Full Stack Development – II

Experiment 1(a): Linking an External JavaScript File with an HTML Page

---

Aim

To write a JavaScript program to link an external JavaScript file with an HTML page.

Introduction

JavaScript is a scripting language used to make web pages interactive. It is one of the three core technologies of web development along with HTML and CSS. HTML provides the structure of a webpage, CSS is used for styling, and JavaScript adds interactivity**. JavaScript code can be written in a separate file and linked to an HTML page using the `<script>` tag.**

Theory

An external JavaScript file has the extension **`.js`** and is linked to an HTML page using the `<script>` tag.

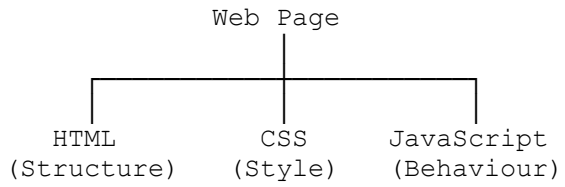
Syntax

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

Using an external JavaScript file offers the following advantages:

- Improves code readability.
- Promotes code reusability.
- Simplifies maintenance.
- Separates webpage structure from functionality.

HTML, CSS and JavaScript Relationship



Algorithm

Step 1: Create an HTML file named **index.html**.

Step 2: Create a JavaScript file named **script.js**.

Step 3: Link the JavaScript file using the `<script>` tag.

Step 4: Save both files in the same folder.

Step 5: Execute **index.html** in a web browser.

Program

index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>JavaScript Demo</title>
</head>

<body>

<h2>Welcome to Full Stack Development-II</h2>

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

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

---

script.js

```
alert("JavaScript File Linked Successfully");
```

Code Explanation

HTML

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

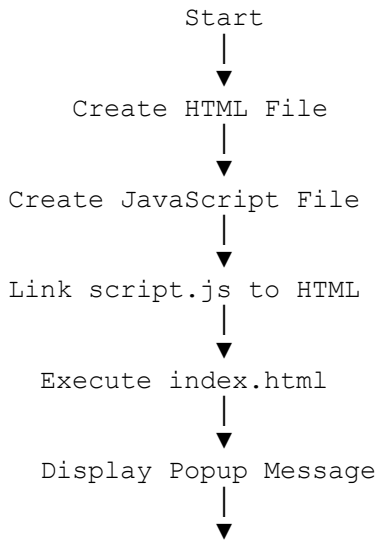
This statement links the external JavaScript file (`script.js`) with the HTML page.

JavaScript

```
alert("JavaScript File Linked Successfully");
```

The `alert()` function displays a popup dialog box containing the specified message.

Flowchart



```
End
```

Output

Webpage

```
Welcome to Full Stack Development-II
```

Popup Message

```
---------------------------------------
JavaScript File Linked Successfully
                 OK
---------------------------------------
```

Result

Thus, the JavaScript program to link an external JavaScript file with an HTML page was implemented and executed successfully.

Viva Questions

1. What is JavaScript?
JavaScript is a scripting language used to make web pages interactive.

2. Which HTML tag is used to include JavaScript?
The `<script>` tag.

3. What is the purpose of the `src` attribute?
It specifies the location of the external JavaScript file.

4. Why do we use an external JavaScript file?
To improve code readability, reusability, and maintenance.

5. Which file is executed to run the program?
`index.html` is executed, and it automatically loads `script.js`.

Experiment 1(b): Selecting HTML Elements Using DOM Selectors

Aim

To write a JavaScript program to select HTML elements using DOM selectors.

Introduction

The **Document Object Model (DOM)** is a programming interface for HTML documents. It represents the webpage as a tree-like structure, allowing JavaScript to access and modify HTML elements dynamically. DOM selectors are methods used to locate specific elements on a webpage based on their **id**, **class**, or **tag name**.

Theory

JavaScript provides several DOM selector methods to access HTML elements.

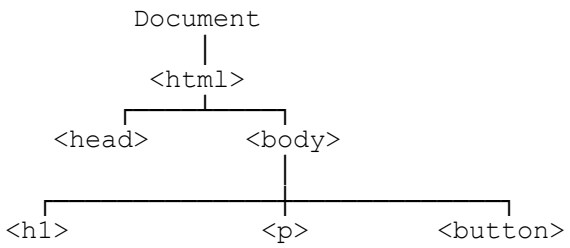
Some commonly used selectors are:

- **`getElementById()`** – Selects an element using its unique **id**.
- **`querySelector()`** – Selects the first element that matches a CSS selector.
- **`querySelectorAll()`** – Selects all elements that match a CSS selector.

DOM Selector Overview

Selector	Purpose	Example
`getElementById()`	Selects an element by ID	`document.getElementById("heading")`
`querySelector()`	Selects the first matching element	`document.querySelector(".message")`
`querySelectorAll()`	Selects all matching elements	`document.querySelectorAll("p")`

DOM Structure



JavaScript uses DOM selectors to access these HTML elements.

Algorithm

Step 1: Create an HTML file containing a heading, paragraph, and button.

Step 2: Assign appropriate **id** and **class** attributes.

Step 3: Create a JavaScript file.

Step 4: Select the HTML elements using DOM selector methods.

Step 5: Display the selected elements using `console.log()`.

Step 6: Execute the HTML file in a web browser.

Program

index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>DOM Selectors</title>
</head>

<body>

    <h1 id="heading">Welcome to JavaScript</h1>
```

```
    <p class="message">Learning DOM Selectors</p>

    <button>Click Me</button>

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

</body>

</html>
```

---

script.js

```
// Select element using ID
let heading = document.getElementById("heading");

// Select element using Class
let paragraph = document.querySelector(".message");

// Select element using Tag Name
let button = document.querySelector("button");

// Display selected elements
console.log(heading);
console.log(paragraph);
console.log(button);
```

Code Explanation

1. Selecting an Element by ID

```
let heading = document.getElementById("heading");
```

- `document` represents the current HTML page.
- `getElementById("heading")` selects the element whose **id** is `"heading"`.

---

2. Selecting an Element by Class

```
let paragraph = document.querySelector(".message");
```

- `querySelector()` selects the **first** matching element.
- A dot (`.`) indicates a **class selector**.

---

3. Selecting an Element by Tag Name

```
let button = document.querySelector("button");
```

- Selects the first `<button>` element in the HTML page.

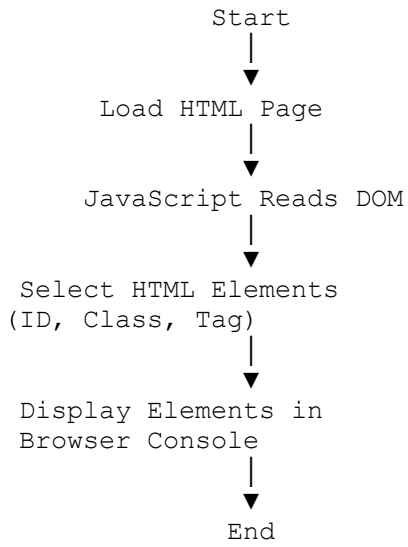
---

4. Displaying the Selected Elements

```
console.log(heading);
console.log(paragraph);
console.log(button);
```

The `console.log()` method displays the selected HTML elements in the browser's Developer Console.

Flowchart



Output

Webpage

```
Welcome to JavaScript

Learning DOM Selectors

[ Click Me ]
```

Browser Console (Press F12 → Console)

```
<h1 id="heading">Welcome to JavaScript</h1>

<p class="message">Learning DOM Selectors</p>

<button>Click Me</button>
```

Result

Thus, the JavaScript program to select HTML elements using DOM selectors was implemented and executed successfully.

Viva Questions

1. What is the DOM?
The Document Object Model (DOM) is a tree-like representation of an HTML document that allows JavaScript to access and modify webpage elements.

---

2. What is a DOM selector?
A DOM selector is a JavaScript method used to locate HTML elements in a webpage.

---

3. What is the difference between `getElementById()` and `querySelector()`?

- `getElementById()` selects an element using its unique **id**.
- `querySelector()` selects the first element matching a CSS selector such as an ID, class, or tag.

---

4. What is the purpose of `console.log()`?
It displays information in the browser's Developer Console for debugging and testing.

---

5. What is the difference between `querySelector()` and `querySelectorAll()`?

- `querySelector()` returns the **first** matching element.

- `querySelectorAll()` returns **all** matching elements as a collection.

Note

- Save both index.html and script.js in the same folder.
- Always execute index.html.
- Press F12 and open the Console tab to view the output of console.log().
- Ensure that the id and class names in the JavaScript code exactly match those in the HTML file (JavaScript is case-sensitive).

Experiment 1(c): Implementing Event Listeners

Aim

To write a JavaScript program to implement an event listener using the `addEventListener()` method.

Introduction

JavaScript allows web pages to respond to user actions such as clicking a button, pressing a key, or moving the mouse. These actions are called **events**. An **event listener** waits for a specific event to occur and then executes the associated JavaScript function. Event listeners are widely used in modern web applications to create interactive web pages.

Theory

An **event** is an action or occurrence detected by the .browser

Examples of events include:

- Mouse click
- Double click
- Mouse over
- Form submission

The **`addEventListener()`** method is used to attach an event handler to an HTML element.

Syntax

```
element.addEventListener("event", function () {
    // Statements to execute
});
```

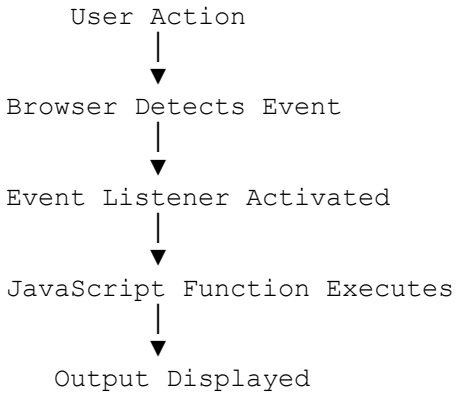
Where:

- **element** → HTML element on which the event occurs.
- **"event"** → Name of the event (e.g., `click`).
- **function()** → Function executed when the event occurs.

Common JavaScript Events

Event	Description
`click`	Triggered when an element is clicked
`dblclick`	Triggered when an element is double-clicked
`mouseover`	Triggered when the mouse pointer moves over an element
`mouseout`	Triggered when the mouse pointer leaves an element
`keydown`	Triggered when a keyboard key is pressed

Event Handling Process



Algorithm

Step 1: Create an HTML page with a button.

Step 2: Assign an **id** to the button.

Step 3: Create an external JavaScript file.

Step 4: Select the button using `getElementById()`.

Step 5: Attach a **click** event listener using `addEventListener()`.

Step 6: Display an alert message when the button is clicked.

Step 7: Execute the HTML page in a web browser.

Program

index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Event Listener Example</title>
</head>

<body>

    <h2>Event Listener Demo</h2>

    <button id="btn">Click Me</button>

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

</body>

</html>
```

---

script.js

```
// Select the button
let button = document.getElementById("btn");
```

```
// Add an event listener
button.addEventListener("click", function () {

    alert("Button Clicked Successfully!");

});
```

Code Explanation

Selecting the Button

```
let button = document.getElementById("btn");
```

This statement selects the HTML button whose **id** is `"btn"` and stores it in the variable `button`.

Adding an Event Listener

```
button.addEventListener("click", function () {
```

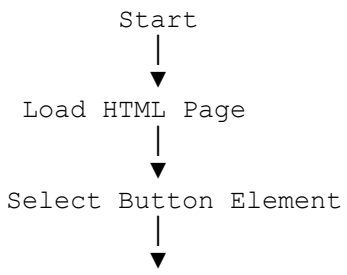
- `addEventListener()` attaches an event to the selected button.
- `"click"` specifies that the event occurs when the button is clicked.
- `function()` contains the code to be executed.

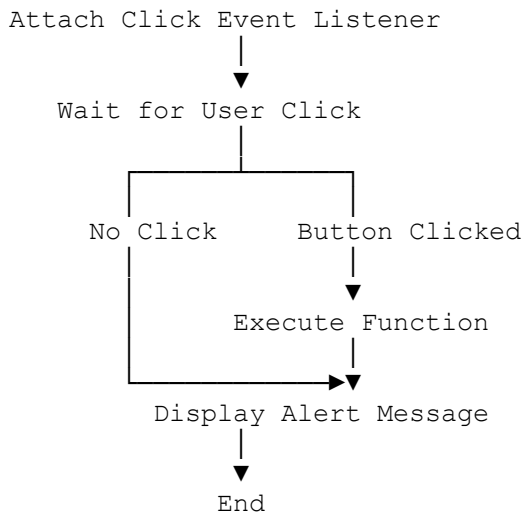
Displaying the Alert

```
alert("Button Clicked Successfully!");
```

The `alert()` function displays a popup message when the button is clicked.

Flowchart





Output

Webpage

```
Event Listener Demo

[ Click Me ]
```

After Clicking the Button

```
---------------------------------------
Button Clicked Successfully!
                OK
---------------------------------------
```

Result

Thus, the JavaScript program to implement an event listener using the `addEventListener()` method was implemented and executed successfully.

Viva Questions

1. What is an event?

An event is an action performed by the user or browser, such as clicking a button or pressing a key.

---

2. What is an event listener?

An event listener is a JavaScript method that waits for a specific event and executes a function when the event occurs.

---

3. What is the syntax of `addEventListener()`?

```
element.addEventListener("event", function () {
    // Code
});
```

---

4. Name any four JavaScript events.

- `click`
- `dblclick`
- `mouseover`
- `keydown`

---

5. What is the advantage of using `addEventListener()`?

It allows multiple event handlers to be attached to the same HTML element and provides greater flexibility than the `onclick` property.

Note

- `addEventListener()` is the preferred method for handling events in modern JavaScript because it is flexible and supports multiple event handlers on the same element.
- The event name (`click`) is case-sensitive. Ensure that the button `id` in HTML matches the `id` used in JavaScript.

Experiment 1(d)-Handling Click Events for HTML Button Elements

---

Aim

To write a JavaScript program to handle the click event of an HTML button using the `onclick` event.

Introduction

Buttons are one of the most commonly used elements in a webpage. When a user clicks a button, JavaScript can perform different actions such as displaying a message, changing text, changing the background color, or performing calculations.

The **`onclick`** event is used to execute JavaScript code whenever a button is clicked.

Theory

The **`onclick`** event is a JavaScript event that is triggered when a user clicks an HTML element.

The `onclick` property is assigned a function, and the statements inside the function are executed when the button is clicked.

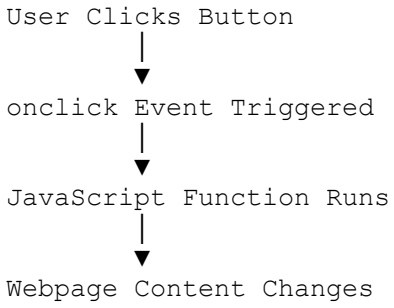
Syntax

```
element.onclick = function () {
    // Code to execute
};
```

Explanation

- **element** → The HTML element (button).
- **onclick** → Click event.
- **function()** → Code that runs when the button is clicked.

Working of onclick Event



Algorithm

Step 1: Create an HTML page containing a paragraph and a button.

Step 2: Assign an **id** to both the paragraph and the button.

Step 3: Create an external JavaScript file.

Step 4: Select the button using `getElementById()`.

Step 5: Handle the click event using the `onclick` property.

Step 6: Change the paragraph text using `innerHTML`.

Step 7: Execute the program in a web browser.

Program

index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Button Click Event</title>
</head>

<body>

<h2>Button Click Event Demo</h2>

<p id="message">Welcome Students</p>
```

```
<button id="btn">Click Here</button>

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

</body>

</html>
```

---

script.js

```
// Select the button
let button = document.getElementById("btn");

// Handle button click
button.onclick = function () {

    document.getElementById("message").innerHTML =
    "Button Clicked Successfully!";

};
```

Code Explanation

Line 1

```
let button = document.getElementById("btn");
```

This line selects the button whose **id** is `"btn"` and stores it in the variable **button**.

Line 2

```
button.onclick = function () {
```

This line tells JavaScript:

"When the button is clicked, execute the following function."

Line 3

```
document.getElementById("message").innerHTML =
"Button Clicked Successfully!";
```

This line performs two tasks:

- `getElementById("message")` selects the paragraph.
- `innerHTML` changes the text inside the paragraph.

Before clicking:

```
Welcome Students
```

After clicking:

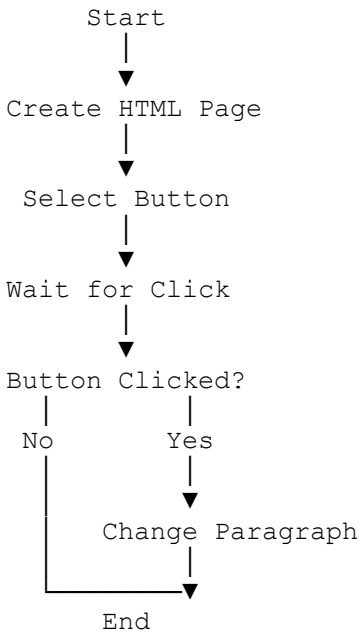
```
Button Clicked Successfully!
```

Line 4

```
};
```

Ends the function assigned to the `onclick` event.

Flowchart



Output

Before Clicking

```
Button Click Event Demo

Welcome Students

[ Click Here ]
```

After Clicking

```
Button Click Event Demo

Button Clicked Successfully!

[ Click Here ]
```

Result

Thus, the JavaScript program to handle the click event of an HTML button using the `onclick` property was implemented and executed successfully.

Viva Questions

1. What is the `onclick` event?

The `onclick` event is triggered when a user clicks an HTML element.

---

2. What is `innerHTML`?

`innerHTML` is used to get or change the content inside an HTML element.

---

3. Which method is used to select the button?

`getElementById()`

---

4. What happens when the button is clicked in this program?

The paragraph text changes from **"Welcome Students"** to **"Button Clicked Successfully!"**

---

5. What is the difference between `onclick` and `addEventListener()`?

- `onclick` allows only one click event for an element.
- `addEventListener()` allows multiple event handlers and is more flexible.

Note

- `innerHTML` is commonly used to change the text or HTML content of an element dynamically.

Experiment 1(e)

Types of JavaScript Functions

Aim

To write a JavaScript program demonstrating the three types of functions in JavaScript:

1. Function Declaration
2. Function Expression (Function Definition)
3. Arrow Function

Introduction

A **function** is a reusable block of code that performs a specific task. Instead of writing the same code multiple times, we can write it once inside a function and call it whenever needed. JavaScript provides different ways to create functions, making programs easier to read, maintain, and reuse.

Theory

JavaScript supports different types of functions. The most commonly used are:

1. Function Declaration

A function is created using the `function` keyword followed by a function name.

Syntax

```
function functionName() {
    // statements
}
```

---

2. Function Expression

A function is stored inside a variable.

Syntax

```
let functionName = function () {
    // statements
};
```

---

3. Arrow Function

An arrow function is a shorter and modern way of writing functions using the `=>` operator.

Syntax

```
let functionName = () => {
    // statements
};
```

Comparison of Function Types

Function Declaration	Function Expression	Arrow Function
Uses `function` keyword with a name	Function stored in a variable	Uses the `=>` operator
Traditional method	Variable-based method	Modern and concise syntax

---

Algorithm

Step 1: Create an HTML file.

Step 2: Create an external JavaScript file.

Step 3: Write a Function Declaration and call it.

Step 4: Write a Function Expression and call it.

Step 5: Write an Arrow Function and call it.

Step 6: Execute the HTML page.

Program

index.html

```
<!DOCTYPE html>
<html>

<head>
    <title>Types of Functions</title>
</head>

<body>

<h2>JavaScript Functions</h2>

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

</body>

</html>
```

---

script.js

```
// Function Declaration
function greetDeclaration() {
    alert("Hello from Function Declaration");
}

greetDeclaration();


// Function Expression
let greetExpression = function () {
    alert("Hello from Function Expression");
};

greetExpression();
```

```
// Arrow Function
let greetArrow = () => {
    alert("Hello from Arrow Function");
};

greetArrow();
```

Code Explanation

Function Declaration `()`

```
function greetDeclaration {

    alert("Hello from Function Declaration");
}

greetDeclaration();
```

- `function` is the keyword used to create a function.
- `greetDeclaration()` is the function name.
- `alert()` displays a popup message.
- `greetDeclaration();` calls the function.

Function Expression

```
let greetExpression = function () {
    alert("Hello from Function Expression");
};

greetExpression();
```

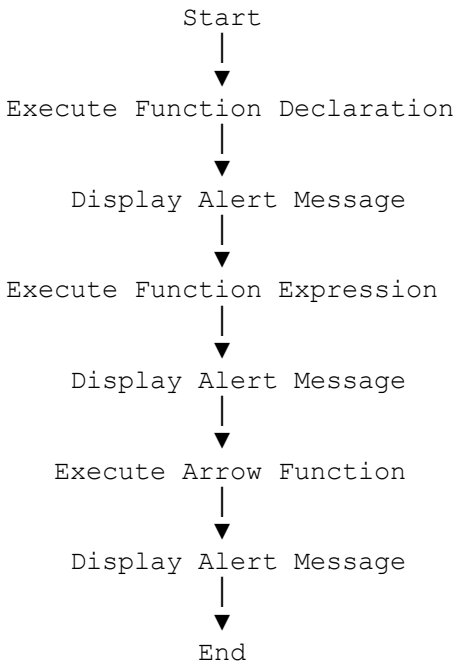
- `let` creates a variable.
- The function is stored inside the variable `greetExpression`.
- `greetExpression();` executes the function.

Arrow Function

```
let greetArrow = () => {
    alert("Hello from Arrow Function");
};

greetArrow();
```

- `=>` is called the **arrow operator**.
- Arrow functions provide a shorter syntax for writing functions.
- `greetArrow();` calls the arrow function.

Flowchart



Output

Three popup messages are displayed one after another.

Popup 1

```
Hello from Function Declaration
```

↓

Popup 2

```
Hello from Function Expression
```

↓

Popup 3

```
Hello from Arrow Function
```

Result

Thus, the JavaScript program demonstrating Function Declaration, Function Expression, and Arrow Function was implemented and executed successfully.

Viva Questions

1. What is a function?

A function is a reusable block of code that performs a specific task.

---

2. What are the three types of functions in JavaScript?

- Function Declaration
- Function Expression
- Arrow Function

---

3. What is the difference between a Function Declaration and a Function Expression?

A Function Declaration is created using the `function` keyword with a function name, whereas a Function Expression stores a function inside a variable.

---

4. What is an Arrow Function?

An Arrow Function is a shorter way of writing a function using the `=>` operator.

---

5. Why are functions used in programming?

Functions reduce code repetition, improve readability, and make programs easier to maintain.

Note

- A **Function Declaration** can be called before its definition due to **hoisting**, but **Function Expressions** and **Arrow Functions** should be defined before they are called.

Experiment 2

Basics of React.js

Aim

To understand the fundamentals of React.js and develop simple React applications using Class Components, Functional Components, Event Handling, Conditional Rendering, and String Literals.

Introduction and Theory

What is React?

React.js is an open-source JavaScript library developed by **Meta (Facebook)** for building fast, interactive, and reusable user interfaces (UI).

Unlike traditional web applications, React updates only the required part of a webpage instead of reloading the entire page. This makes applications faster and provides a better user experience.

Why React?

In HTML and JavaScript applications, as the project grows, the code becomes lengthy and difficult to maintain. React solves this problem by dividing the webpage into small reusable parts called **components**.

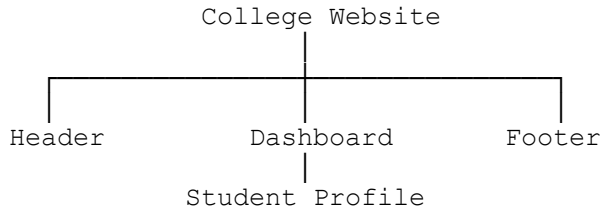
Advantages of React

- Reusable Components
- Faster Updates using Virtual DOM
- Easy to Maintain
- Better Performance
- Widely Used in Industry

What is a Component?

A **component** is an independent and reusable piece of code that represents a part of the user interface.

For example, a college website can be divided into the following components:



Each section can be created as a separate React component and reused wherever required.

Types of Components

React provides two types of components.

Class Component	Functional Component
Created using ES6 Classes	Created using JavaScript Functions
Uses `this.state`	Uses Hooks (`useState`)
Older approach	Modern approach

Note: In this experiment, you will learn both Class Components and Functional Components.

What is JSX?

JSX (JavaScript XML) is a syntax extension that allows HTML-like code to be written inside JavaScript.

Example:

```
<h1>Welcome to React</h1>
```

Although it looks like HTML, it is actually JSX.

Why JSX?

Without JSX, creating elements in React becomes difficult to read.

Using JSX makes the code:

- Easier to write
- Easier to read
- Easier to maintain

React Project Structure

After creating a React project using **Vite**, the project structure appears as follows:

```
student-app
│
├── node_modules
├── public
├── src
│   ├── App.jsx
│   ├── main.jsx
│   └── index.css
├── package.json
└── vite.config.js
```

Important Files

File	Purpose
`App.jsx`	Main component where React programs are written
`main.jsx`	Starts the React application
`package.json`	Stores project information and required packages
`node_modules`	Contains installed React packages

Creating a React Project

Step 1

Open **Terminal** or **Command Prompt**.

Step 2

Create a React project using Vite.

```
npm create vite@5
```

Step 3

Enter the project name.

Example:

```
student-app
```

Step 4

Choose:

```
React
```

Step 5

Choose:

```
JavaScript
```

Step 6

Move into the project folder.

```
cd student-app
```

Step 7

Install the required packages.

```
npm install
```

Step 8

Run the React application.

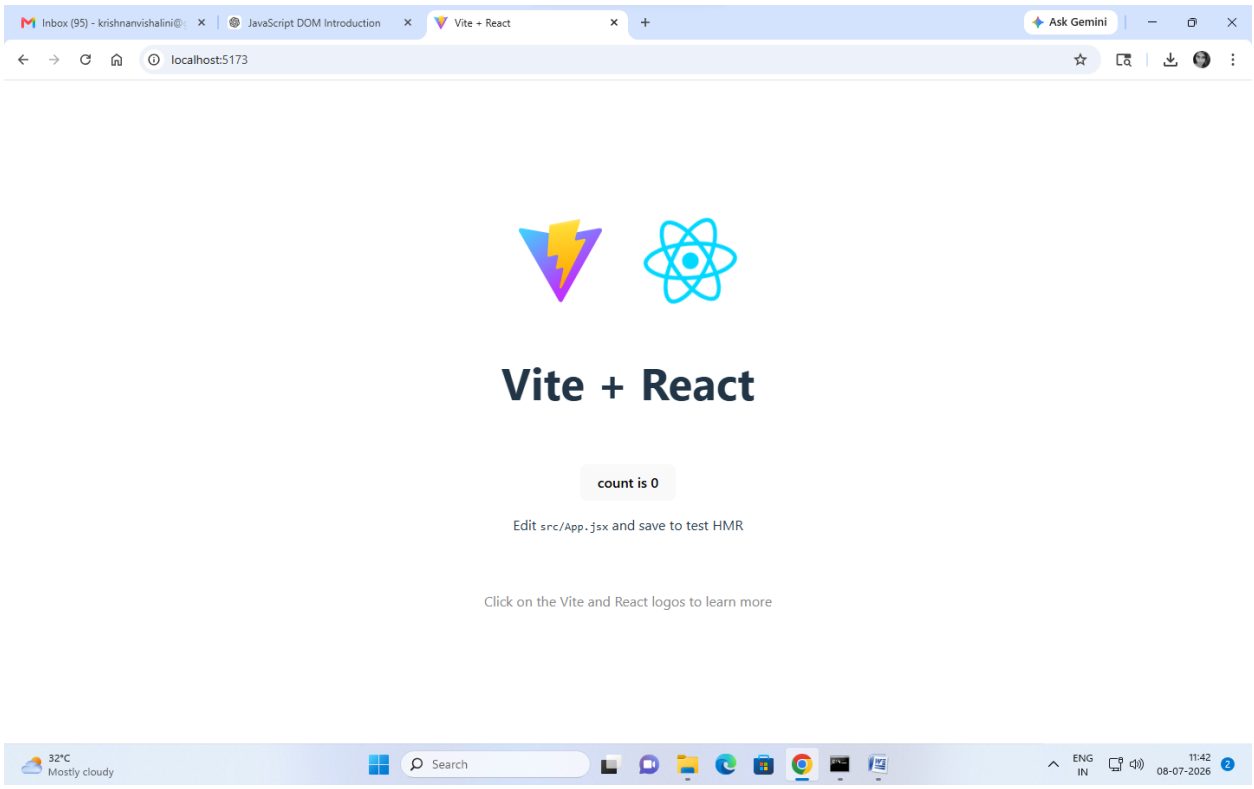
```
npm run dev
```

Step 9

Open the URL displayed in the terminal (usually `http://localhost:5173`) in a web browser.

OUTPUT:

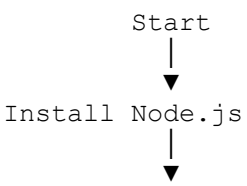
```
C:\Users\sitams.L5-04.000>npm create vite@5
Need to install the following packages:
create-vite@5.5.5
Ok to proceed? (y) Y


> npx
> cva

√ Project name: ... STUDENT-APP
√ Package name: ... student-app
√ Select a framework: » React
√ Select a variant: » JavaScript

Scaffolding project in C:\Users\sitams.L5-04.000\STUDENT-APP...

Done. Now run:

  cd STUDENT-APP
  npm install
  npm run dev


C:\Users\sitams.L5-04.000> cd STUDENT-APP

C:\Users\sitams.L5-04.000\STUDENT-APP>npm install

added 268 packages, and audited 269 packages in 27s

113 packages are looking for funding
  run `npm fund` for details

2 vulnerabilities (1 moderate, 1 high)

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

C:\Users\sitams.L5-04.000\STUDENT-APP>npm run dev

> student-app@0.0.0 dev
> vite


  VITE v5.4.21  ready in 522 ms

  ➜  Local:   http://localhost:5173/
  ➜  Network: use --host to expose
  ➜  press h + enter to show help
```

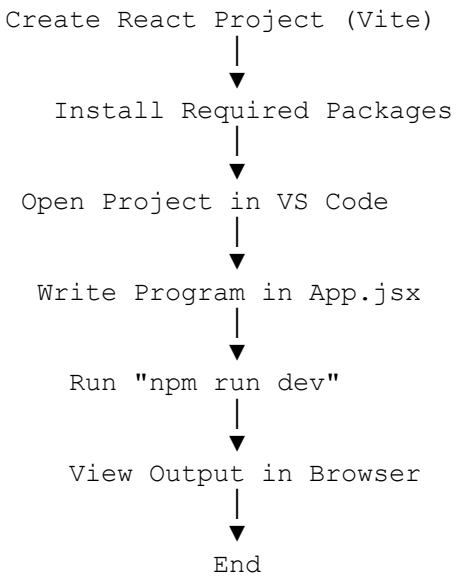


Algorithm

1. Install Node.js on the system.
2. Open Terminal or Command Prompt.
3. Create a React project using Vite.
4. Install the required packages.
5. Open the project in Visual Studio Code.
6. Write the React program in `App.jsx`.
7. Save the file.
8. Execute the program using `npm run dev`.

Flowchart





Note:

- Create **only one React project** for the entire Experiment 2.
- Use the same project for Experiments **2(a)–2(e)**.
- Write all programs in **`App.jsx`** unless instructed otherwise.
- Save the file before checking the browser output.
- Ensure the development server (`npm run dev`) is running while testing your programs.

Absolutely. From now on, this will be our **official Student Lab Manual format**. It is:

- ✓Based on **SITAMS R23 FSD-II Syllabus**
- ✓Tested on **Vite 5.4.21** (works on your college lab)
- ✓No unnecessary theory
- ✓Student-friendly
- ✓Includes Viva Questions **with answers**
- ✓No repeated React concepts already covered in the common section

---

Experiment 2(a)

Write a React Program to Implement a Counter Button Using a React Class Component

Aim

To write and execute a React program that implements a counter button using a React Class Component.

Introduction and Theory

A **Class Component** is a React component created using the **ES6 class** keyword. It extends `React.Component` and can store changing data using **State**.

A **State** is a built-in object used to store values that may change while the program is running. Whenever the state is updated using **`setState()`**, React automatically updates the displayed output without refreshing the webpage.

In this experiment, a counter is initialized with the value **0**. Each time the **Increase** button is clicked, the counter value increases by **1**.

Note: Although modern React mainly uses Functional Components, Class Components are included to understand the fundamentals of React.

Program Logic

- Create a Class Component.
- Initialize the counter value to **0**.
- Display the counter.
- Create an **Increase** button.
- When the button is clicked, increase the counter using `setState()`.
- React automatically updates the displayed value.

Algorithm

Step 1: Open the React project created using **Vite 5**.

Step 2: Open the `src` folder and edit the `App.jsx` file.

Step 3: Remove the default Vite code from `App.jsx`.

Step 4: Create a React Class Component.

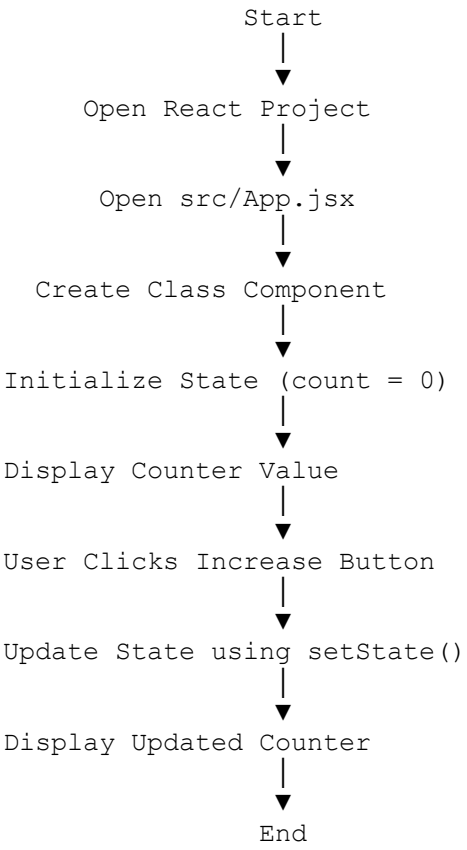
Step 5: Initialize the counter value using State.

Step 6: Create an `increase()` function using `setState()`.

Step 7: Display the counter and create an **Increase** button.

Step 8: Save the file (`Ctrl + S`) and observe the output in the browser.

Flowchart



Program

App.jsx

```
import React, { Component } from "react";

class App extends Component {

  constructor() {
    super();

    this.state = {
      count: 0
    };
  }

  increase = () => {
    this.setState({
      count: this.state.count + 1
    });
  };

  render() {
    return (
      <div style={{ textAlign: "center", marginTop: "50px" }}>

        <h1>React Counter Application</h1>

        <h2>Count : {this.state.count}</h2>

        <button onClick={this.increase}>
          Increase
        </button>

      </div>
    );
  }
}

export default App;
```

Code Explanation

Line 1

```
import React, { Component } from "react";
```

Imports the React library and the `Component` class required to create a Class Component.

Line 3

```
class App extends Component
```

Creates a Class Component named **App** by inheriting the features of `React.Component`.

Lines 5–11

```
constructor() {
  super();

  this.state = {
    count: 0
  };
}
```

- `constructor()` initializes the component.
- `super()` calls the parent class constructor.
- `this.state` creates the State object.
- `count` is initialized with **0**.

Lines 13–17

```
increase = () => {
  this.setState({
    count: this.state.count + 1
  });
};
```

Creates a function that increases the counter value by **1** whenever it is called.

`setState()` updates the State and React automatically refreshes the displayed output.

Lines 19–30

```
render() {
```

The `render()` method returns the JSX that should be displayed in the browser.

Line 24

```
<h2>Count : {this.state.count}</h2>
```

Displays the current value of the counter stored in the State.

Line 26

```
<button onClick={this.increase}>
```

Calls the `increase()` function whenever the button is clicked.

Last Line

```
export default App;
```

Exports the component so that it can be displayed by React.

Output

Initially

```
React Counter Application

Count : 0

[ Increase ]
```

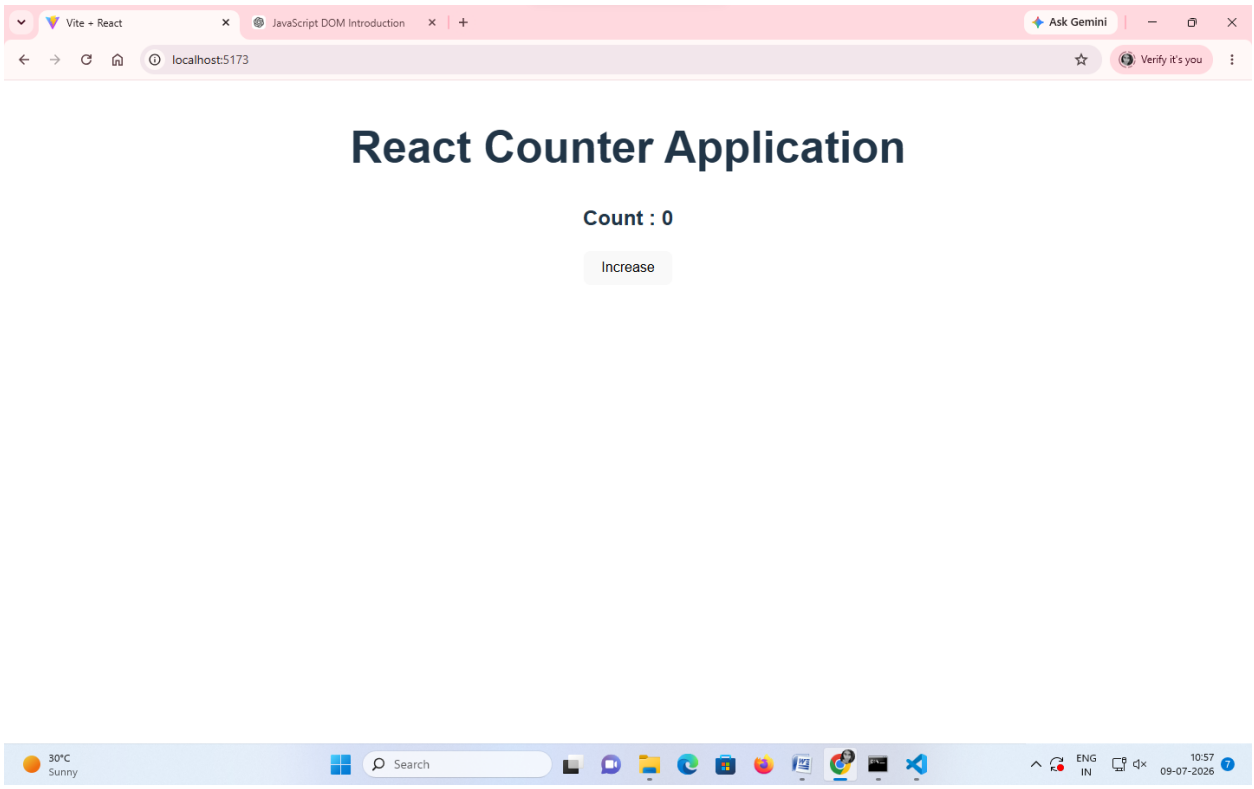
After First Click

```
React Counter Application

Count : 1

[ Increase ]
```

After Three Clicks

```
React Counter Application

Count : 3

[ Increase ]
```

Result



Thus, the React program to implement a counter button using a React Class Component was executed successfully.

Viva Questions and Answers

1. What is a Class Component?

Answer:

A Class Component is a React component created using the ES6 `class` keyword. It extends `React.Component` and can store data using **State**.

---

2. What is State?

Answer:

State is a built-in object used to store data that changes during program execution. Whenever the State changes, React automatically updates the user interface.

---

3. Why is `setState()` used?

Answer:

`setState()` is used to update the State. When it is called, React automatically re-renders the component with the updated values.

---

4. What is the purpose of the `render()` method?

Answer:

The `render()` method returns the JSX that should be displayed in the browser. It is executed whenever the component is displayed or updated.

---

5. Why is `super()` used inside the constructor?

Answer:

`super()` calls the constructor of the parent class (`React.Component`). It must be called before using `this` inside the constructor.

---

6. Why are Functional Components preferred over Class Components?

Answer:

Functional Components are easier to write, require less code, and support Hooks such as `useState()` and `useEffect()`. Therefore, modern React applications mainly use Functional Components.

---

7. What happens if `setState()` is not used?

Answer:

If the State is changed directly without using `setState()`, React will not know that the data has changed, so the user interface will not update correctly.

Troubleshooting

Problem	Solution
Browser still shows the default Vite page	Replace the entire contents of `src/App.jsx` with the program above and save the file.
Changes are not visible	Save the file (`Ctrl + S`) and ensure `npm run dev` is still running.
`localhost:5173` does not open	Check whether the Vite development server is running in the terminal.
Error after editing `App.jsx`	Check for missing brackets `{}`, parentheses `()`, or semicolons if required.

Experiment 2(b)

Write a React Program to Implement a Counter Button Using a Functional Component

---

Aim

To write and execute a React program to implement a counter button using a Functional Component.

Introduction and Theory

A **Functional Component** is a JavaScript function that returns JSX. It is the modern and recommended way of creating React components because it is simpler, requires less code, and supports **React Hooks**.

In Functional Components, **State** is managed using the **`useState()`** Hook.

What is `useState()`?

`useState()` is a React Hook that allows Functional Components to store and update data.

It returns two values:

- **State Variable** – stores the current value.
- **Setter Function** – updates the value.

Syntax

```
const [count, setCount] = useState(0);
```

Here,

- `count` stores the current counter value.
- `setCount()` updates the counter value.
- `0` is the initial value.

Before You Begin

1. Open the React project created in Experiment 2.
2. Open the project in Visual Studio Code.
3. Run the development server.

```
npm run dev
```

4. Open the browser and visit:

```
http://localhost:5173
```

5. Open **src → App.jsx**.
6. Replace the previous experiment code with the following program.

Program Logic

- Create a Functional Component.
- Initialize the counter using `useState(0)`.
- Display the counter value.
- Create an **Increase** button.
- When the button is clicked, increase the counter using `setCount()`.
- React automatically updates the displayed value.

Algorithm

Step 1: Open the React project.

Step 2: Open `src/App.jsx`.

Step 3: Import the `useState` Hook.

Step 4: Create a Functional Component.

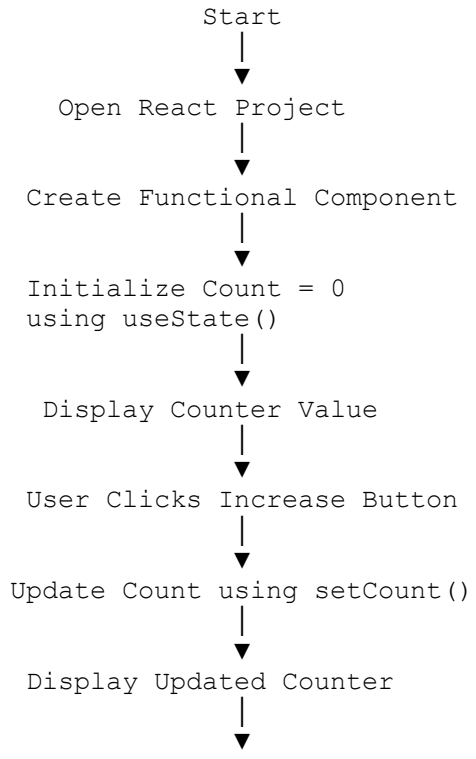
Step 5: Initialize the counter using `useState(0)`.

Step 6: Create a function to increase the counter value.

Step 7: Display the counter and create an **Increase** button.

Step 8: Save the file and observe the output.

Flowchart



```
                    End
```

Program

App.jsx

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

function App() {

  const [count, setCount] = useState(0);

  const increase = () => {
    setCount(count + 1);
  };

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>

      <h1>React Counter Application</h1>

      <h2>Count : {count}</h2>

      <button onClick={increase}>
        Increase
      </button>

    </div>
  );
}

export default App;
```

Code Explanation

Line 1

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

Imports the **`useState` Hook** from the React library.

Line 5

```
const [count, setCount] = useState(0);
```

Creates a state variable named **count**.

- `count` stores the current value.
- `setCount()` updates the value.
- `0` is the initial value.

Lines 7–9

```
const increase = () => {
    setCount(count + 1);
};
```

Creates a function that increases the counter value by **1** whenever it is called.

Line 15

```
<h2>Count : {count}</h2>
```

Displays the current counter value.

Line 17

```
<button onClick={increase}>
```

Calls the `increase()` function whenever the button is clicked.

Last Line

```
export default App;
```

Exports the component so that React can display it.

Output

Initially

```
------------------------------
React Counter Application

Count : 0

[ Increase ]
------------------------------
```

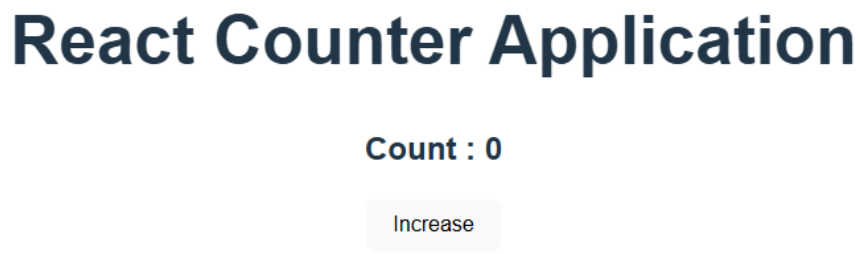
After First Click

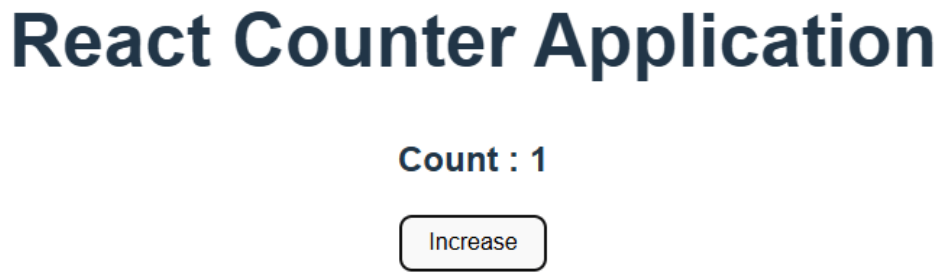
```
------------------------------
React Counter Application

Count : 1

[ Increase ]
------------------------------
```

After Three Clicks

```
------------------------------
React Counter Application

Count : 3

[ Increase ]
------------------------------
```

Result





Thus, the React program to implement a counter button using a Functional Component was executed successfully.

Viva Questions & Answers

1. What is a Functional Component?

Answer:

A Functional Component is a JavaScript function that returns JSX. It is the modern way of creating React components.

---

2. What is `useState()`?

Answer:

`useState()` is a React Hook used to create and manage state in a Functional Component.

---

3. What does `useState(0)` mean?

Answer:

It initializes the state variable with the value **0**.

---

4. What are the two values returned by `useState()`?

Answer:

- State Variable (e.g., `count`)
- Setter Function (e.g., `setCount`)

---

5. What is the purpose of `setCount()`?

Answer:

`setCount()` updates the value of the state variable and automatically refreshes the displayed output.

---

6. What is the difference between `setState()` and `setCount()`?

Answer:

- `setState()` is used in **Class Components**.
- `setCount()` is the setter function returned by `useState()` in **Functional Components**.

---

7. Why are Functional Components preferred over Class Components?

Answer:

Functional Components are simpler, require less code, are easier to understand, and support React Hooks such as `useState()` and `useEffect()`.

Troubleshooting

Problem	Solution
`useState is not defined`	Import `useState` using `import { useState } from "react";`
Counter does not increase	Check whether `setCount()` is called inside the `increase()` function.
Browser still shows previous output	Save the file (`Ctrl + S`) and ensure `npm run dev` is running.
Error after copying code	Check for missing brackets, parentheses, or semicolons.

Think & Explore

1. Add a **Decrease** button.
2. Add a **Reset** button.
3. Increase the counter by **5** instead of **1**.
4. Prevent the counter from going below **0**.

Experiment 2(c)

Write a React Program to Handle Click Events for HTML Button Elements

---

Aim

To write and execute a React program to handle click events for HTML button elements using a Functional Component.

Introduction and Theory

An **event** is an action performed by the user, such as clicking a button, pressing a key, or moving the mouse.

React handles events using **Event Handlers**. An **Event Handler** is a function that executes automatically when a specific event occurs.

In this experiment, the **`onClick`** event is used to execute a function whenever the user clicks a button.

Syntax

```
<button onClick={handleClick}>
    Click Me
```

```
</button>
```

Note: React uses **camelCase** for event names. Therefore, use **`onClick`** instead of **`onclick`**.

Before You Begin

1. Open the React project created in Experiment 2.
2. Open the project in Visual Studio Code.
3. Start the React development server.

```
npm run dev
```

4. Open the browser and visit:

```
http://localhost:5173
```

5. Open **src → App.jsx**.
6. Replace the previous experiment code with the following program.

Program Logic

- Create a Functional Component.
- Create an event handler function named `handleClick()`.
- Display a heading and a button.
- Associate the button with the `onClick` event.
- Display a message whenever the button is clicked.

Algorithm

Step 1: Open the React project.

Step 2: Open the `App.jsx` file.

Step 3: Create a Functional Component.

Step 4: Create an event handler function named `handleClick()`.

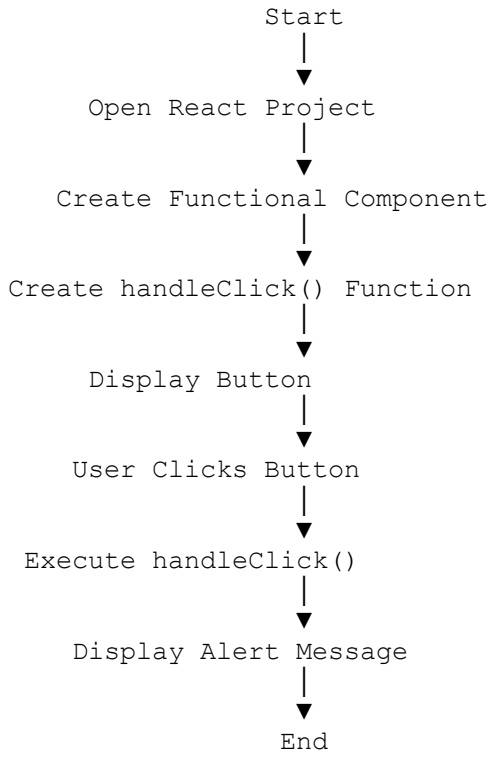
Step 5: Create a button.

Step 6: Attach the `handleClick()` function to the button using the `onClick` event.

Step 7: Save the program.

Step 8: Observe the output in the browser.

Flowchart



Program

App.jsx

```
function App() {

  const handleClick = () => {
    alert("Button Clicked Successfully!");
  };

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>
      <h1>React Button Click Event</h1>
```

```
      <button onClick={handleClick}>
        Click Me
      </button>
    </div>
  );
}

export default App;
```

Code Explanation

Line 1

```
function App() {
```

Creates a Functional Component named **App**.

Lines 3–5

```
const handleClick = () => {
  alert("Button Clicked Successfully!");
};
```

Creates an event handler function named **handleClick()**. When the button is clicked, an alert box displaying the message **"Button Clicked Successfully!"** appears.

Line 10

```
<button onClick={handleClick}>
```

Associates the button with the `handleClick()` function using the **`onClick`** event.

Whenever the button is clicked, the `handleClick()` function is executed.

Line 11

```
Click Me
```

Displays the text **"Click Me"** on the button.

Last Line

```
export default App;
```

Exports the component so that React can render it in the browser.

Output

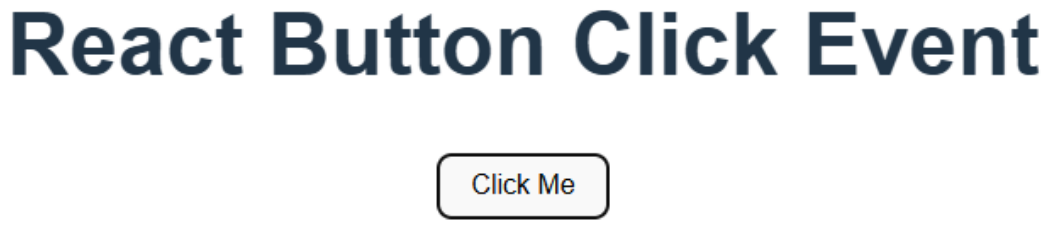
Before Clicking the Button

```
----------------------------------

    React Button Click Event

          [ Click Me ]

----------------------------------
```

After Clicking the Button

```
----------------------------------

        Alert Box

Button Clicked Successfully!

          [ OK ]

----------------------------------
```

Result



Thus, the React program to handle click events for HTML button elements was executed successfully.

Viva Questions & Answers

1. What is an event in React?

Answer:

An event is an action performed by the user, such as clicking a button, pressing a key, or moving the mouse.

---

2. What is an Event Handler?

Answer:

An Event Handler is a JavaScript function that is executed whenever a specific event occurs.

---

3. Which event is used to handle button click events in React?

Answer:

The `onClick` event is used to handle button click events.

---

4. Why is `onClick` written in camelCase?

Answer:

React follows JavaScript naming conventions, so event names are written in **camelCase**. Therefore, `onClick` is used instead of `onclick`.

---

5. What is the difference between `onclick` and `onClick`?

Answer:

HTML	React
`onclick`	`onClick`
Lowercase	camelCase
HTML attribute	React event attribute

---

6. Can one event handler be used for multiple buttons?

Answer:

Yes. The same event handler function can be attached to multiple buttons if they perform the same action.

---

7. What happens if the `onClick` event is not used?

Answer:

The button will be displayed, but clicking it will not perform any action because no function is associated with it.

---

8. Can an event handler perform tasks other than displaying an alert?

Answer:

Yes. An event handler can update state, display messages, navigate to another page, perform calculations, or execute any JavaScript code.

Perfect! Since **Experiment 2(c)** introduced **Event Handling**, we won't repeat it. In **Experiment 2(d)**, we'll introduce only the **new concept: Conditional Rendering**.

Experiment 2(d)

Write a React Program to Implement Conditional Rendering

Aim

To write and execute a React program to implement Conditional Rendering using a Functional Component.

Introduction and Theory

Conditional Rendering is a technique in React that allows different content to be displayed based on a condition.

Instead of showing the same output every time, React checks whether a condition is **true** or **false** and displays the appropriate content.

In React, Conditional Rendering can be implemented using:

- `if` statement
- `if...else` statement
- Ternary Operator (`condition ? true : false`)
- Logical AND Operator (`&&`)

In this experiment, the **Ternary Operator** is used to display different messages based on the login status.

Before You Begin

1. Open the React project created in Experiment 2.
2. Open the project in Visual Studio Code.
3. Start the React development server.

```
npm run dev
```

4. Open the browser and visit:

```
http://localhost:5173
```

5. Open **src → App.jsx**.
6. Replace the previous experiment code with the following program.

Program Logic

- Create a Functional Component.
- Create a state variable to store the login status.
- Initially, the user is logged out.
- When the **Login** button is clicked, change the login status.
- Display different messages based on the login status using Conditional Rendering.

Algorithm

Step 1: Open the React project.

Step 2: Open the `App.jsx` file.

Step 3: Import the `useState()` Hook.

Step 4: Create a state variable to store the login status.

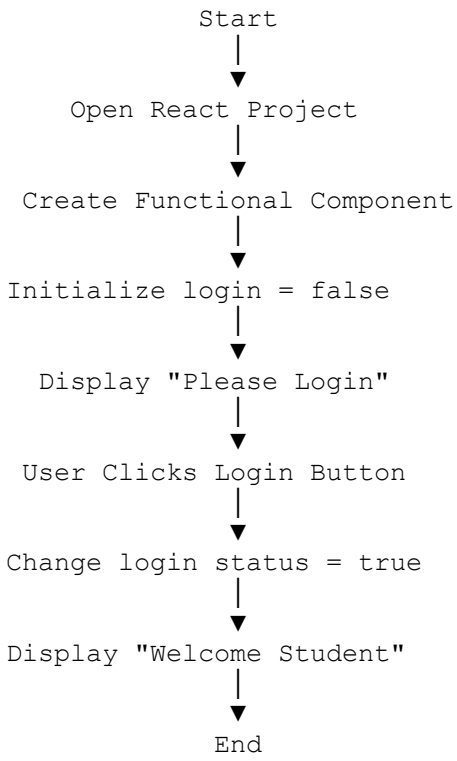
Step 5: Create a button to change the login status.

Step 6: Use the Ternary Operator to display different messages.

Step 7: Save the program.

Step 8: Observe the output in the browser.

Flowchart



Program

App.jsx

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

function App() {

  const [isLoggedIn, setIsLoggedIn] = useState(false);

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>

      <h1>Conditional Rendering Example</h1>

      <h2>
        {isLoggedIn ? "Welcome Student" : "Please Login"}
      </h2>

      <button onClick={() => setIsLoggedIn(!isLoggedIn)}>
        {isLoggedIn ? "Logout" : "Login"}
      </button>
```

```
    </div>
  );
}

export default App;
```

Code Explanation

Line 1

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

Imports the **`useState()` Hook** to store and update the login status.

Line 5

```
const [isLoggedIn, setIsLoggedIn] = useState(false);
```

Creates a state variable named **isLoggedIn**.

- `false` means the user is not logged in.
- `setIsLoggedIn()` updates the login status.

Line 12

```
{isLoggedIn ? "Welcome Student" : "Please Login"}
```

Uses the **Ternary Operator**.

- If `isLoggedIn` is **true**, it displays **"Welcome Student"**.
- Otherwise, it displays **"Please Login"**.

Line 15

```
<button onClick={() => setIsLoggedIn(!isLoggedIn)}>
```

When the button is clicked:

- `false` changes to `true`

- `true` changes to `false`

This changes the login status.

Line 16

```
{isLoggedIn ? "Logout" : "Login"}
```

Changes the button text dynamically.

- When logged out → **Login**
- When logged in → **Logout**

Last Line

```
export default App;
```

Exports the component so that React can render it.

Output

Initially

```
---------------------------------------

Conditional Rendering Example

Please Login

[ Login ]

---------------------------------------
```

After Clicking Login

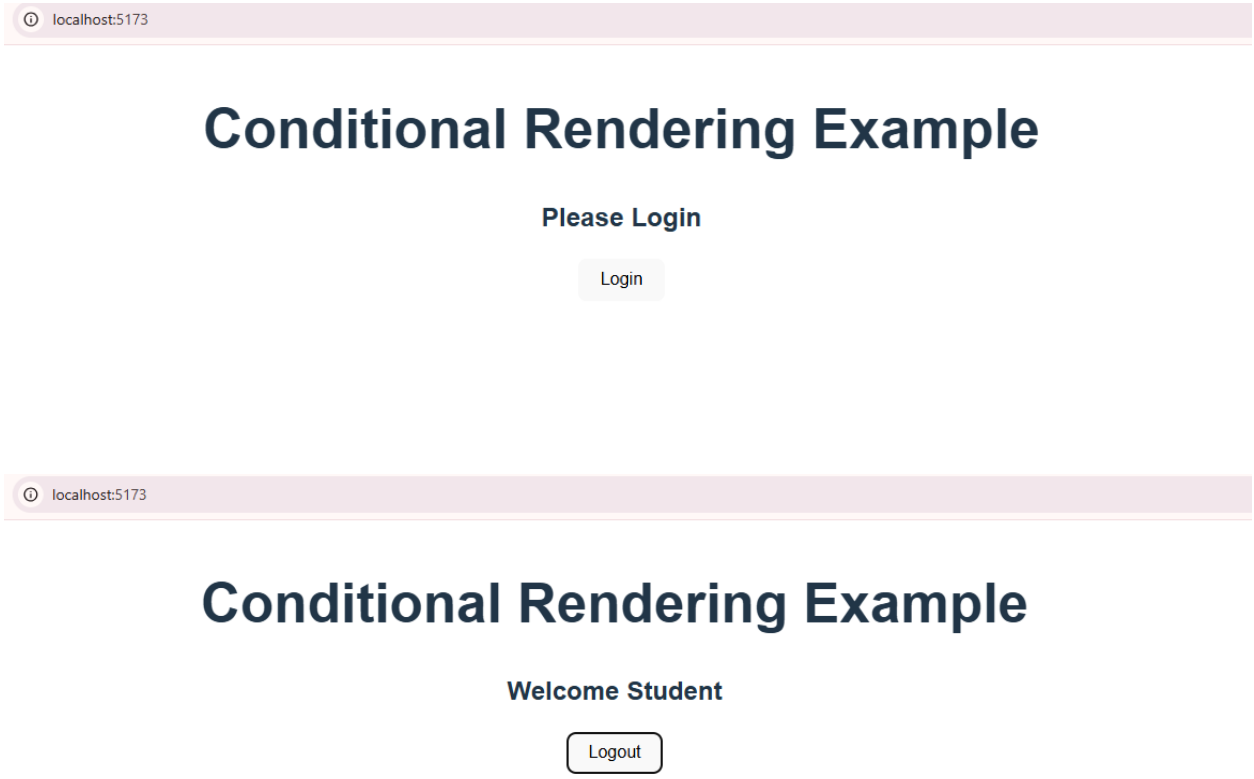
```
---------------------------------------

Conditional Rendering Example

Welcome Student
```

```
[ Logout ]

--------------------------------------
```

After Clicking Logout

```
--------------------------------------
Conditional Rendering Example

Please Login

[ Login ]

--------------------------------------
```

Result



Thus, the React program to implement Conditional Rendering was executed successfully.

Viva Questions & Answers

1. What is Conditional Rendering?

Answer:

Conditional Rendering is a technique in React used to display different content based on a condition.

---

2. Which operators are commonly used for Conditional Rendering?

Answer:

- `if`
- `if...else`
- Ternary Operator (`condition ? true : false`)
- Logical AND (`&&`)

---

3. What is the Ternary Operator?

Answer:

The Ternary Operator is a shorthand form of the `if...else` statement.

Syntax:

```
condition ? trueStatement : falseStatement
```

---

4. What is the purpose of `useState(false)` in this program?

Answer:

It creates a state variable named `isLoggedIn` and initializes it with the value **false**, indicating that the user is initially logged out.

---

5. What does `!isLoggedIn` mean?

Answer:

The `!` (NOT) operator reverses the current value.

- `false` becomes `true`
- `true` becomes `false`

---

6. Why is the button text changing automatically?

Answer:

The button text is displayed using Conditional Rendering. When the login status changes, React automatically updates the displayed text.

---

7. Can Conditional Rendering be used to display entire components?

Answer:

Yes. Conditional Rendering can display or hide complete React components, sections of a webpage, or individual elements.

---

8. Give one real-world application of Conditional Rendering.

Answer:

Examples include:

- Login/Logout pages
- Admin/User dashboards
- Shopping cart with "Cart Empty" message
- Showing loading messages while data is being fetched

NOTE

Observe that **both the message and the button text change** based on the same condition. This reinforces that **conditional rendering can control multiple elements simultaneously**, not just a single line of text. It's a simple enhancement that strengthens their understanding without increasing the complexity of the program.

Experiment 2(e)

Write a React Program to Display Text Using String Literals

Aim

To write and execute a React program to display text using Template (String) Literals in React.

Introduction and Theory

A **Template Literal** (also called a **String Literal**) is a modern JavaScript feature used to create strings that can include variables or expressions.

Template Literals use **backticks ( )** instead of single (`' '`) or double (`" "`) quotes.

Variables are inserted into the string using **`${}`**.

This makes the code easier to read and eliminates the need for string concatenation.

Syntax

```
`Welcome ${name}`
```

Example

```
const name = "Rahul";

const message = `Welcome ${name}`;
```

Output

```
Welcome Rahul
```

Before You Begin

1. Open the React project created in Experiment 2.
2. Open the project in Visual Studio Code.

3. Start the React development server.

```
npm run dev
```

4. Open the browser and visit:

```
http://localhost:5173
```

5. Open **src → App.jsx**.
6. Replace the previous experiment code with the following program.

Program Logic

- Create a Functional Component.
- Declare variables to store the student's name and department.
- Create a Template Literal using backticks.
- Display the generated message on the webpage.

Algorithm

Step 1: Open the React project.

Step 2: Open the `App.jsx` file.

Step 3: Create variables to store the student's name and department.

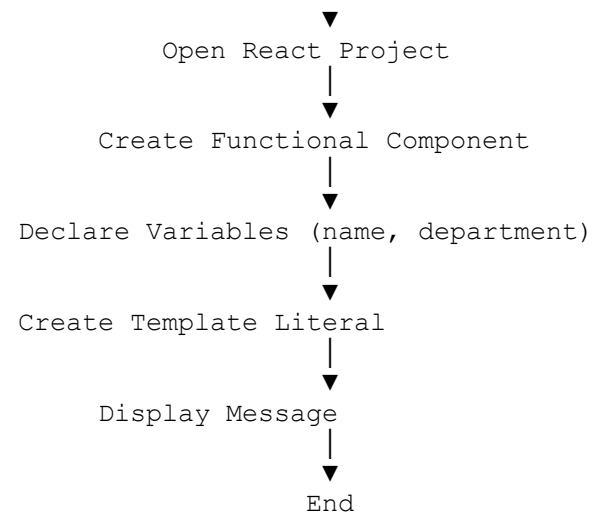
Step 4: Create a Template Literal using `${}`.

Step 5: Display the message using JSX.

Step 6: Save the program.

Step 7: Observe the output in the browser.

Flowchart



Program

App.jsx

```
function App() {

  const name = "Moses";
  const department = "Data Science";

  const message = `Welcome ${name} to the ${department} Department`;

  return (
    <div style={{ textAlign: "center", marginTop: "50px" }}>

      <h1>Template Literal Example</h1>

      <h2>{message}</h2>

    </div>
  );
}

export default App;
```

Code Explanation

Line 3

```
const name = "Moses";
```

Stores the student's name.

Line 4

```
const department = "Data Science";
```

Stores the department name.

Line 6

```
const message = `Welcome ${name} to the ${department} Department`;
```

Creates a **Template Literal**.

- Backticks ( ) create the Template Literal.
- `${name}` inserts the value stored in the variable `name`.
- `${department}` inserts the value stored in the variable `department`.

Line 12

```
<h2>{message}</h2>
```

Displays the complete message on the webpage.

Last Line

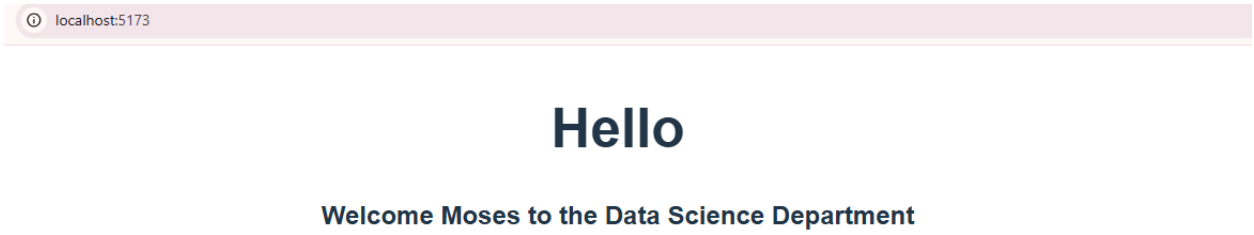
```
export default App;
```

Exports the component so that React can render it.

Output

```
------------------------------------------

        Template Literal Example

Welcome Moses to the Data Science Department

------------------------------------------
```

Result



Thus, the React program to display text using Template Literals was executed successfully.

Viva Questions & Answers

1. What is a Template Literal?

Answer:

A Template Literal is a modern JavaScript feature used to create strings containing variables or expressions. It uses **backticks ( )** instead of single or double quotes.

---

2. What is the purpose of `${}`?

Answer:

`${}` is used to insert the value of a variable or expression into a Template Literal.

---

3. Which symbol is used to create a Template Literal?

Answer:

Backticks ( ) are used to create a Template Literal.

---

4. What is the advantage of Template Literals over string concatenation?

Answer:

Template Literals make the code shorter, easier to read, and easier to maintain by avoiding the use of the + operator for combining strings.

---

5. Can multiple variables be used inside one Template Literal?

Answer:

Yes. Multiple variables and expressions can be included using `${}`.

Example:

```
`Hello ${name}, Welcome to ${department}`
```

---

6. Can JavaScript expressions be used inside `${}`?

Answer:

Yes. Variables, arithmetic expressions, and function calls can all be used inside `${}`.

Example:

```
`${10 + 20}`
```

Output

```
30
```

---

7. What will be the output of the following statement?

```
const city = "Chittoor";
console.log(`Welcome to ${city}`);
```

Answer:

```
Welcome to Chittoor
```

---

8. Where are Template Literals commonly used in React?

Answer:

Template Literals are commonly used to display dynamic text such as:

- Welcome messages
- User names
- Product names
- Scores
- Prices
- Student details
- Dashboard messages

Experiment 3

Important Concepts of React.js

Learning Objectives

After completing this experiment, students will be able to:

- Understand the purpose of React Hooks.
- Create and manage state using the `useState()` Hook.
- Fetch data from an API using the `useEffect()` Hook.
- Share data between components using Props.
- Create and handle forms in React.
- Display lists dynamically using the `map()` function.
- Build simple, interactive React applications.

Introduction & Theory

In the previous experiment, you learned the basics of React and created simple components. In this experiment, you will explore some of the most important features that make React powerful and suitable for developing modern web applications.

React applications are built using **components**. Components become more useful when they can **store data, respond to user actions, communicate with other components, and display changing information**. React provides these capabilities through Hooks, Props, Forms, and list rendering.

This experiment introduces five essential concepts that are widely used in almost every React application:

1. useState()

The **useState() Hook** allows a functional component to store and update data called **state**.

Whenever the state changes, React automatically updates the user interface without reloading the page.

Example:

- Counter application
- Like button
- Theme switcher

---

2. useEffect()

The **useEffect() Hook** performs actions after a component is rendered.

It is commonly used to:

- Fetch data from an API
- Load information when the page opens
- Perform background operations
- Synchronize data

---

3. Props

Props (Properties) allow a parent component to send data to a child component.

Props make components reusable because the same component can display different information based on the data it receives.

---

4. Forms

Forms allow users to enter information through input fields such as text boxes, radio buttons, checkboxes, and buttons.

React uses state variables to store and manage form data, making it easier to validate and process user input.

---

5. Iterative Rendering using map()

The **map()** function is used to display multiple items from an array.

Instead of writing repetitive HTML, React creates UI elements dynamically from data, making applications more efficient and easier to maintain.

Real-World Applications

The concepts covered in this experiment are used in many everyday applications.

React Concept	Real-World Example
useState()	Shopping cart quantity, Like button
useEffect()	Weather information, News feed
Props	Product cards, Student profiles
Forms	Login page, Registration form
map()	Product list, Student list, Employee records

Folder Structure

Throughout **Experiment 3**, students will use **one React project**.

```
student-app
│
├── src
│    ├── App.jsx
│    ├── App.css
│    ├── Student.jsx (Only for Experiment 3(c))
│    ├── main.jsx
│    └── index.css
│
├── package.json
├── vite.config.js
└── node_modules
```

Note: `App.css` is created once and reused throughout Experiment 3. Students only modify `App.jsx` (and `Student.jsx` for Experiment 3(c)) while performing the sub-experiments.

```
/* Common CSS for Experiment 3 */

*{

   margin:0;

   padding:0;

   box-sizing:border-box;

}
```

```
body{
    font-family:Arial, Helvetica, sans-serif;
    background:#f4f6fb;
}

.container{
    min-height:100vh;
    display:flex;
    justify-content:center;
    align-items:center;
    padding:30px;
}

.lab-card{
    width:500px;
    background:white;
    padding:30px;
    border-radius:10px;
    box-shadow:0 4px 12px rgba(0,0,0,.2);
}

h1{
    text-align:center;
```

```
    margin-bottom:20px;
    color:#1f3b73;
}

h2{
    margin-bottom:15px;
}

p{
    margin:8px 0;
}

button{
    padding:10px 20px;
    background:#1f3b73;
    color:white;
    border:none;
    border-radius:5px;
    cursor:pointer;
}

button:hover{
    background:#3057a5;
```

```
}

label{
    display:block;
    margin-top:10px;
    font-weight:bold;
}

input{
    width:100%;
    padding:10px;
    margin:8px 0 15px;
    border:1px solid #ccc;
    border-radius:5px;
}

hr{
    margin:15px 0;
}

ul{
    margin-left:20px;
}
```

```
li{
    margin:8px 0;
}

.result-box{
    margin-top:20px;
    padding:15px;
    border-left:5px solid #1f3b73;
    background:#eef4ff;
}
```

Experiment 3(a)

Write a React Program to Implement a Counter Button Using React `useState()` Hook

---

Aim

To write and execute a React program to implement a counter button using the **`useState()` Hook.

Introduction & Theory

Many applications display information that changes while the program is running. Examples include the number of items in a shopping cart, the number of likes on a social media post, or the score in a game.

In React, such changing information is called **State**.

The **`useState()` Hook** is used in Functional Components to create and manage state. Whenever the state value changes, React automatically updates only the affected part of the webpage without reloading the entire page.

This makes React applications fast, interactive, and easy to maintain.

Syntax

```
const [count, setCount] = useState(0);
```

Where:

- **count** → Stores the current value.
- **setCount()** → Updates the value.
- **0** → Initial value.

Program Logic

The program creates a counter with an initial value of **0**.

When the user clicks the **Increase** button, the `increaseCount()` function updates the state using `setCount()`. React detects the state change and immediately displays the updated count on the webpage.

Algorithm

Step 1: Open the existing React project.

Step 2: Open the **App.jsx** file.

Step 3: Import the `useState()` Hook.

Step 4: Create a state variable with an initial value of **0**.

Step 5: Create a function to increase the counter value.

Step 6: Display the counter value.

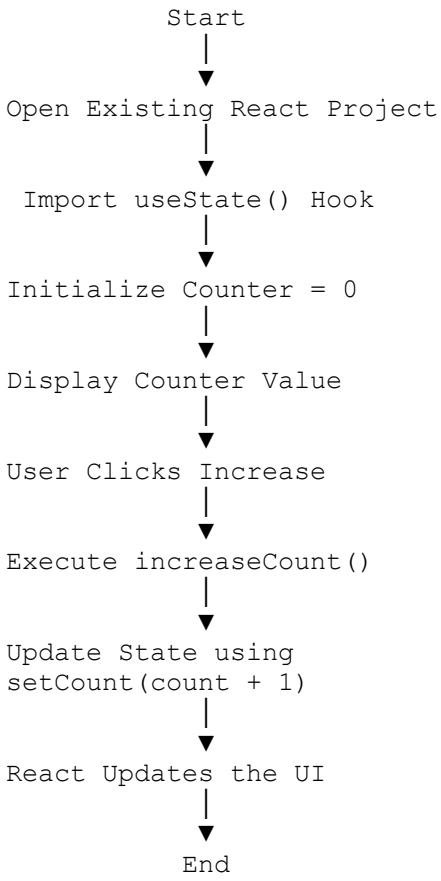
Step 7: Create an **Increase** button and connect it to the function.

Step 8: Save the program.

Step 9: Execute the application using:

```
npm run dev
```

Flowchart



Program

App.jsx

```
import { useState } from "react";
import "./App.css";

function App() {

    const [count, setCount] = useState(0);

    function increaseCount() {
        setCount(count + 1);
    }

    return (

        <div className="container">

            <div className="lab-card">

                <h1>Student Counter</h1>

                <h2>Current Count: {count}</h2>

                <button onClick={increaseCount}>
                    Increase
                </button>

            </div>

        </div>

    );

}

export default App;
```

Code Explanation

1. Import Statements

```
import { useState } from "react";
import "./App.css";
```

- `useState` is imported from the React library to create and manage state.
- `App.css` provides the common styling used throughout Experiment 3.

---

2. Creating the State Variable

```
const [count, setCount] = useState(0);
```

This statement creates a state variable.

- **count** stores the current counter value.
- **setCount()** updates the counter value.
- **0** is the initial value displayed when the application starts.

---

3. Creating the Event Function

```
function increaseCount() {
    setCount(count + 1);
}
```

The `increaseCount()` function increases the counter value by **1** whenever it is called.

---

4. JSX Structure

```
<div className="container">

    <div className="lab-card">

        <h1>Student Counter</h1>

        <h2>Current Count: {count}</h2>

        <button onClick={increaseCount}>
            Increase
        </button>

    </div>

</div>
```

This section creates the user interface.

- The **container** centers the content on the page.
- The **lab-card** displays the counter inside a styled card.
- The `<h2>` element displays the current value stored in the state.
- Clicking the button calls the `increaseCount()` function.

---

5. React Concept Used

This experiment demonstrates the **`useState()` Hook**.

Whenever `setCount()` updates the value, React automatically refreshes only the displayed counter without reloading the webpage.

---

6. Export Statement

```
export default App;
```

Exports the `App` component so it can be rendered by React.

Sample Output

Initial Output

```
----------------------------

     Student Counter

Current Count : 0

      [ Increase ]

----------------------------
```

After Clicking the Button Once

```
----------------------------

     Student Counter

Current Count : 1

      [ Increase ]

----------------------------
```

After Clicking the Button Again

```
----------------------------

     Student Counter

Current Count : 2
```

```
      [ Increase ]

--------------------------
```

Result

Thus, the React program to implement a counter button using the **`useState()` Hook** was executed successfully.

Remember

✅`useState()` is used to manage changing data in Functional Components.

✅Always update the state using the setter function (`setCount()`).

❌Do not update the state directly.

```
count = count + 1;
```

✔Correct way:

```
setCount(count + 1);
```

Viva Questions & Answers

Q1. What is State in React?

Answer:
State is data that can change while a React application is running. When the state changes, React automatically updates the user interface.

Q2. What is the purpose of the `useState()` Hook?

Answer:
The `useState()` Hook is used to create and manage state in a Functional Component.

Q3. Why do we use `setCount()` instead of `count = count + 1`?

Answer:
`setCount()` informs React that the state has changed. React then re-renders the component and updates the displayed output. Directly changing the variable does not trigger a UI update.

Experiment 3(b)

Write a React Program to Fetch Data from an API Using React `useEffect()` Hook

---

Aim

To write and execute a React program to fetch data from an API using the **`useEffect()` Hook.

Introduction & Theory

Many modern web applications display information that comes from an online server instead of being stored inside the program. This information is usually obtained through an **Application Programming Interface (API)**.

The **`useEffect()` Hook** is used to perform actions after a React component is rendered. One of its most common uses is fetching data from an API.

In this experiment, the application connects to a free online API, retrieves student information, stores it using **`useState()`**, and displays the first three records on the webpage.

Syntax

```
useEffect(() => {

    // Code to execute

}, []);
```

Explanation

- **useEffect()** executes the given function after the component is rendered.
- **[]** is called the **dependency array**.
- An empty dependency array means the effect runs **only once** when the component is first loaded.

Program Logic

When the application starts, the `useEffect()` Hook automatically calls an online API using the `fetch()` function.

The received data is converted into JSON format and stored in the **students** state variable.

Only the **first three students** are displayed on the webpage using the `map()` function.

Algorithm

Step 1: Open the existing React project.

Step 2: Open the **App.jsx** file.

Step 3: Import `useState` and `useEffect`.

Step 4: Create a state variable to store the API data.

Step 5: Use `useEffect()` to call the API.

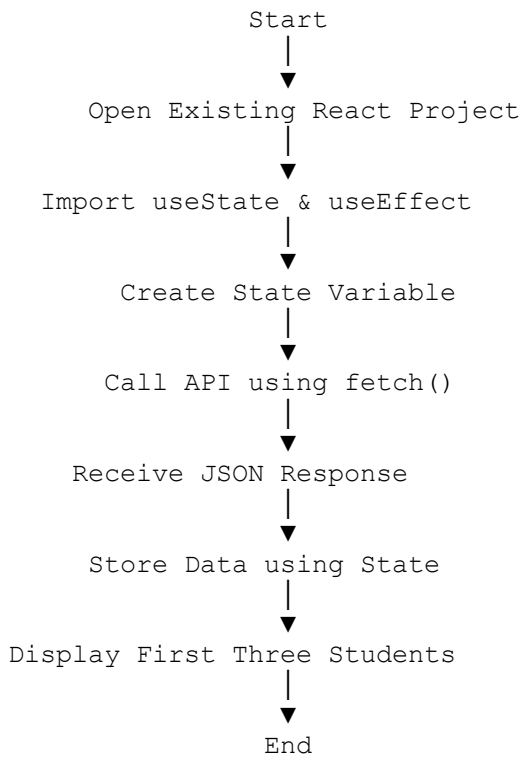
Step 6: Convert the received data into JSON.

Step 7: Store the data in the state variable.

Step 8: Display the first three student records.

Step 9: Save and execute the application.

Flowchart



Program

App.jsx

```
import { useState, useEffect } from "react";
import "./App.css";

function App() {

    const [students, setStudents] = useState([]);

    useEffect(() => {

        fetch("https://jsonplaceholder.typicode.com/users")
            .then((response) => response.json())
            .then((data) => {
                setStudents(data);
            });

    }, []);

    return (

        <div className="container">

            <div className="lab-card">
```

```
                <h1>Student List</h1>
                {students.slice(0, 3).map((student) => (
                    <div key={student.id}>
                        <h3>{student.name}</h3>
                        <p><strong>Email:</strong> {student.email}</p>
                        <p><strong>City:</strong> {student.address.city}</p>
                        <hr />
                    </div>
                ))}
            </div>
        </div>
    );
}
export default App;
```

Code Explanation

1. Import Statements

```
import { useState, useEffect } from "react";
import "./App.css";
```

- `useState()` stores the fetched data.
- `useEffect()` calls the API after the component is rendered.
- `App.css` provides the common styling used throughout Experiment 3.

---

2. Creating the State Variable

```
const [students, setStudents] = useState([]);
```

Creates a state variable named **students**.

Initially, it stores an empty array because no data has been received from the API.

---

3. Fetching Data

```
fetch("https://jsonplaceholder.typicode.com/users")
```

Sends a request to the online API to retrieve user information.

---

4. Converting JSON Data

```
.then((response) => response.json())
```

Converts the API response into JSON format so that React can use the data.

---

5. Updating the State

```
.then((data) => {
    setStudents(data);
});
```

Stores the received data in the **students** state variable.

Updating the state causes React to automatically refresh the user interface.

---

6. Displaying the Data

```
students.slice(0, 3).map((student) => (
```

- `slice(0,3)` displays only the first three students.
- `map()` reads each student object and creates JSX dynamically.

---

7. React Concept Used

This experiment demonstrates how **`useEffect()`** works together with **`useState()`**.

- `useEffect()` fetches the data.
- `useState()` stores the data.

- React automatically updates the webpage after the data is received.

---

8. Export Statement

```
export default App;
```

Exports the `App` component so it can be rendered by React.

Sample Output

```
-----------------------------------

          Student List

-----------------------------------

Leanne Graham

Email : Sincere@april.biz

City : Gwenborough

-----------------------------------

Ervin Howell

Email : Shanna@melissa.tv

City : Wisokyburgh

-----------------------------------

Clementine Bauch

Email : Nathan@yesenia.net

City : McKenziehaven

-----------------------------------
```

Result

Thus, the React program to fetch data from an API using the **`useEffect()` Hook** was executed successfully.

Remember

✅Import both `useState()` and `useEffect()` before using them.

✅Use an empty dependency array (`[]`) to execute the effect only once.

✅Store fetched data using State.

Viva Questions & Answers

Q1. What is an API?

Answer:
An API (Application Programming Interface) allows different software applications to communicate and exchange data.

Q2. What is the purpose of the `useEffect()` Hook?

Answer:
The `useEffect()` Hook is used to perform side effects in React, such as fetching data, updating the document title, or interacting with external systems after a component is rendered.

Q3. Why is an empty dependency array (`[]`) used?

Answer:
An empty dependency array ensures that the `useEffect()` function runs only once when the component is first loaded.

Q4. Why do we use `fetch()`?

Answer:
The `fetch()` function sends a request to an API and retrieves data from a server.

Note

- Your computer must be connected to the Internet.
- The API URL must be correct.
- If the API server is unavailable or there is no Internet connection, the program may not display any data even if the React code is correct.

Experiment 3(c)

Write a React Program with Two React Components Sharing Data Using Props

---

Aim

To write and execute a React program that shares data between two React components using **Props**.

Introduction & Theory

As React applications grow, writing all the code in a single component becomes difficult to manage. React solves this problem by dividing the application into **components**.

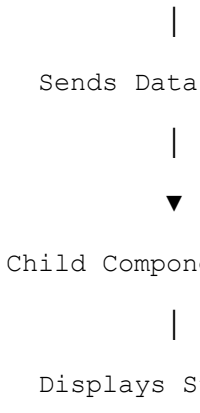
Often, one component needs to send information to another component. This is done using **Props (Properties)**.

Props allow data to flow from a **Parent Component** to a **Child Component**. They help make components reusable because the same component can display different information based on the data it receives.

Note: Props are **read-only**, which means the Child Component can use the data but should not modify it.

Parent–Child Relationship

```
Parent Component (App)
```



Program Logic

The program consists of two components:

- **App Component (Parent)** – Passes the student details.
- **Student Component (Child)** – Receives the details through Props and displays them.

The same **Student** component is reused to display the details of **Daniel** and **Esther**, demonstrating component reusability.

Algorithm

Step 1: Open the existing React project.

Step 2: Create a new file named **Student.jsx** inside the **src** folder.

Step 3: Create a Functional Component in **Student.jsx**.

Step 4: Pass student details from **App.jsx** using Props.

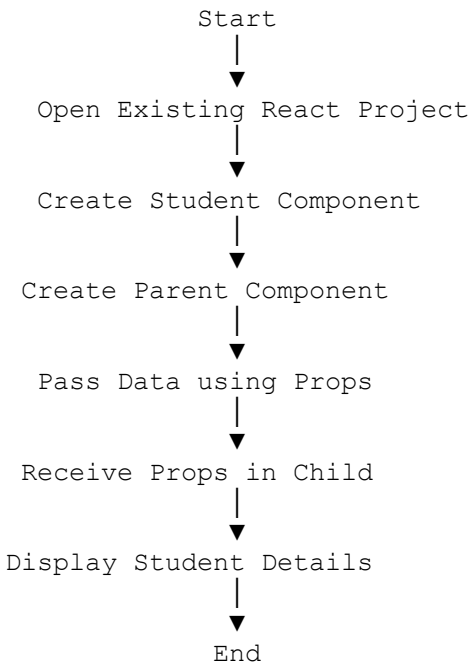
Step 5: Receive the Props inside **Student.jsx**.

Step 6: Display the student details.

Step 7: Save the program.

Step 8: Execute the application.

Flowchart



Program

Student.jsx

```
function Student(props) {

    return (

        <div>

            <h2>Student Details</h2>

            <p><strong>Name :</strong> {props.name}</p>

            <p><strong>Roll Number :</strong> {props.rollNo}</p>

            <p><strong>Department :</strong> {props.department}</p>

            <hr />

        </div>

    );

}
```

```
export default Student;
```

App.jsx

```
import "./App.css";
import Student from "./Student";

function App() {

    return (

        <div className="container">

            <div className="lab-card">

                <h1>Student Management System</h1>

                <Student
                    name="Daniel"
                    rollNo="23DS001"
                    department="Data Science"
                />

                <Student
                    name="Esther"
                    rollNo="23DS002"
                    department="Data Science"
                />

            </div>

        </div>

    );

}

export default App;
```

Code Explanation

1. Import Statements

```
import "./App.css";
import Student from "./Student";
```

- `App.css` provides the common styling for Experiment 3.
- `Student.jsx` is imported so that it can be used inside the `App` component.

---

2. Creating the Child Component

```
function Student(props)
```

Creates a Functional Component named **Student**.

The `props` object receives data sent by the Parent Component.

---

3. Displaying Props

```
props.name
props.rollNo
props.department
```

These expressions display the values received from the Parent Component.

---

4. Passing Props

```
<Student
    name="Daniel"
    rollNo="23DS001"
    department="Data Science"
/>
```

The Parent Component sends:

- Student Name
- Roll Number
- Department

to the Child Component through Props.

---

5. Component Reusability

```
<Student ... />

<Student ... />
```

The same **Student** component is used twice with different values.

This demonstrates one of React's most important advantages—**component reusability**.

---

6. React Concept Used

This experiment demonstrates **Props**.

Props allow data to flow from a Parent Component to a Child Component, making components reusable and reducing code duplication.

---

7. Export Statement

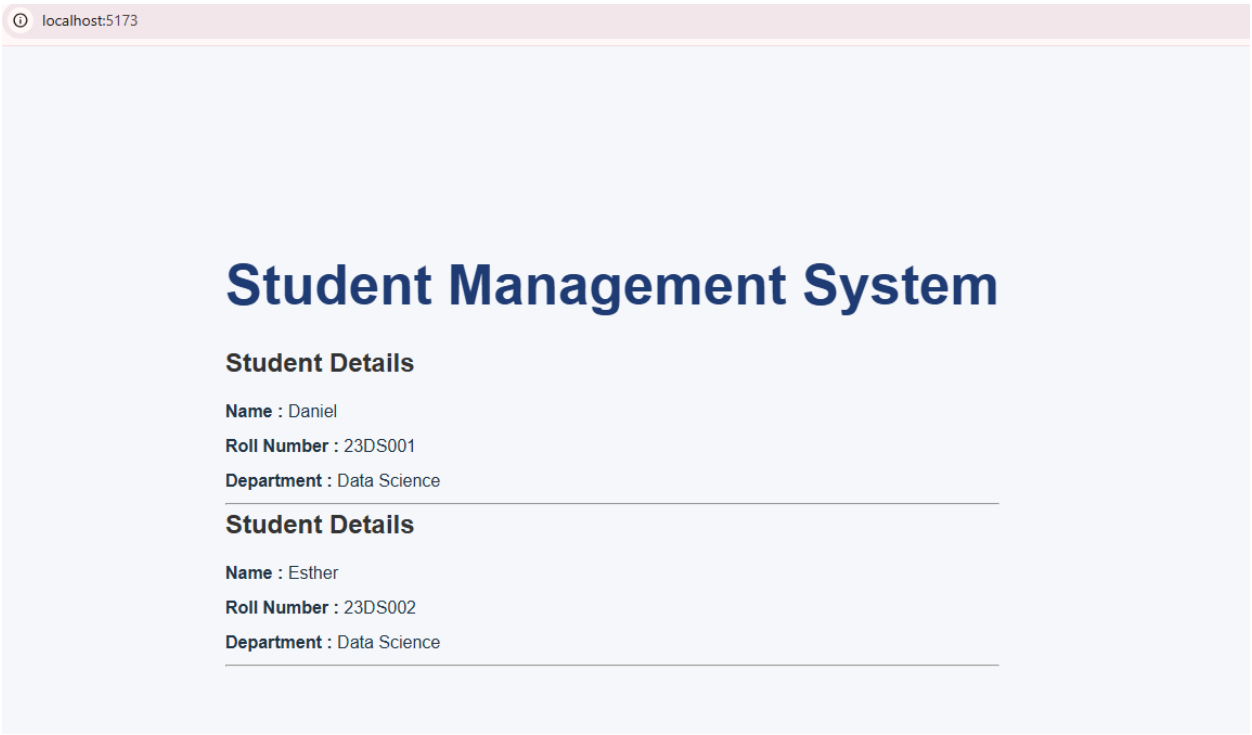
```
export default Student;
```

Exports the `Student` component so it can be used in other files.

Sample Output

```
-----------------------------------

    Student Management System

-----------------------------------

Student Details

Name : Daniel

Roll Number : 23DS001

Department : Data Science

-----------------------------------

Student Details

Name : Esther

Roll Number : 23DS002

Department : Data Science
```

Result



Thus, the React program to share data between two components using **Props** was executed successfully.

Remember

✅Props are used to pass data from a Parent Component to a Child Component.

✅Props are **read-only**.

✅The same component can be reused by passing different Prop values.

Viva Questions & Answers

Q1. What are Props in React?

Answer:
Props (Properties) are used to pass data from a Parent Component to a Child Component.

Q2. Why do we use Props?

Answer:
Props allow components to share data and make components reusable.

Q3. Can a Child Component modify Props?

Answer:
No. Props are read-only and should not be modified by the Child Component.

Q4. Which component sends Props?

Answer:
The Parent Component sends Props to the Child Component.

Q5. Which component receives Props?

Answer:
The Child Component receives Props and displays or uses the data.

Perfect. Since we are rebuilding **Version 2.0**, I will also improve **3(d)**.

The biggest change is that **we won't use `alert()` anymore**. Instead, we'll display the submitted details on the page. This better demonstrates how React updates the UI using state.

Experiment 3(d)

Write a React Program to Implement Forms

Aim

To write and execute a React program to implement a **Student Registration Form** using React Forms.

Introduction & Theory

Forms are used to collect information from users such as names, email addresses, passwords, and phone numbers.

In React, forms are usually implemented using **Controlled Components**, where the values entered by the user are managed using the **`useState()` Hook**.

Whenever the user types in a form field, React updates the corresponding state variable. When the form is submitted, the stored data can be displayed, processed, or sent to a server.

Common form elements include:

- Text Box
- Password
- Email
- Radio Button
- Check Box
- Submit Button

Program Logic

The program displays a **Student Registration Form** with three input fields:

- Student Name
- Roll Number
- Department

When the user enters the details and clicks the **Submit** button, React stores the values using **State** and displays the submitted information below the form.

Algorithm

Step 1: Open the existing React project.

Step 2: Open **App.jsx**.

Step 3: Import the `useState()` Hook.

Step 4: Create state variables for the input fields.

Step 5: Design the Student Registration Form.

Step 6: Update the state using `onChange()`.

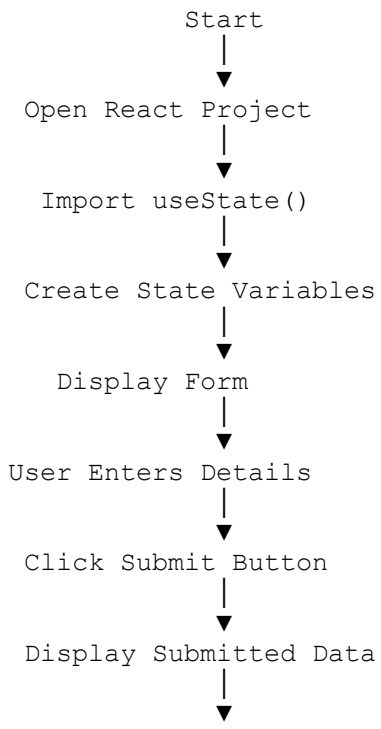
Step 7: Display the submitted details.

Step 8: Save the program.

Step 9: Execute using:

```
npm run dev
```

Flowchart



End

Program

App.jsx

```
import { useState } from "react";
import "./App.css";

function App() {

    const [name, setName] = useState("");
    const [rollNo, setRollNo] = useState("");
    const [department, setDepartment] = useState("");

    const [submitted, setSubmitted] = useState(false);

    function handleSubmit() {

        setSubmitted(true);

    }

    return (

        <div className="container">

            <div className="lab-card">

                <h1>Student Registration Form</h1>

                <label>Student Name</label>

                <input
                    type="text"
                    value={name}
                    onChange={(e) => setName(e.target.value)}
                />

                <label>Roll Number</label>

                <input
                    type="text"
                    value={rollNo}
                    onChange={(e) => setRollNo(e.target.value)}
                />

                <label>Department</label>

                <input
                    type="text"
                    value={department}
                    onChange={(e) => setDepartment(e.target.value)}
```

```
                />

                <button onClick={handleSubmit}>
                    Submit
                </button>

                {
                    submitted && (

                        <div className="result-box">

                            <h2>Submitted Details</h2>

                            <p><strong>Name :</strong> {name}</p>

                            <p><strong>Roll Number :</strong> {rollNo}</p>

                            <p><strong>Department :</strong> {department}</p>

                        </div>

                    )
                }

            </div>

        </div>

    );

}

export default App;
```

Code Explanation

1. Import Statements

```
import { useState } from "react";
import "./App.css";
```

Imports the `useState()` Hook to manage form data and the common stylesheet for Experiment 3.

---

2. Creating State Variables

```
const [name, setName] = useState("");
const [rollNo, setRollNo] = useState("");
```

```
const [department, setDepartment] = useState("");
```

These state variables store the values entered in the Student Name, Roll Number, and Department fields.

---

3. Creating the Submit Status

```
const [submitted, setSubmitted] = useState(false);
```

This state variable keeps track of whether the form has been submitted.

Initially, its value is **false**.

---

4. Updating Form Values

```
onChange={(e) => setName(e.target.value)}
```

The `onChange()` event updates the corresponding state variable whenever the user types in an input field.

The same approach is used for the Roll Number and Department fields.

---

5. Submit Function

```
function handleSubmit() {

    setSubmitted(true);

}
```

When the **Submit** button is clicked, the value of `submitted` changes to **true**.

This causes React to display the submitted information.

---

6. Conditional Rendering

```
submitted && (
```

This is an example of **Conditional Rendering**.

The submitted details are displayed only after the user clicks the **Submit** button.

---

7. React Concepts Used

This experiment demonstrates:

- `useState()`
- Controlled Components
- Event Handling
- Conditional Rendering

---

8. Export Statement

```
export default App;
```

Exports the `App` component.

Sample Output

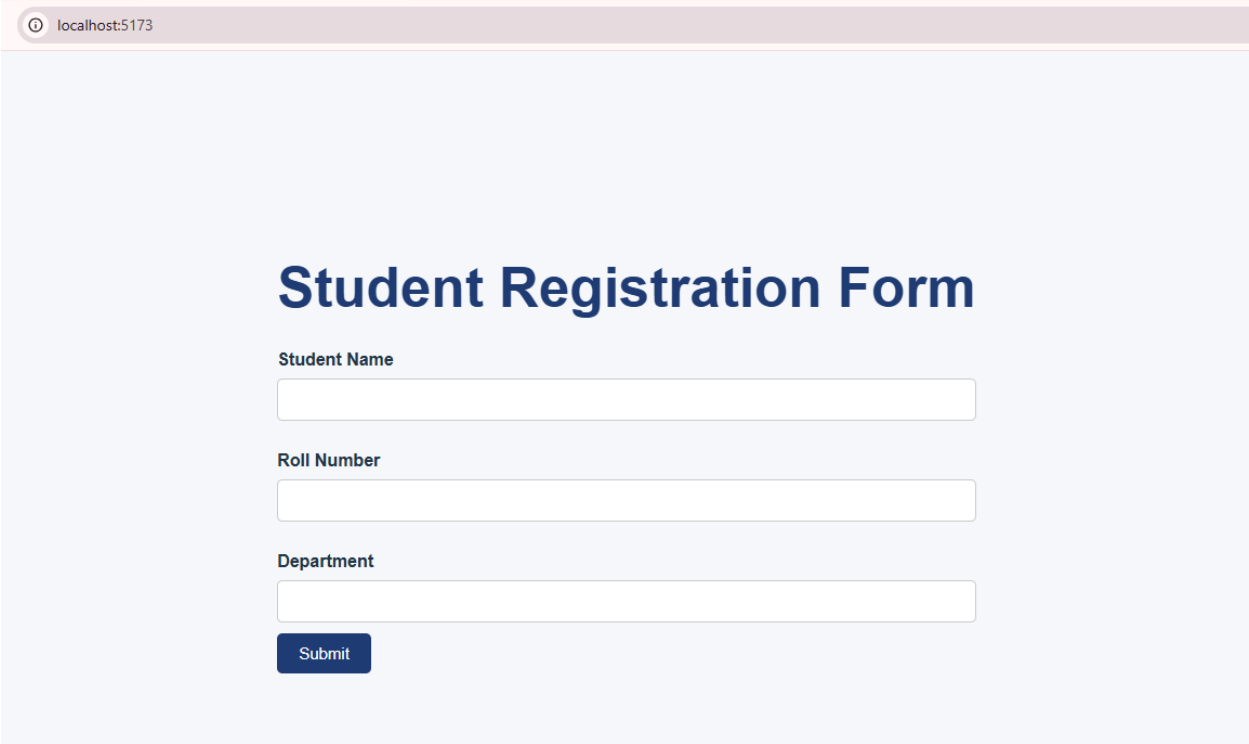
Before Submission

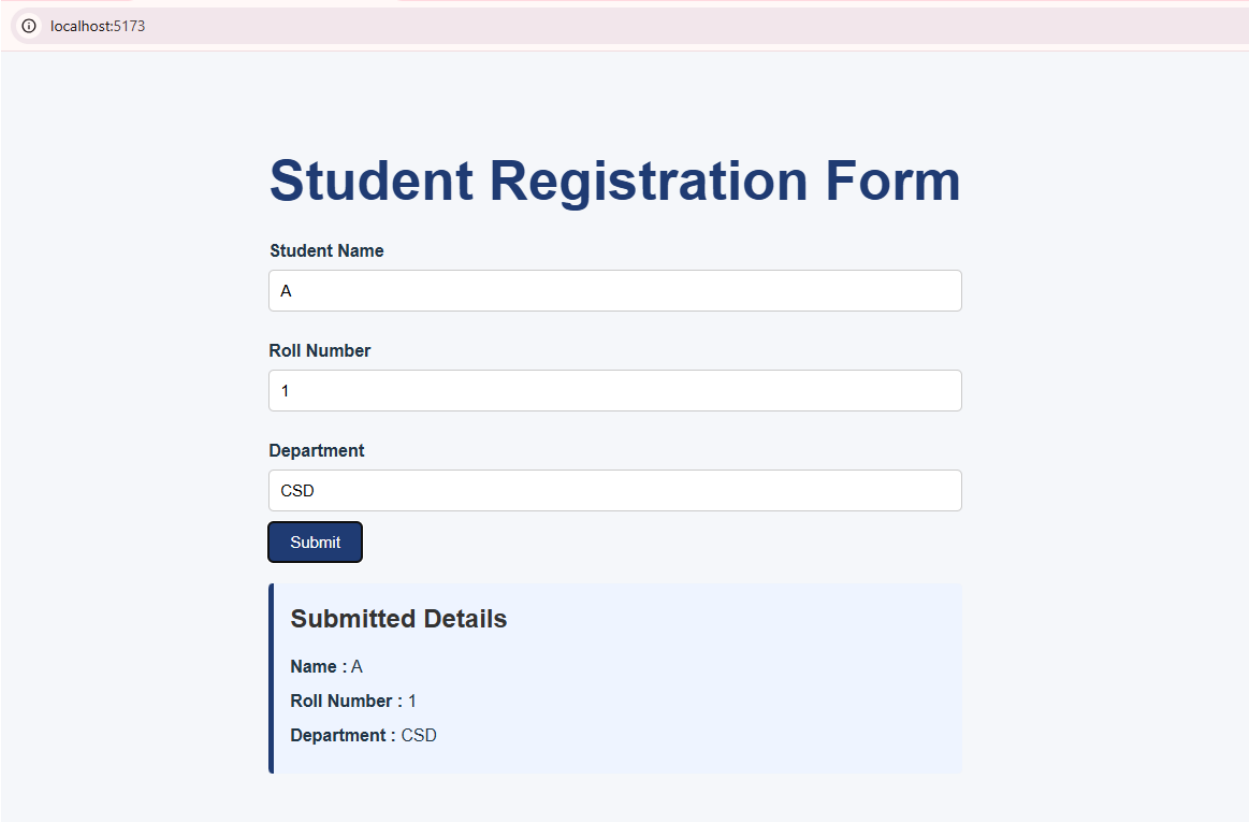
```
-----------------------------------

Student Registration Form

Student Name

_____________

Roll Number

_____________

Department

_____________

[ Submit ]

-----------------------------------
```

After Submission

```
-----------------------------------

Student Registration Form

Student Name : Daniel

Roll Number : 23DS001

Department : Data Science

Submitted Details

Name : Daniel

Roll Number : 23DS001

Department : Data Science

-----------------------------------
```

Result





Thus, the React program to implement a **Student Registration Form** using React Forms was executed successfully.

Remember

✓React Forms usually use **Controlled Components**.

✓Use `value` and `onChange()` together to manage form data.

✓React updates the user interface automatically whenever the state changes.

Viva Questions & Answers

Q1. What is a Controlled Component?

Answer:
A Controlled Component is a form element whose value is managed by React State using the `value` and `onChange()` properties.

Q2. Why is the `onChange()` event used?

Answer:
The `onChange()` event updates the state whenever the user enters or modifies data in an input field.

Q3. What is Conditional Rendering?

Answer:
Conditional Rendering is the process of displaying content only when a specified condition is true. In this experiment, the submitted details are shown only after the **Submit** button is clicked.

Perfect! This will be the **final experiment of Experiment 3**, and I want it to feel like the culmination of everything students have learned.

Compared to Version 1, I've improved it by:

- ✅Using an **array of objects** instead of simple strings.
- ✅Displaying **Name** and **Department**.
- ✅Reusing the common `App.css`.
- ✅Preparing students for React projects they'll build later.
- ✅Keeping the code beginner-friendly.

Experiment 3(e)

Write a React Program to Implement Iterative Rendering Using the `map()` Function

---

Aim

To write and execute a React program to display a list of students using the **`map()`** function.

Introduction & Theory

In many web applications, the same type of information needs to be displayed multiple times. For example, a student management system may display a list of students, while an online shopping website displays a list of products.

Instead of writing separate HTML elements for every item, React uses the **`map()`** function to generate user interface elements dynamically.

The `map()` function reads each element in an array and creates a JSX element for every item. This process is known as **Iterative Rendering**.

It helps developers write cleaner, shorter, and reusable code.

Common Applications

- Student Lists
- Employee Records
- Product Catalogues
- Hospital Patient Records
- Course Lists

Program Logic

The program creates an array of student objects.

Each object contains:

- Student ID
- Student Name
- Department

The `map()` function reads each object one by one and displays the student's details on the webpage.

Algorithm

Step 1: Open the existing React project.

Step 2: Open **App.jsx**.

Step 3: Create an array of student objects.

Step 4: Use the `map()` function to iterate through the array.

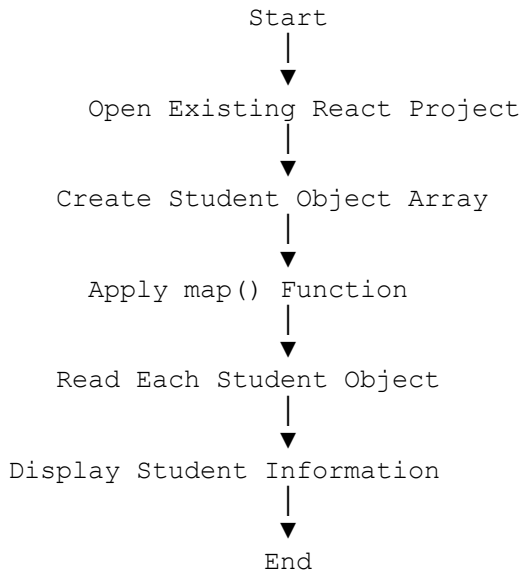
Step 5: Display the student details using JSX.

Step 6: Save the program.

Step 7: Execute the application using:

```
npm run dev
```

Flowchart



Program

App.jsx

```
import "./App.css";

function App() {

    const students = [

        {
            id: 1,
            name: "Daniel",
            department: "Data Science"
        },

        {
            id: 2,
            name: "Esther",
            department: "Data Science"
        },

        {
            id: 3,
            name: "David",
            department: "Data Science"
        }

    ];

    return (

        <div className="container">

            <div className="lab-card">

                <h1>Student List</h1>

                {

                    students.map((student) => (

                        <div key={student.id}>

                            <h3>{student.name}</h3>

                            <p><strong>Department :</strong>
{student.department}</p>

                            <hr />

                        </div>

                    ))

                }

            </div>
```

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

Code Explanation

1. Import Statement

```
import "./App.css";
```

Imports the common stylesheet used throughout Experiment 3.

---

2. Creating an Array of Objects

```
const students = [
```

Creates an array named **students**.

Each element in the array is an object containing:

- `id`
- `name`
- `department`

---

3. Using the `map()` Function

```
students.map((student) => (
```

The `map()` function reads each object in the array one by one.

The variable `student` represents the current object being processed.

---

4. Displaying Student Details

```
<h3>{student.name}</h3>

<p>{student.department}</p>
```

Displays the **Name** and **Department** of each student.

React automatically creates a separate section for every object in the array.

---

5. Using the `key` Attribute

```
key={student.id}
```

The `key` attribute uniquely identifies each rendered element.

It helps React efficiently update the user interface whenever the data changes.

---

6. React Concept Used

This experiment demonstrates **Iterative Rendering** using the **`map()`** function.

The `map()` function allows React to generate multiple JSX elements from an array without writing repetitive code.

---

7. Export Statement

```
export default App;
```

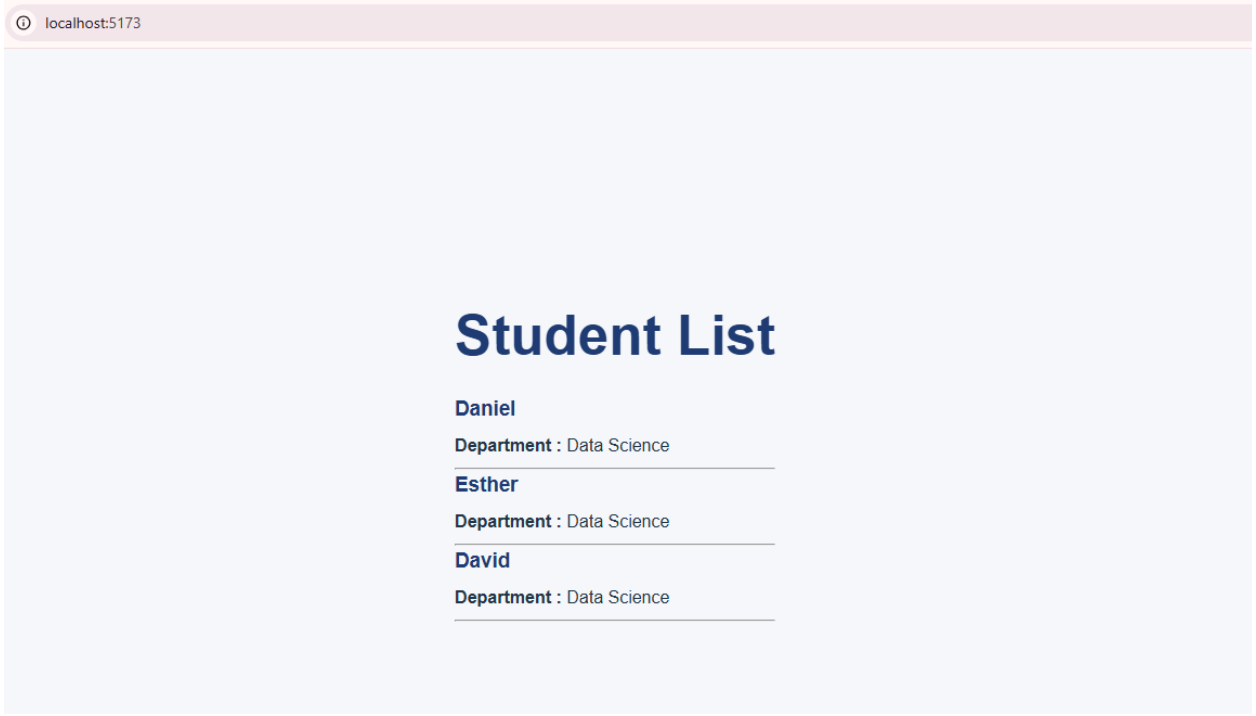
Exports the `App` component.

Sample Output

```
-----------------------------------

          Student List

-----------------------------------

Daniel
```

```
Department : Data Science

-----------------------------------

Esther

Department : Data Science

-----------------------------------

David

Department : Data Science

-----------------------------------
```

Result



Thus, the React program to implement iterative rendering using the **`map()`** function was executed successfully.

Remember

✅The `map()` function is used to display multiple items from an array.

✅Each rendered element should have a unique **`key`**.

✅Arrays of objects are commonly used in real-world React applications.

Viva Questions & Answers

Q1. What is the purpose of the `map()` function in React?

Answer:
The `map()` function is used to iterate through an array and generate JSX elements dynamically for each item.

Q2. What is Iterative Rendering?

Answer:
Iterative Rendering is the process of displaying multiple user interface elements by iterating through an array using the `map()` function.

Q3. Why do we use the `key` attribute?

Answer:
The `key` attribute uniquely identifies each rendered element and helps React efficiently update the user interface.

Q4. Why is an array of objects used instead of an array of strings?

Answer:
An array of objects can store multiple pieces of information about each item, such as ID, Name, and Department. This better represents real-world data and makes applications more flexible.

Connecting the Concepts

Throughout **Experiment 3**, you have learned the core concepts of modern React:

- `useState()` to manage changing data.
- `useEffect()` to fetch data from an API.
- **Props** to share data between components.
- **Forms** to collect user input.
- `map()` to display multiple records dynamically.

These concepts work together to build interactive React applications and provide a strong foundation for the next experiment, where you will use **Node.js and Express.js** to create backend services that React applications can communicate with.