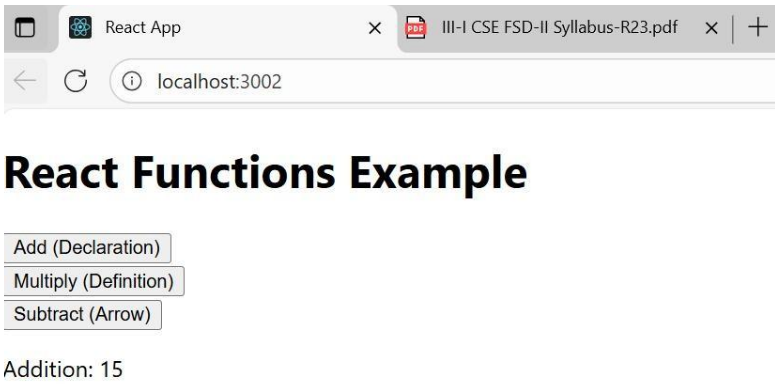
The third and modern type is called an **Arrow Function**, which was introduced in ES6. Arrow functions use the => symbol and provide a much shorter and cleaner syntax compared to normal functions. They are widely used in modern web development, especially in frameworks like ReactJS, because they make the code shorter and also automatically handle this keyword in JavaScript, which is usually tricky for beginners. **Source Code:**

```
import React, { useState } from "react";
export default function Appp() {
  const [result, setResult] = useState("");
  function add(a, b) {
    return a + b;
  }
  const multiply = function (a, b) {
    return a * b;
```

```
 };
 const subtract = (a, b) => a -
 b; const x = 10; const y = 5;
 return (
  <div className="p-6 text-center">
   <h1 className="text-2xl font-bold mb-4">React Functions Example</h1>
   <div className="space-x-4">
    <button onClick={() => setResult(`Addition:
     ${add(x, y)}`)}
     className="px-4 py-2 bg-blue-500 text-white rounded-2xl shadow-md>
     Add (Declaration)
    </button>
    <br></br>

    <button  onClick={()  =>  setResult(`Multiplication:  ${multiply(x,  y)}`)}
     className="px-4 py-2 bg-green-500 text-white rounded-2xl shadow-md">
     Multiply (Definition)
    </button>
    <br></br>

    <button onClick={() => setResult(`Subtraction: ${subtract(x, y)}`)}
     className="px-4 py-2 bg-red-500 text-white rounded-2xl shadow-
     md"> Subtract (Arrow)
    </button>
    <br></br>
   </div>

   <p className="mt-6 text-xl font-semibold">{result}</p>
  </div>
 );
}
```

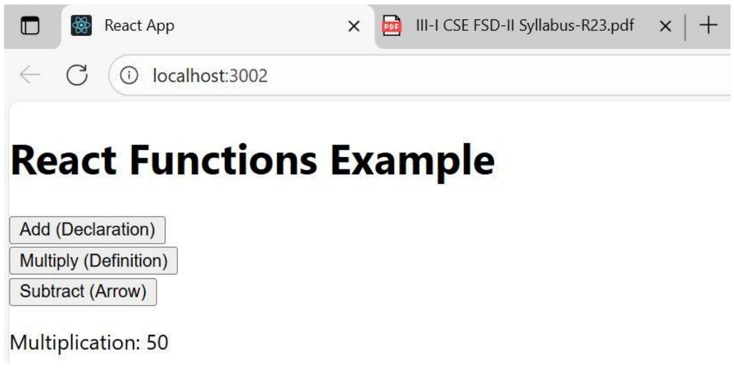
OUTPUT:





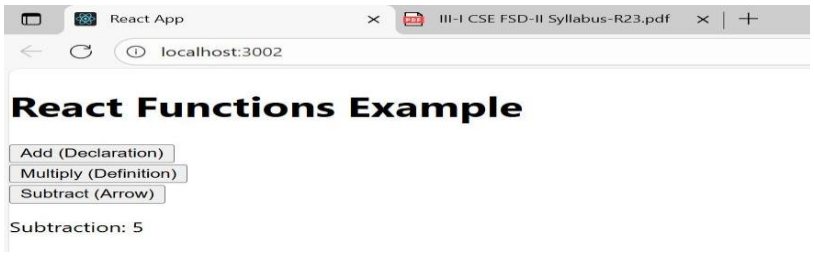
React App | III-I CSE FSD-II Syllabus-R23.pdf | localhost:3002

```
React Functions Example

Add (Declaration)
Multiply (Definition)
Subtract (Arrow)

Addition: 15
```



React App | III-I CSE FSD-II Syllabus-R23.pdf | localhost:3002

```
React Functions Example

Add (Declaration)
Multiply (Definition)
Subtract (Arrow)

Multiplication: 50
```



React App | III-I CSE FSD-II Syllabus-R23.pdf | localhost:3002

```
React Functions Example

Add (Declaration)
Multiply (Definition)
Subtract (Arrow)

Subtraction: 5
```

WEEK-2 Basics of React. Js

A) Program Statement: Write a React program to implement a counter button using react class components.

Description:

This program demonstrates how to build a Counter Application using React class components. It mainly focuses on state management and event handling in React.

A **class component** is:

- A JavaScript **class** that extends React.Component. Must have a render() method which returns JSX.
- Can maintain **state** (internal data that can change).

Key Parts of a Class Component

1. **Constructor (optional):**
   Initializes the state and binds methods.
2. **State:**
   Holds data that can change.
   Example: this.state = { count: 0 }.
3. **Methods:**
   Functions that update the state, usually using this.setState().
4. **Render method:**
   Returns JSX (UI representation).
5. **Binding:**

   binding methods" usually refers to **binding this context** to class methods so they behave correctly when used as event handlers or callbacks.

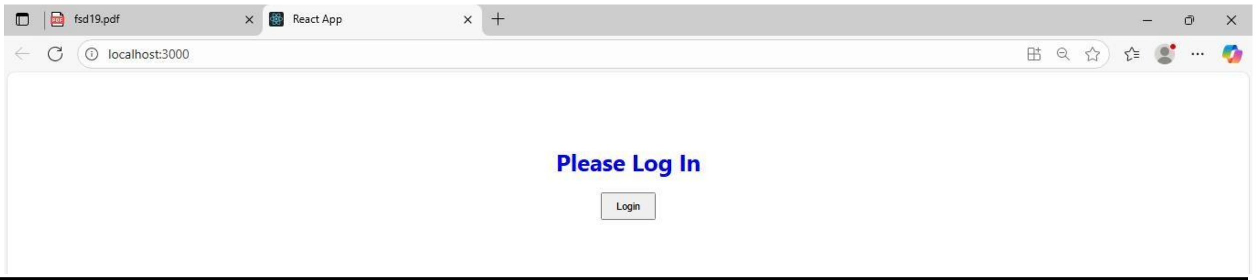
Source Code:

```
import React, { Component } from 'react'; import
'./App.css';
class App extends Component
{ constructor(props) {
super(props)this.state = {
count: 0, };
// Bind the handleClick method to this
this.handleClick = this.handleClick.bind(this);
 }
handleClick()
{ this.setState({ count: this.state.count + 1 });
}
render()
{ return
(
        <div className="App">
        <h1>Counter App</h1>
        <p>Count: {this.state.count}</p>
        <button onClick={this.handleClick}>Increment</button>
        </div>
        );
} } export default
App;
```

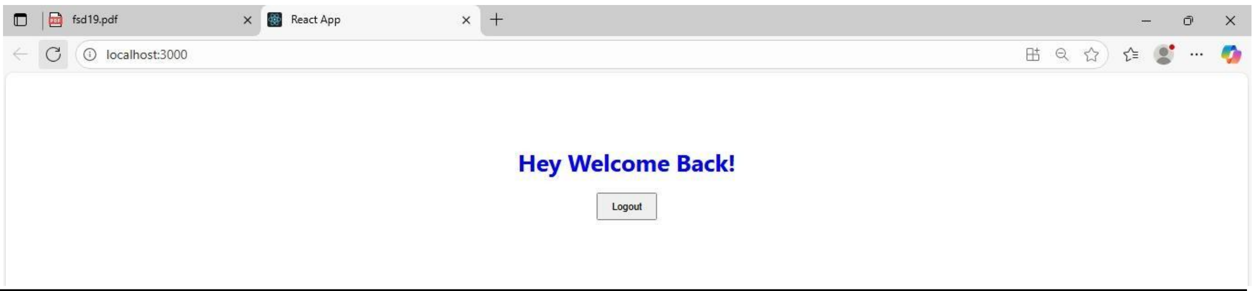
index.js:

```
import React from 'react'; import ReactDOM from 'react-dom/client';
import './index.css'; import App from './App'; const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
<React.StrictMode>
<App />
</React.StrictMode>
);
```

OUTPUT:



Please Log In



B) Program Statement: Write a React program to implement a counter button using react functional components.

Description: We are going to build a Counter App in React using functional components and the useState Hook. • It will display a number (the counter). • There will be one buttons:Increment ,Every time you click a button, the counter value will increase . • React will automatically re-render the component when the state changes.

Key Features of Functional Components

1. **Simple Syntax** → just a function that returns JSX.
2. **Hooks for State & Lifecycle** → useState, useEffect, etc.
3. **No this keyword** → state and functions are handled inside the function.
4. **Preferred in modern React** (lighter, easier to read, more powerful).

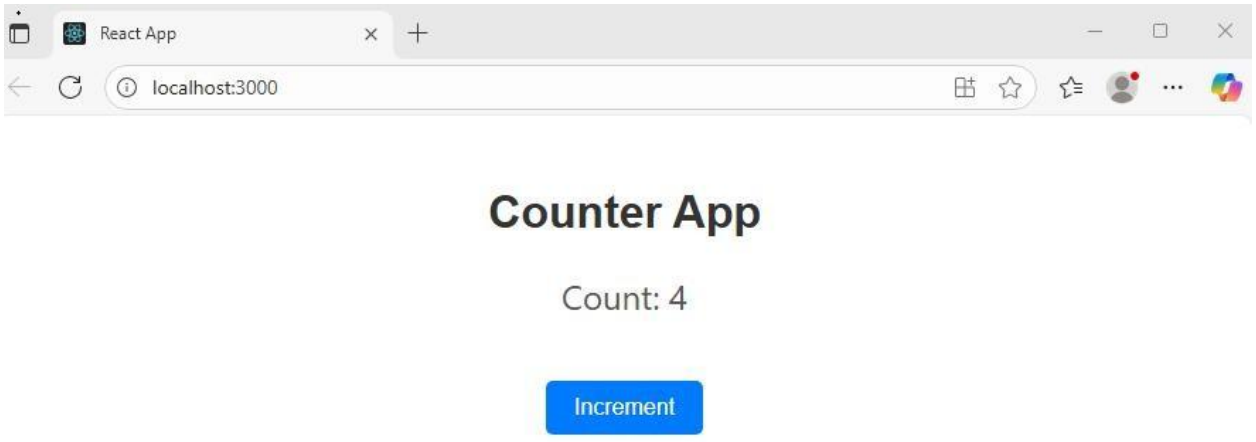
Source Code:

```
// src/App.js import React, { useState
} from 'react';
import './App.css'; function
App() {
// Declare a state variable 'count' and updater function 'setCount' const
[count, setCount] = useState(0);
// Function to handle increment
 const handleClick = () => {
 setCount(count + 1); };
return (
<div className="App">
<h1>Counter App</h1>
<p>Count: {count}</p>
<button onClick={handleClick}>Increment</button>
</div> );} export
default App;
```

index.js:

```
import React from 'react'; import ReactDOM from 'react-dom/client';
import './index.css'; import App from './App'; const root =
ReactDOM.createRoot(document.getElementById('root')); root.render(
<React.StrictMode>
<App />
</React.StrictMode>
);
.App {
text-align: center;
margin-top: 50px; }
h1 {
font-family: Arial, sans-serif;
color: #333; } p {
font-size: 24px; color:
#555; } button { padding:
10px 20px; margin-top:
20px; font-size: 16px;
background-color: #007bff;
color: white; border: none;
border-radius: 5px; cursor:
pointer;
transition: 0.3s ease; }
```

OUTPUT:



C)Program Statement: Write a React program to handle the button click events in functional component. Description:

In React, functional components can respond to user actions such as button clicks by using event handlers.

1. We define a function inside the component that contains the logic we want to run when a button is clicked.
2. We attach this function to a button using the onClick attribute.
3. If we want the UI to change when the button is clicked, we use the useState hook to update the state.
4. When the state changes, React re-renders the component automatically and updates the display.

Source Code:

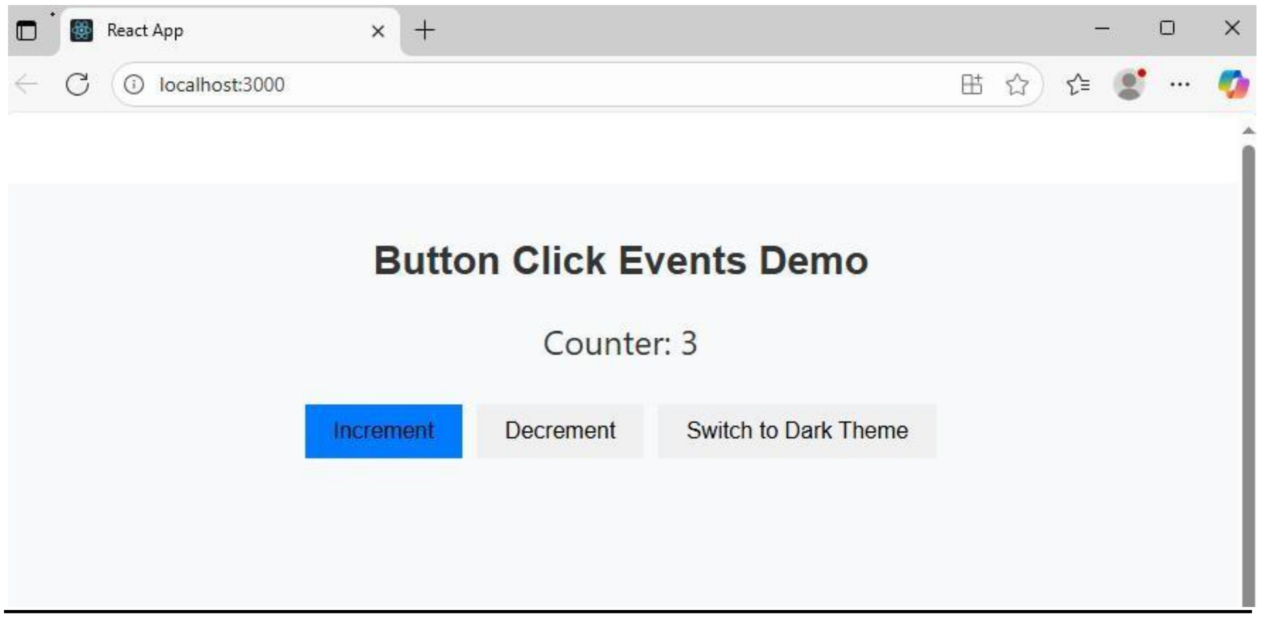
App.js
```
import React, { useState } from "react"; import
"./App.css"; function App() { const [count, setCount] =
useState(0); const [isDarkTheme, setIsDarkTheme] =
useState(false); const handleIncrement = () => {
setCount(count + 1);
};
const handleDecrement = () =>
{ if (count > 0) {
setCount(count - 1);
} } const handleToggleTheme = () => {
setIsDarkTheme(!
isDarkTheme);
};
return (
<div className={`App ${isDarkTheme ? "dark" : "light"}`}>
<h1>Button Click Events Demo</h1>
<p>Counter: {count}</p>
<button onClick={handleIncrement}>Increment</button>
<button onClick={handleDecrement}>Decrement</button>
<button onClick={handleToggleTheme}>
Switch to {isDarkTheme ? "Light" : "Dark"} Theme
</button>
</div>
);
} export default App; index.js import React from "react"; import
ReactDOM from "react-dom/client"; import "./index.css"; import App
from "./App"; const root =
ReactDOM.createRoot(document.getElementById("root")); root.render(
<React.StrictMode>
<App />
</React.StrictMode>
);
```

App.css

```
.App {
text-align: center; margin-
top: 50px; padding: 20px;
min-height: 100vh;
transition: all 0.3s ease; }
.App.light { background-
color: #f8f9fa; color: #333; }
.App.dark { background-
color: #333; color: #f8f9fa; }
h1 {
font-family: Arial, sans-serif;
font-size: 28px; } p {
```

```
font-size: 24px; }
button { padding:
10px 20px; font-
size: 16px; margin:
5px; border: none;
cursor: pointer; }
button:nth-child(1)
{ background-color:
#28a745;
} button:nth-
child(1):hover {
background-color:
#218838; }
button:nth-child(2)
{ background-color:
#dc3545; } button:nth-
child(2):hover {
background-color:
#c82333; }
button:nth-child(3)
{ background-color:
#007bff; } button:nth-
child(3):hover {
background-color:
#0056b3;
}
```

OUTPUT:



Button Click Events Demo

Counter: 3

D)Program Statement: Write a React program to conditionally render a component in the browser.

Description:

- **Conditional rendering** means **showing components or elements only when a certain condition is true**.
- It's like using **if/else** or **ternary operators** inside JSX to decide what to render.
- Commonly used when:
  ○ Showing login/logout buttons ○
  Displaying a loader while fetching data ○
  Rendering error messages

Methods of Conditional Rendering

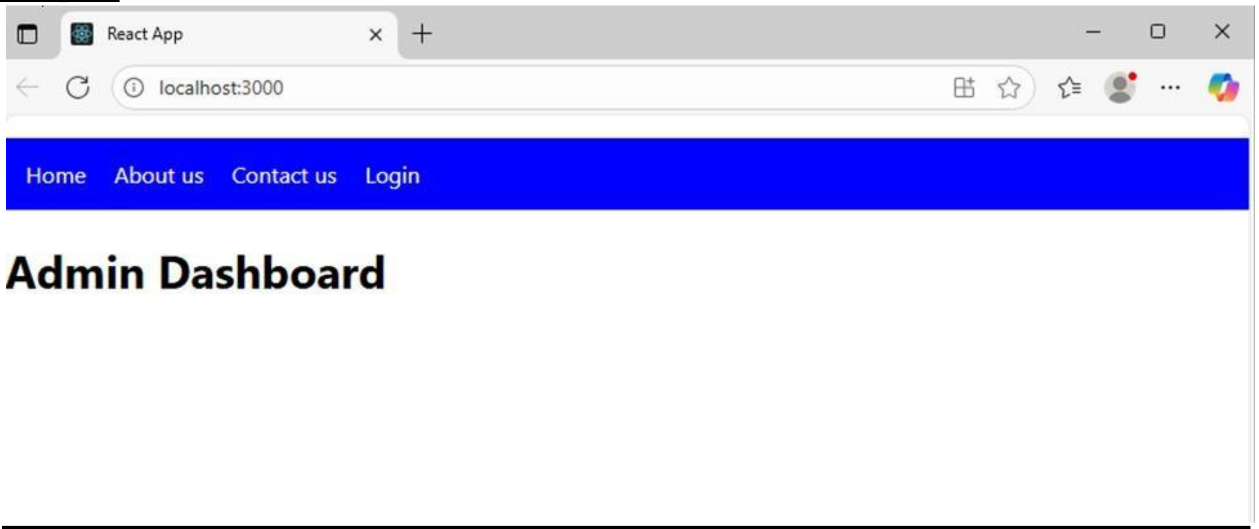
1. **if/else statements** → inside the component logic
2. **Ternary operator (? :)** → inline condition in JSX
3. **Logical AND (&&)** → render only if condition is true

Source Code:

App.js import React from

```
"react"; import
"./App.css"; function
App() { let loggedIn =
false; let isAdmin = true;
return (
<>
<Header loggedIn={loggedIn} />
{isAdmin ? <Admin /> : null}
</>
);
} function Header({ loggedIn
}) { return (
<ul>
<li>Home</li>
<li>About us</li>
<li>Contact us</li>
{loggedIn ? <li>Logout</li> : <li>Login</li>}
</ul> ); } function Admin() { return
<h1>Admin Dashboard</h1>}
export default App; App.css ul {
list-style-type: none;
display: flex; gap:
20px; background-
color: blue; padding:
15px; color: #fff;
}
```

Output:



Admin Dashboard

E) Program Statement: Write a React program to display text using String literals.

Description: This React program demonstrates how to use **JavaScript String Literals (Template Literals)** to display dynamic text in a web application.

1. **String Literals (Template Literals)** ○ Use backticks ` instead of quotes. ○ Allow **embedding variables and expressions** using ${variable}.
   - Support **multi-line strings** without needing \n.
2. **React Functional Component** ○ The program uses a functional component called App. ○ Inside the component, we define variables like name, course, and subject.
   - We create a **message** string using template literals that includes these variables.
3. **Displaying Text in React** ○ React JSX allows embedding JavaScript expressions inside {}. ○ We use <p>{message}</p> to display the dynamically created string.

Source Code:

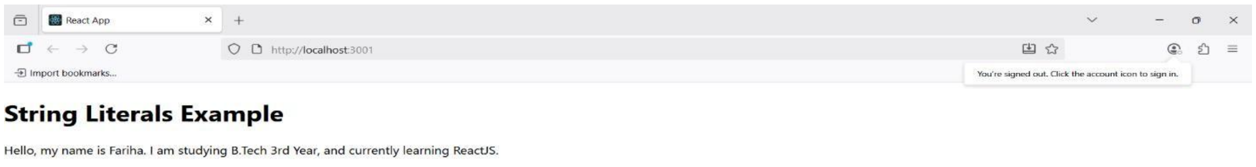
App.js

```
import React from "react";

export default function App() {
  const name = "Taheseen"; const
  course = "B.Tech 2nd Year"; const
  subject = "ReactJS";

  const message = `Hello, my name is ${name}.
I am studying ${course}, and currently learning ${subject}.`;

  return (
    <div className="p-6 text-center">
      <h1 className="text-2xl font-bold mb-4">String Literals Example</h1>
      <p className="text-lg whitespace-pre-line">{message}</p>
    </div>
  );
}
```

Index.js

```
import React from "react"; import ReactDOM from
"react-dom/client"; import "./index.css"; // optional if
using Tailwind or CSS import App from "./App";

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

OUTPUT:



String Literals Example

Hello, my name is Fariha. I am studying B.Tech 3rd Year, and currently learning ReactJS.

WEEK-3

A)Program Statement: Write a React program to implement a counter button using React use State hook.

Description: useState is a **Hook** in React that lets you add **state** to functional components. It returns an **array with two values**:

1. The current state value
2. A function to update that state
3. **Interactive Controls**:
    1. Increment button (increases count by 1)
    2. Decrement button (decreases count by 1)
    3. Reset button (sets count back to 0)
4. **Visual Feedback**:
    1. Color-coded counter value (green for positive, red for negative, blue for zero)
    2. Smooth transitions and hover effects on buttons
    3. Responsive design that works on both desktop and mobile devices

Source Code:

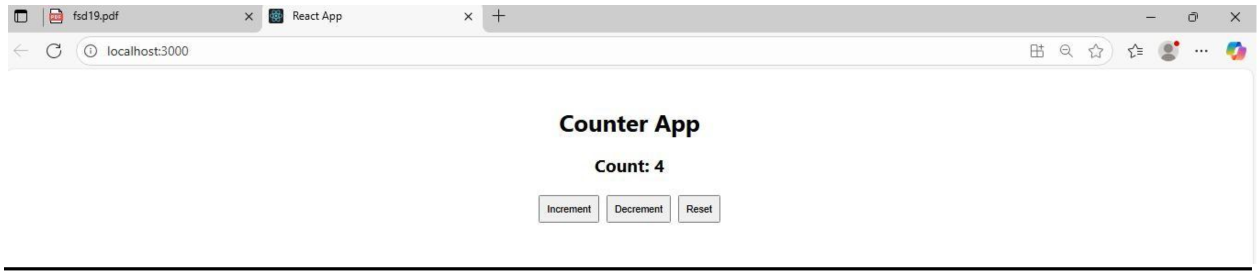
CountIncDec.js

```
import React, { useState } from 'react'; function CountIncDec()
{const [count, setCount] = useState(0);
 const increment = () => {
  setCount(count + 1);
 };
 const decrement = () => {
  setCount(count - 1);
 };
 const reset = () => {
  setCount(0);
 };
 return (
    <div style={{ textAlign: 'center', marginTop: '50px' }}>
      <h1>Counter App</h1>
      <h2>Count: {count}</h2>
      <button onClick={increment} style={{ margin: '5px', padding: '10px' }}>
        Increment
      </button>
      <button onClick={decrement} style={{ margin: '5px', padding: '10px' }}>
        Decrement
      </button>
      <button onClick={reset} style={{ margin: '5px', padding: '10px' }}>
        Reset
      </button>
    </div>
  );
}

export default CountIncDec
```

Index.js

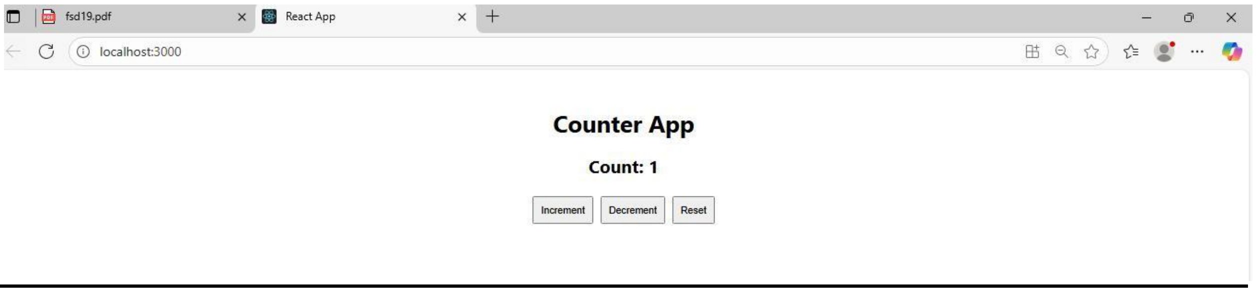
```
import React from 'react'; import
ReactDOM from 'react-dom/client'; import
'./index.css'; import CountIncDec from
'./CountIncDec';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <CountIncDec/>
  </React.StrictMode>
);
```

OUTPUT:



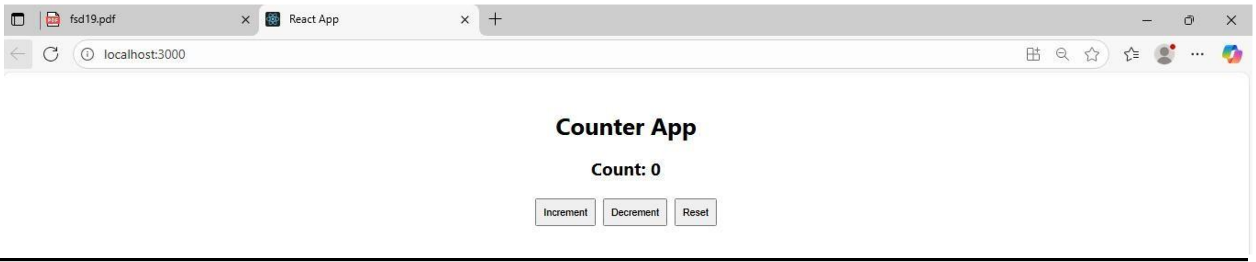
Counter App

Count: 4



Counter App

Count: 1



Counter App

Count: 0

B)Program Statement: Write a React program to fetch the data from an API using React use Effect hook.

Description: This program demonstrates **side effects** in React using the **useEffect hook**, which is used for **API calls, subscriptions, timers**, etc.

1. **useEffect Hook**:
   - Syntax: useEffect(() => { /* effect code */ }, [dependencies]); o Runs **after the component renders**.
   - If the dependency array is empty [], it runs **once after initial render**.
2. **Fetching Data**:
   - Uses fetch() API to get data asynchronously.
   - useState stores fetched data.
3. **Asynchronous Operations**:

- API calls return Promises → we handle with .then() or async/await

Source Code:

ApiData.js import React , {useState, useEffect}

```
from 'react'; const ApiData = ( ) => {
const [data, setData] = useState([ ]); useEffect( ( ) => {
fetch('https://jsonplaceholder.typicode.com/todos').then(response =>
response.json()).then(json => setData(json))}, []) return(
<div>
{data.map(item => <li key={item.id}> {item.title} </li>)}
</div>
);
}
export default ApiData;
```

Replace code in the return sec:

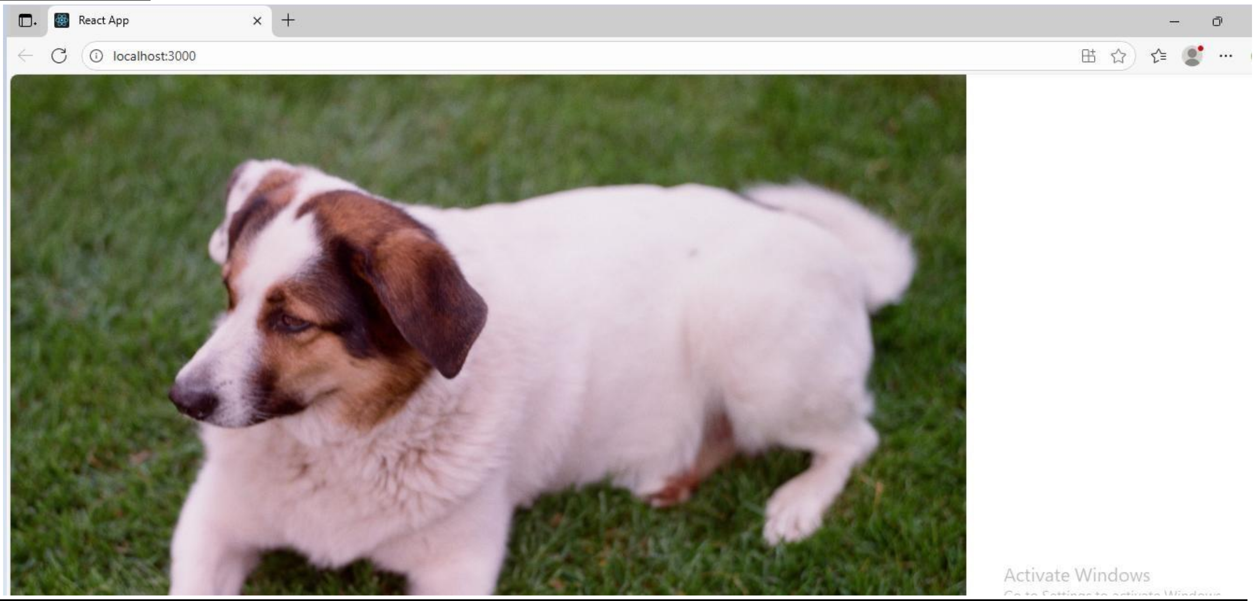
```
<ul>
{data.map(item => (
<li key={item.id}>
<strong>ID:</strong> {item.id},
<strong> User ID:</strong> {item.userId},
<strong> Title:</strong> {item.title},
<strong> Completed:</strong> {item.completed ? 'Yes' : 'No'}
</li>
))}
</ul>
```

DogsApp.js

```
//Fetch Data from API using UseEffect Hook
import React , {useState, useEffect} from 'react';
function DogsApp()
```

```
{ let [ dogImage, setDogImage] = useState(null) useEffect(() => {
fetch("http://dog.ceo/api/breeds/image/random").then(response=>response.json())
.then( data => setDogImage(data.message))
}, [])
return (
<div>
{dogImage && <img src={dogImage} alt="Randpm Images of Dogs Breeds" />}
</div>
);
} export default
DogsApp;
```

OUTPUT:



C)Program Statement: Write a React program with two react components sharing data using Props.

Description: This program demonstrates **component communication** in React using **props**. Props are used to **pass data from a parent component to a child component**.

1. **Props (Properties)**:

- Read-only data passed to child components. o Syntax: <ChildComponent data={value} /> o Access inside child: props.data or via destructuring {data}

2. **Component Composition**:
    - React apps are built from **smaller reusable components**.
    - Parent → passes data → Child → renders data

Source Code:

AppPassData.js import React from 'react' import

SecondComp from './PassingData/SecondComp' import

FirstComp from './PassingData/FirstComp' import

ThirdComp from './PassingData/ThirdComp' function

AppPassData() { return (

<div>

<FirstComp name = "CSE" />

<SecondComp name = "Data Science" />

<ThirdComp name = "Artificial Intelligence" />
</div>) } export default

AppPassData; **FirstComp.js**

import React from 'react' const

FirstComp = (props) => {

return (

<div>)

{name} </div> } export default

FirstComp **SecondComp.js**

import React from 'react' const

SecondComp = (props) => {

return (

<div>) {name} </div> }

export default SecondComp

ThirdComp.js import React

from 'react' const ThirdComp

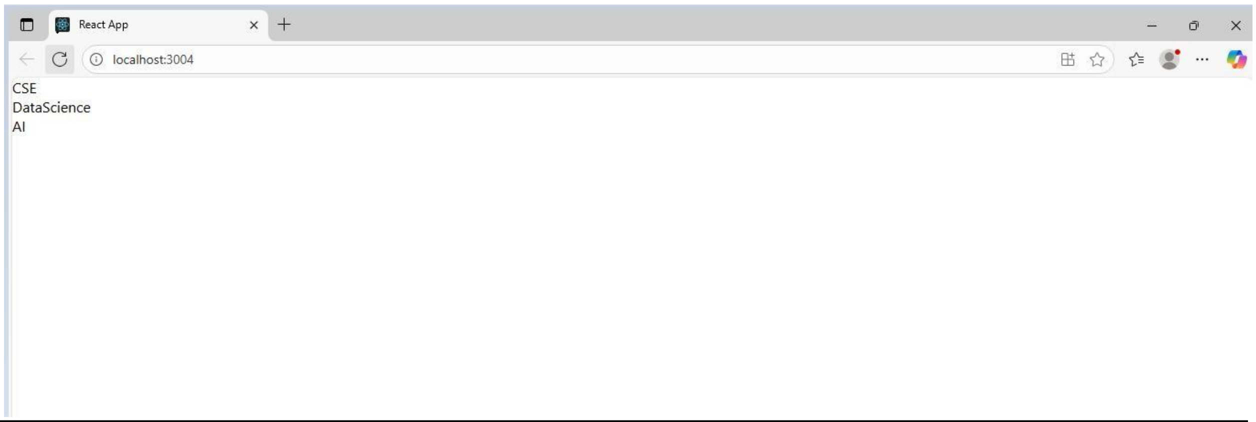
= (props) =>

{ return (

<div>) }

{name} </div>
export default ThirdComp

OUTPUT:



```
CSE
DataScience
AI
```

D)Program Statement: Write a React program to implement the forms in react

Description: This program demonstrates **forms in React** and **controlled components**. Controlled components have their **input values bound to state**, so React always controls the form data.

1. **Controlled Components**:
   - Input value is tied to state: value={stateVariable} o Update state on onChange event.
2. **Form Handling**:
   - Prevent default HTML form submission using e.preventDefault().
   - Handle input dynamically via useState.
3. **Event Handling**:
   - onChange → update state o onSubmit → process form data **Source**

   Code:

SimpleForm.js

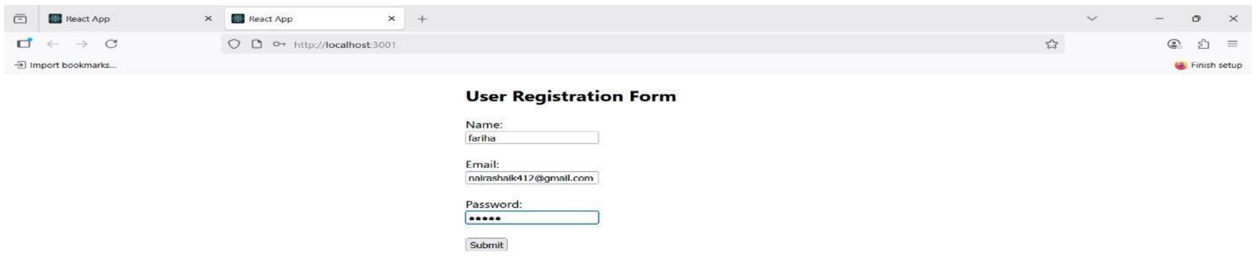
```
import React,{useState} from 'react';

function SimpleForm(){

const[formData,setFormData] = useState(

    { name: "" , email:"", password:" " });

  const[submittedData,setSubmittedData] =

  useState(null); function handleChange(e){ const {

  name,value } = e.target; setFormData(prev =>({

      ...prev,

      [name]:value

    })); }
  function handleSubmit(e){
    e.preventDefault(); if(!formData.name || !formData.email ||

    !formData.password)

    {

       alert("please fill in all fields"); return; }

    setSubmittedData(formData);

    setFormData({name:"",email:"",password:""})

    ;

 }

 return(

    <div style={{maxWidth:400}}>

       <h2>User Registration Form</h2>

       <form onSubmit={handleSubmit}>

          <label>                Name: <br></br>

             <input type="text" name="name" value={formData.name}
onChange={handleChange} required/>

          </label><br></br>

          <label> Email:                <br></br>

             <input type="email" name="email" value={formData.email}
onChange={handleChange} required/>

          </label>            <br></br>
          <label> Password: <br></br>

             <input type="password" name="password" value={formData.password}
onChange={handleChange} required/>
```

```
          </label>              <br></br>
          <label> <br></br>
             <button type="submit">Submit</button></label> </form>
        {submittedData&&(
          <div style={{marginTop:20}}>
             <h3>Form submitted Data</h3>
             <p>{JSON.stringify(submittedData,null,2)}</p>
      </div> )}
   </div> );} export default SimpleForm;
```

index.js import React from 'react'; import ReactDOM from 'react-dom/client'; import './index.css'; import reportWebVitals from './reportWebVitals'; import SimpleForm from './SimpleForm'; const root = ReactDOM.createRoot(document.getElementById('root'));

```
root.render(<React.StrictMode>
   <SimpleForm/>
  </React.StrictMode>);
```

OUTPUT:

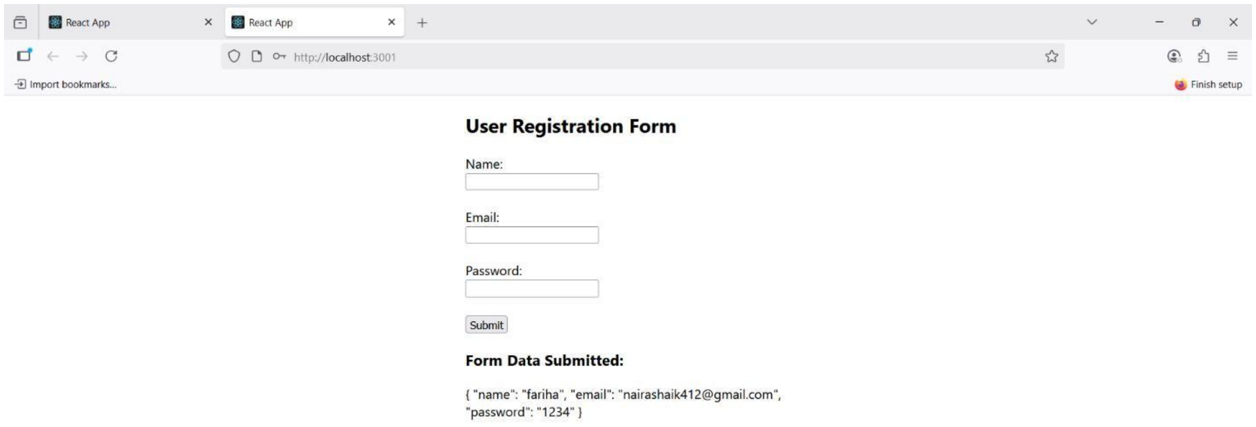


User Registration Form

Name:
fariha

Email:
nairashaik412@gmail.com

Password:
•••••



User Registration Form

Name:

Email:

Password:

Form Data Submitted:

```
{ "name": "fariha", "email": "nairashaik412@gmail.com",
"password": "1234" }
```

<u>E) Program Statement:</u> Write a React program to implement the iterative rendering using map() function.

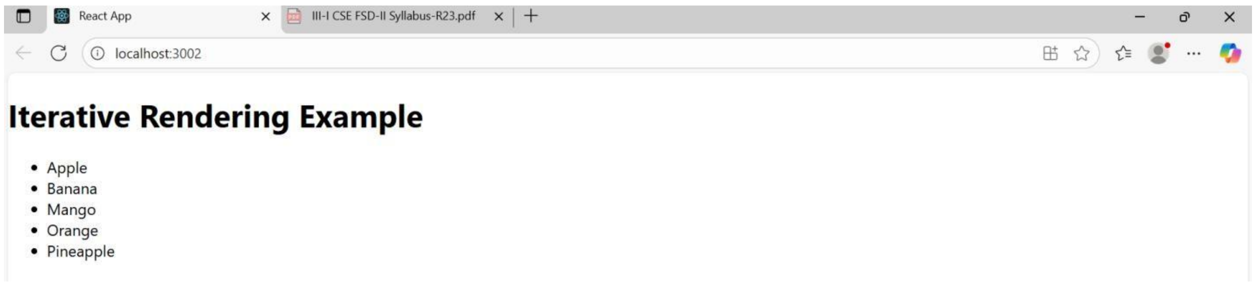
<u>Description:</u> This program demonstrates **iterative rendering** in React using **JavaScript's map() function**. map() allows you to **render multiple elements dynamically** based on an array of data.

1. **Iterative Rendering:**
   - Use map() to loop over arrays.
   - Each child must have a **unique key** prop.
2. **Dynamic Display:**
   - Works with data fetched from API or static arrays. o Allows rendering lists efficiently.

Source Code:

```
import React from "react"; export default function ListRendering() {

const fruits = ["Apple", "Banana", "Mango", "Orange", "Pineapple"];

return (

  <div className="p-6 text-center">

    <h1 className="text-xl font-bold mb-4">Iterative Rendering Example</h1>

    <ul>

      {fruits.map((fruit, index) => (

        <li key={index} className="mb-1">

          {fruit}

        </li>

      ))}

    </ul>
  </div>);} Index.js import React from "react"; import

ReactDOM from "react-dom/client"; import

"./index.css"; // optional if using Tailwind or CSS import

ListRendering from "./ListRendering";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <ListRendering/>
  </React.StrictMode>
);
```

OUTPUT:



Iterative Rendering Example

- Apple
- Banana
- Mango
- Orange
- Pineapple