Mernstack-React.js Assignment:

Theory

Module 1: Introduction to React.js

q.1 what is react.js? how is it different from other javaScript frameworks and libraries?

Ans:

React.js is an **open-source JavaScript library** used for building **user interfaces**, mainly **single-page applications**.
It was created by **Facebook**.

How React is different:

React.js	**Other JS Frameworks**
Library (focuses only on UI)	Framework (Angular, Vue handles routing, state, etc.)
Uses Virtual DOM	Uses Real DOM
Component-based	Often MVC or multi-structure
Faster updates	Slower because real DOM re-render
One-way data binding	Angular uses two-way binding

Question 2: Explain core principles such as Virtual DOM and Component-based architecture.

Ans:

Virtual DOM:

- React creates a **virtual copy** of the DOM in memory.
- Updates are applied first to the virtual DOM.
- React compares it with the real DOM and updates **only what changed** → this makes React **fast and efficient**.

Component-based architecture:

- UI is divided into **small reusable components**.
- Each component has its own **logic, UI, and state**.
- Easy to maintain and reuse.

Question 3: Advantages of using React.js

Ans:

Fast performance using Virtual DOM
Reusable components
Easy to learn
Strong community support
Works with mobile apps (React Native)
SEO friendly
Large ecosystem of tools and libraries

LAB EXERCISE

Task

Create new React project
Display “Hello, React!”

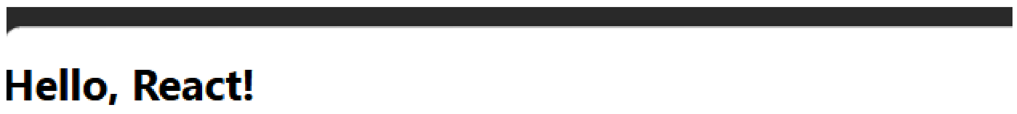
1. Create a new React project

```
npx create-react-app my-react-app

cd my-react-app

npm start
```

2. Basic Component

App.js

```
function App() {
  return <h1>Hello, React!</h1>;
}


export default App;
```



```
Hello, React!
```

Module 2. JSX

Question 1: What is JSX? Why is it used?

Ans:

JSX (JavaScript XML) is a syntax extension for JavaScript that allows us to **write HTML inside JavaScript**.

Why used?

- More readable
- Easy component creation
- React uses JSX internally for rendering

Question 2: How is JSX different from JavaScript? Can you write JS inside JSX?

Ans:

JSX ≠ JavaScript.
JSX looks like HTML but works inside JS.

Yes, we can write JavaScript inside JSX using **curly braces {}**.

Question 3: Importance of curly braces {} in JSX

Ans:

Curly braces allow:
dynamic data
variables
functions
expressions

Example:

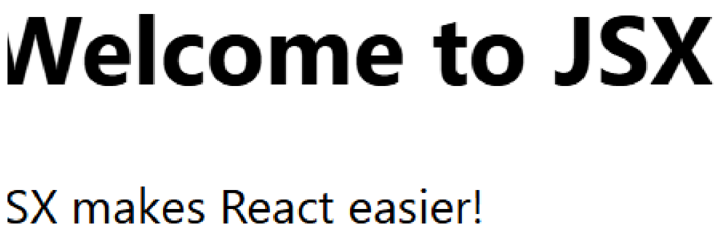
```
<p>My age is {age}</p>
```

LAB EXERCISE

Task

Create component with:
 Heading “Welcome to JSX”
Paragraph with dynamic variable

```
function JSXExample() {
  const topic = "JSX makes React easier!";
  return (
    <div>
      <h1>Welcome to JSX</h1>
      <p>{topic}</p>
    </div>
  );
}

export default JSXExample;
```



Welcome to JSX

SX makes React easier!

Module 3. Components

THEORY EXERCISE

Question 1: What are components? Functional vs Class Components:

Ans:Reusable blocks that return UI.

Functional Component	**Class Component**
Simple JavaScript function	ES6 class extending React.Component
Uses hooks for state	Uses state & lifecycle methods
Preferred in modern React	Older React pattern

Question 2: How to pass data using props?

Ans:

Props = data passed **from parent to child**.

```
<Greeting name="Devika" />
```

Question 3: Role of render() in class components

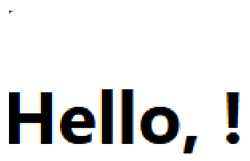
Ans:

- Required method in class components
- Must return JSX
- Called during re-rendering

LAB EXERCISE

Task 1: Functional Component with Props

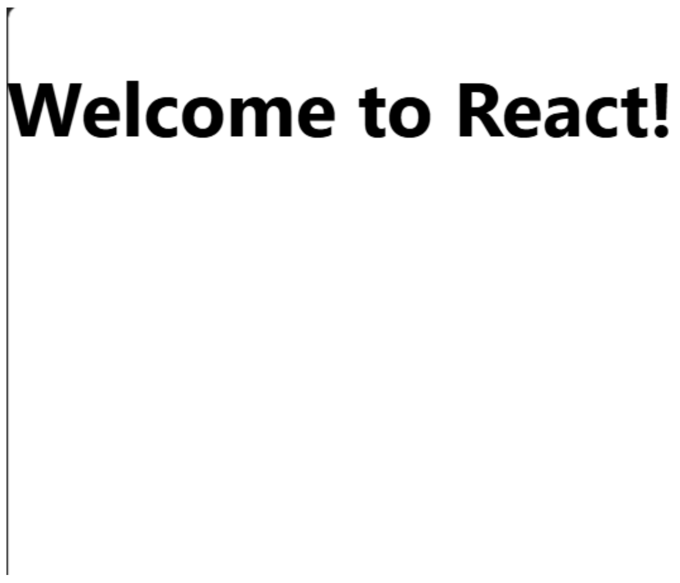
```
function Greeting({ name }) {
  return <h2>Hello, {name}!</h2>;
}

export default Greeting;
```



Hello, !

Task 2: Class Component

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

class WelcomeMessage extends Component {
  render() {
    return <h1>Welcome to React!</h1>;
  }
}

export default WelcomeMessage;
```



Welcome to React!

Module 4. Props & State

THEORY EXERCISE

Question 1: What are props? How are they different from state?

Ans:

Props	**State**
Passed from parent	Managed inside component
Read-only	Mutable
Cannot be changed	Can be changed
Used to pass data	Used to store component data

Question 2: Concept of State

Ans:

- State is data that changes over time.
- State re-renders component automatically.

Question 3: Why use this.setState()?

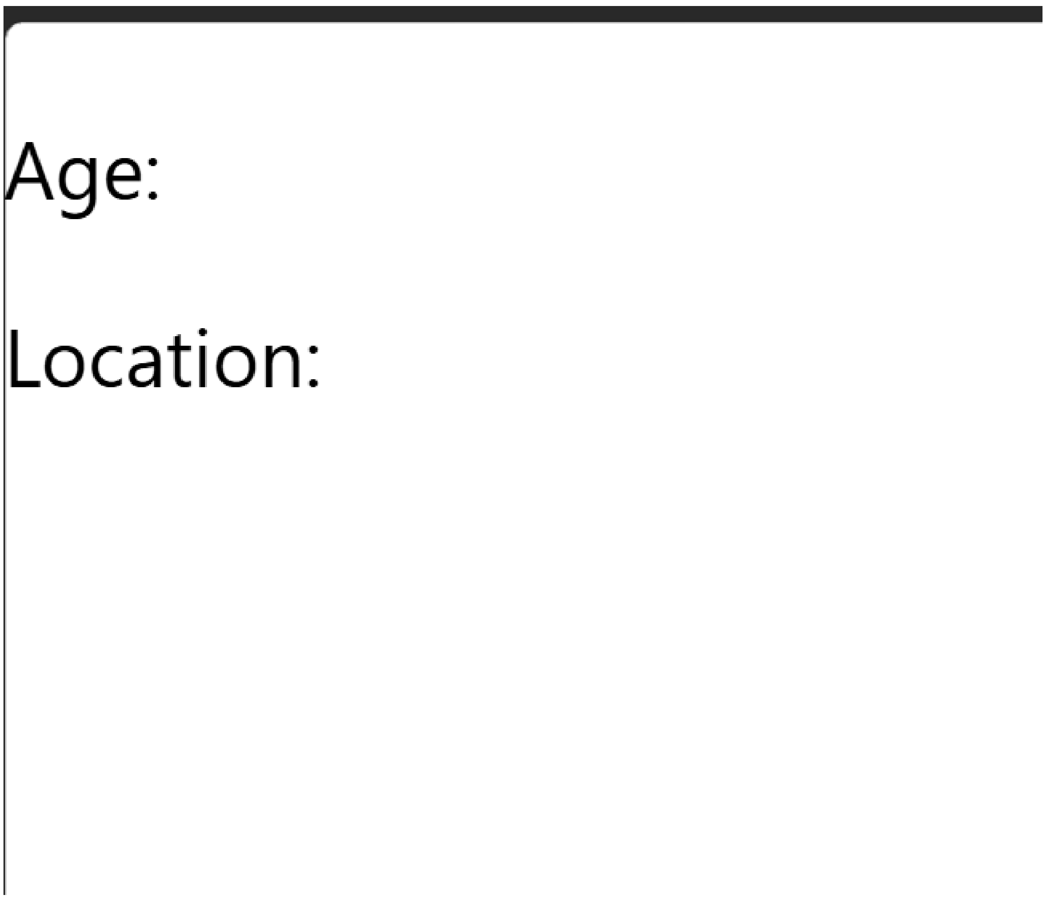
Ans:

- Used **only in class components**
- Updates state
- Triggers UI updates

LAB EXERCISE

Task 1: UserCard Component

```
function UserCard({ name, age, location }) {
  return (
    <div className="card">
      <h3>{name}</h3>
      <p>Age: {age}</p>
      <p>Location: {location}</p>
    </div>
  );
}
export default UserCard;
```

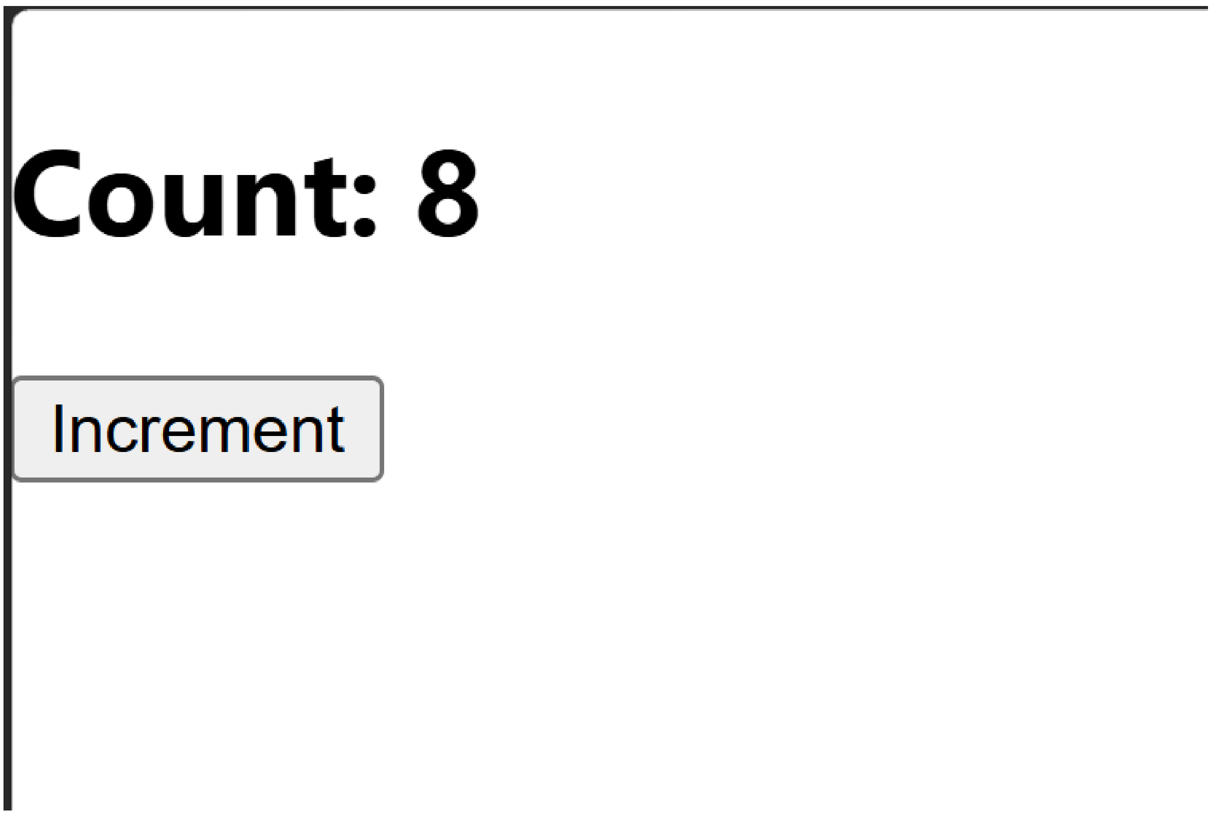

Age:

Location:

Task 2: Counter Component

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

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

  return (
    <div>
      <h2>Count: {count}</h2>
      <button onClick={() => setCount(count +
1)}>Increment</button>
```

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


```
Count: 8
```

Module 5: Event Handling

THEORY EXERCISE

Question 1: Events in React vs JavaScript

Ans:

React uses **Synthetic Events**:

- Wrapper over native events
- Cross-browser compatible

Question 2: Common Event Handlers

Ans:

Event	**Usage**
onClick	button clicks
onChange	input typing
onSubmit	form submission

Example:

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

Question 3: Why bind event handlers in class components?

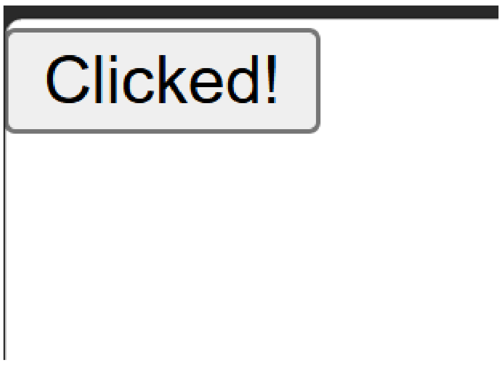
Ans:

- To access this inside functions
- Without binding, this becomes undefined

LAB EXERCISE

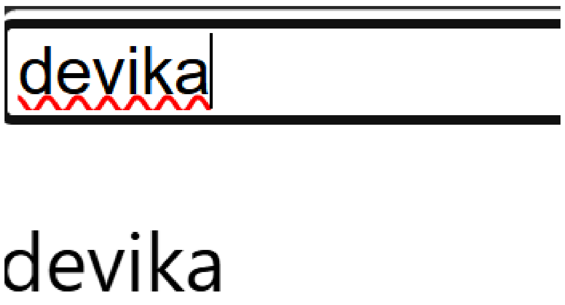
Task 1: Toggle Text

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

function ClickDemo() {
  const [text, setText] = useState("Not Clicked");

  return (
    <button onClick={() => setText("Clicked!")}>{text}</button>
  );
}
export default ClickDemo;
```



Task 2: Input Form

```
function InputDemo() {
  const [value, setValue] = useState("");

  return (
    <div>
      <input onChange={(e) => setValue(e.target.value)} />
      <p>{value}</p>
    </div>
  );
}

export default InputDemo;
```


```
devika

devika
```

Module 6: Conditional Rendering

THEORY EXERCISE

Question 1: What is conditional rendering?

Ans:

Show/hide elements based on conditions.

Question 2: Using if-else, ternary, &&

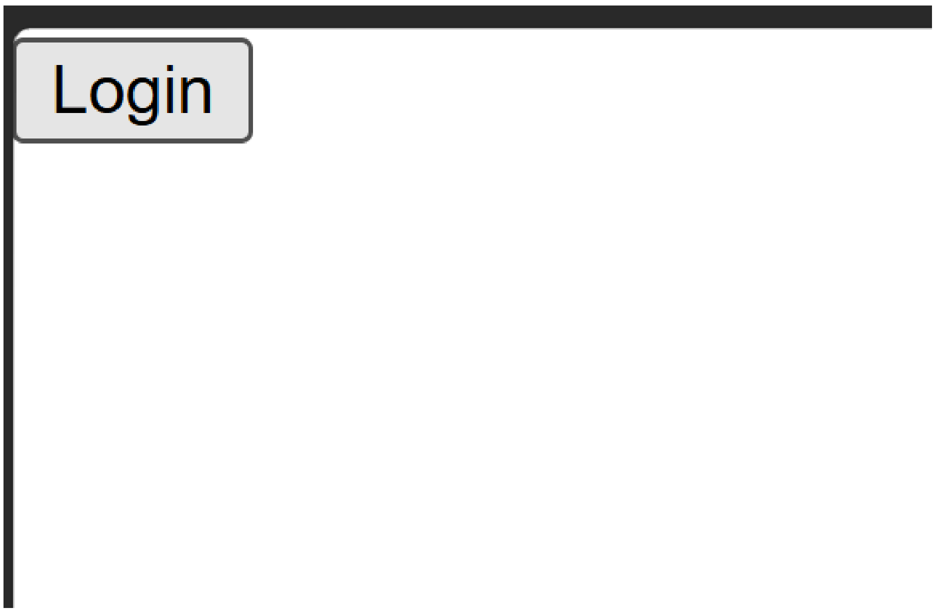
Ans:

```
condition ? <A/> : <B/>   // Ternary
condition && <A/>         // AND operator
```

LAB EXERCISE

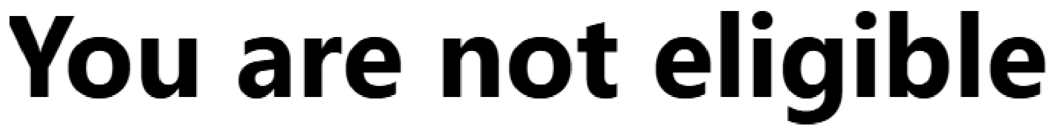
Task 1: Login / Logout

```
function LoginStatus({ isLoggedIn }) {
  return (
    <div>
      {isLoggedIn ? <button>Logout</button> : <button>Login</button>}
    </div>
  );
}
```



Task 2: Voting Eligibility

```
function Vote({ age }) {
  return (
    <h2>{age >= 18 ? "You are eligible to vote" : "You are not eligible"}</h2>
  );
}
```



```
You are not eligible
```

Module 7. Lists & Keys

THEORY EXERCISE

Question 1: How to render a list? Importance of keys?

Ans:

Use map() to render lists.
Keys help React identify changed elements.

Question 2: What are keys?

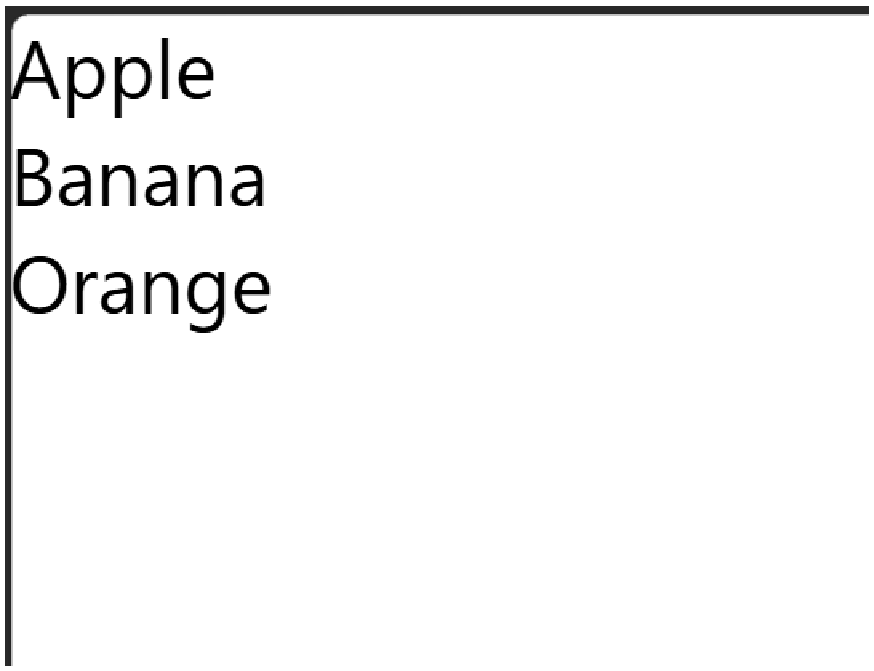
Ans:

- Unique identifier for list items
- Without keys → React re-renders incorrectly

LAB EXERCISE

Task 1: Render Fruit List

```
function FruitList() {
  const fruits = ["Apple", "Banana", "Orange"];

  return fruits.map(f => <li>{f}</li>);
}
```



```
Apple
Banana
Orange
```

Task 2: Users with Unique Keys

```
function UserList() {
  const users = [
    { id: 1, name: "Devika" },
    { id: 2, name: "Chirag" },
  ];

  return users.map(user => <p
key={user.id}>{user.name}</p>);
}
```

Devika

Chirag

Module 8. Forms in React

THEORY EXERCISE

Question 1: What are controlled components?

Ans:

Form inputs where value is managed by **React state**.

Question 2: Controlled vs Uncontrolled

Ans:

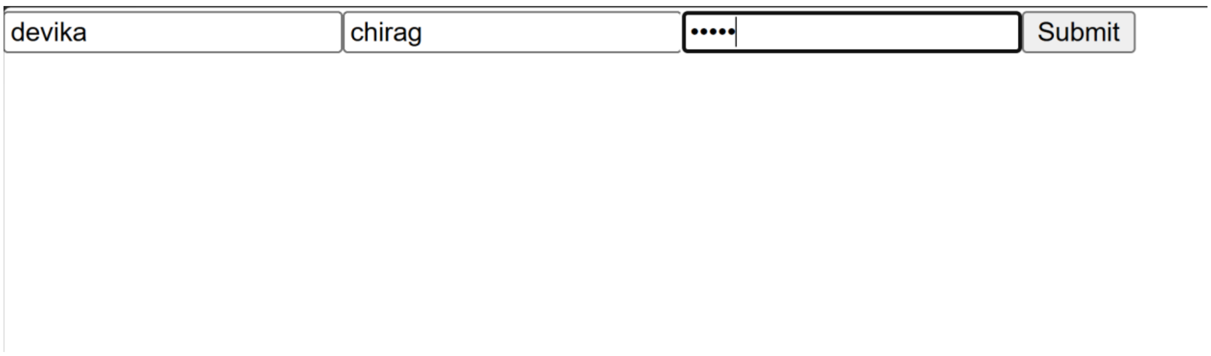
Controlled	Uncontrolled
Uses state	Uses ref
React controls value	DOM controls value

LAB EXERCISE

Task 1: Form with State

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

function FormDemo() {
  const [form, setForm] = useState({
    name: "",
    email: "",
    password: ""
  });

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(form);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input value={form.name}
onChange={(e)=>setForm({...form,name:e.target.value})} />
      <input value={form.email}
onChange={(e)=>setForm({...form,email:e.target.value})} />
```

```
        <input type="password" value={form.password}
onChange={(e)=>setForm({...form,password:e.target.value})}
/>
        <button type="submit">Submit</button>
      </form>
    );
  }

  export default FormDemo;
```



Task 2: Email Validation

```
{!form.email.includes("@") && <p>Email is invalid</p>}
```

Module 9. Lifecycle Methods:

THEORY EXERCISE

Question 1: What are lifecycle methods?

Ans:

Phases of React class component:

1. **Mounting**
2. **Updating**
3. **Unmounting**

Question 2: Explain 3 methods

Ans:

Method	**Phase**	**Purpos**
componentDidMount()	Mounting	API calls
componentDidUpdate()	Updating	respond to state/prop changes
componentWillUnmount()	Unmounting	cleanup, remove listeners

LAB EXERCISE

Task 1: Fetch API on Mount

```
class FetchDemo extends React.Component {
  state = { data: [] };

  componentDidMount() {
    fetch("https://jsonplaceholder.typicode.com/posts")
    .then(res => res.json())
    .then(data => this.setState({ data }));
  }

  render() {
    return this.state.data.map(item =>
<p>{item.title}</p>);
  }
}
```

Task 2: Log on Update & Unmount

```
class Logger extends React.Component {
  componentDidUpdate() {
    console.log("Component updated!");
  }
  componentWillUnmount() {
    console.log("Component unmounted!");
  }
  render() {
    return <h1>Logger Component</h1>;
  }
}
```