**React - Styling &Amp; Advance React**

## Conditional Rendering

## THEORY EXERCISE

### ****Question 1: What is conditional rendering in React? How can you conditionally render elements in a React component?****

**Answer:**  
Conditional rendering in React means displaying different UI elements based on certain conditions or states. It allows components to decide what to render based on variables like props, state, or logic.

You can conditionally render using:

* if-else statements
* Ternary operator (condition ? trueOutput : falseOutput)
* Logical && operator (condition && trueOutput)

### ****Question 2: Explain how if-else, ternary operators, and && (logical AND) are used in JSX for conditional rendering.****

**Answer:**

**Using if-else:**

* + Place the logic before the return statement.

if (isLoggedIn) {

  return <LogoutButton />;

} else {

  return <LoginButton />;

}

**Using ternary operator:**

* Used inside JSX.

{isLoggedIn ? <LogoutButton /> : <LoginButton />}

**Using logical AND (**&&**):**

* Renders an element **only if the condition is true**.

{age >= 18 && <p>You are eligible to vote</p>}

**LAB EXERCISE**

**Task 1: Create a component that conditionally displays a login or logout button based on the user’s login status.**

import React, { useState } from 'react';

function LoginControl() {

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

  const handleClick = () => setIsLoggedIn(!isLoggedIn);

  return (

    <div>

      <h2>{isLoggedIn ? 'Welcome Back!' : 'Please log in'}</h2>

      <button onClick={handleClick}>

        {isLoggedIn ? 'Logout' : 'Login'}

      </button>

    </div>

  );

}

export default LoginControl;

**Task 2: Implement a component that displays a message like "You are eligible to vote" if theuser is over 18, otherwise display "You are not eligible to vote."**

import React from 'react';

function VotingEligibility({ age }) {

  return (

    <div>

      <h2>Voting Eligibility Check</h2>

      {age >= 18 ? (

        <p>You are eligible to vote.</p>

      ) : (

        <p>You are not eligible to vote.</p>

      )}

    </div>

  );

}

export default VotingEligibility;

Output:

<VotingEligibility age={20} /> // shows: You are eligible to vote.

<VotingEligibility age={16} /> // shows: You are not eligible to vote.

**Hooks and keys**

**THEORY EXERCISE**

### Question 1: How do you render a list of items in React? Why is it important to use keys when rendering lists?

**Answer:**

In React, you render a list by using the JavaScript map() function to iterate over an array and return a React element for each item.

**Example:**

const fruits = ['Apple', 'Banana', 'Mango'];

const listItems = fruits.map(fruit => <li>{fruit}</li>);

You use this listItems in JSX like:

<ul>{listItems}</ul>

**Why are keys important?**

Keys help React **identify which items have changed, been added, or removed**. This improves **performance** and prevents **unnecessary re-renders**.

### Question 2: What are keys in React, and what happens if you do not provide a unique key?

**Answer:**

**Keys** are special string attributes that help React identify which elements in a list are changed, updated, or deleted.

**Good Key:**

* Must be **unique** among siblings.
* Should ideally be **stable** (not change over time).
* Common choice: a **unique ID** from the data.

**If you don’t provide a unique key:**

* React will fallback to using the index as a key.

It may lead to **incorrect UI updates**, especially during item reordering or deletion.

## LAB EXERCISE

### ****Question 1: How do you handle forms in React? Explain the concept of controlled components.****

**Answer:**  
In React, **forms are handled** by using **state** to track the values of input fields. You attach onChange handlers to inputs, and store their values in the component’s state.

A **controlled component** is a form input whose value is **controlled by React state**. The input’s value is set by a value prop, and changes are managed with onChange events.

**Forms in react**

## THEORY EXERCISE

### ****Question 1: How do you handle forms in React? Explain the concept of controlled components.****

**Answer:**  
In React, **forms are handled** by using **state** to track the values of input fields. You attach onChange handlers to inputs, and store their values in the component’s state.

A **controlled component** is a form input whose value is **controlled by React state**. The input’s value is set by a value prop, and changes are managed with onChange events.

const [name, setName] = useState('');

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

### ****Question 2: What is the difference between controlled and uncontrolled components in React?****

| **Feature** | **Controlled Component** | **Uncontrolled Component** |
| --- | --- | --- |
| Value stored in | React State | DOM (via ref) |
| React control | Full control over the input | Minimal or no control |
| onChange handler | Required | Optional |
| Use case | When validation or real-time value tracking is needed | Simpler use cases or legacy integration |
| Example | <input value={name} onChange={...} /> | <input ref={inputRef} /> |

**LAB EXERCISE**

**Task 1: Create a form with inputs for name, email, and password. Use state to control the form and display the form data when the user submits it.**

import React, { useState } from 'react';

function SignupForm() {

  const [formData, setFormData] = useState({

    name: '',

    email: '',

    password: ''

  });

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData((prev) => ({ ...prev, [name]: value }));

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    alert(`Submitted:\nName: ${formData.name}\nEmail: ${formData.email}\nPassword: ${formData.password}`);

  };

  return (

    <form onSubmit={handleSubmit}>

      <h2>Sign Up Form</h2>

      <label>

        Name:

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

      </label><br />

      <label>

        Email:

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

      </label><br />

      <label>

        Password:

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

      </label><br />

      <button type="submit">Submit</button>

    </form>

  );

}

export default SignupForm;

**Task 2: Add validation to the form created above. For example, ensure that the email input contains a valid email address.**

import React, { useState } from 'react';

function SignupFormWithValidation() {

  const [formData, setFormData] = useState({

    name: '',

    email: '',

    password: ''

  });

  const [errors, setErrors] = useState({});

  const validate = () => {

    const newErrors = {};

    if (!/\S+@\S+\.\S+/.test(formData.email)) {

      newErrors.email = 'Invalid email address';

    }

    if (formData.password.length < 6) {

      newErrors.password = 'Password must be at least 6 characters';

    }

    return newErrors;

  };

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData((prev) => ({ ...prev, [name]: value }));

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    const validationErrors = validate();

    if (Object.keys(validationErrors).length > 0) {

      setErrors(validationErrors);

    } else {

      setErrors({});

      alert(`Submitted:\nName: ${formData.name}\nEmail: ${formData.email}`);

    }

  };

  return (

    <form onSubmit={handleSubmit}>

      <h2>Sign Up Form with Validation</h2>

      <label>

        Name:

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

      </label><br />

      <label>

        Email:

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

        {errors.email && <p style={{color:'red'}}>{errors.email}</p>}

      </label><br />

      <label>

        Password:

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

        {errors.password && <p style={{color:'red'}}>{errors.password}</p>}

      </label><br />

      <button type="submit">Submit</button>

    </form>

  );

export default SignupFormWithValidation;

## Context API

## THEORY EXERCISE

### ****Question 1: What is the Context API in React? How is it used to manage global state across multiple components?****

**Answer:**  
The **Context API** in React provides a way to **share state globally** across the component tree **without passing props manually** at every level.

It's useful when:

* Multiple components need access to the same state (like theme, user authentication, language).
* You want to avoid "prop drilling" — passing data through many nested components.

To use it:

* Create a context using createContext().
* Provide context value using a Context.Provider.
* Consume context using useContext() hook.

### ****Question 2: Explain how createContext() and useContext() are used in React for sharing state.****

**Answer:**

* createContext(): Creates a new context object.

const ThemeContext = React.createContext();

* <Context.Provider>: Makes the context value available to child components.

<ThemeContext.Provider value={theme}>

  <App />

</ThemeContext.Provider>

* useContext(): Allows any child component to read the context value directly.

const theme = useContext(ThemeContext);

**LAB EXERCISE**

**Task 1: Create a simple theme toggle (light/dark mode) using the Context API. The theme state should be shared across multiple components.**

**Create Context File**: ThemeContext.js

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

export const ThemeContext = createContext();

export const ThemeProvider = ({ children }) => {

  const [theme, setTheme] = useState("light");

  const toggleTheme = () => setTheme(prev => prev === "light" ? "dark" : "light");

  return (

    <ThemeContext.Provider value={{ theme, toggleTheme }}>

      {children}

    </ThemeContext.Provider>

  );

};

**Wrap App in Provider**: index.js or App.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import { ThemeProvider } from './ThemeContext';

ReactDOM.render(

  <ThemeProvider>

    <App />

  </ThemeProvider>,

  document.getElementById('root')

);

**Use Context in Components**: App.js

import React, { useContext } from 'react';

import { ThemeContext } from './ThemeContext';

function App() {

  const { theme, toggleTheme } = useContext(ThemeContext);

  const appStyle = {

    background: theme === 'dark' ? '#333' : '#eee',

    color: theme === 'dark' ? '#fff' : '#000',

    padding: '20px',

    textAlign: 'center',

    height: '100vh'

  };

  return (

    <div style={appStyle}>

      <h1>{theme.toUpperCase()} MODE</h1>

      <button onClick={toggleTheme}>Toggle Theme</button>

    </div>

  );

}

export default App;

**Task 2: Use the Context API to create a global user authentication system. If the user is logged in, display a welcome message; otherwise, prompt them to log in.**

Create the Context → AuthContext.js

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

// 1. Create context

export const AuthContext = createContext();

// 2. Create provider

export const AuthProvider = ({ children }) => {

  const [user, setUser] = useState(null); // null means not logged in

  const login = (username) => {

    setUser({ name: username });

  };

  const logout = () => {

    setUser(null);

  };

  return (

    <AuthContext.Provider value={{ user, login, logout }}>

      {children}

    </AuthContext.Provider>

  );

};

Wrap Your App with the Provider → index.js

import React from 'react';

import ReactDOM from 'react-dom';

import App from './App';

import { AuthProvider } from './AuthContext';

ReactDOM.render(

  <AuthProvider>

    <App />

  </AuthProvider>,

  document.getElementById('root')

);

Build the UI → App.js

aimport React, { useContext, useState } from 'react';

import { AuthContext } from './AuthContext';

function App() {

  const { user, login, logout } = useContext(AuthContext);

  const [input, setInput] = useState("");

  return (

    <div style={{ textAlign: 'center', marginTop: '50px' }}>

      <h2>User Authentication Example</h2>

      {user ? (

        <>

          <p>✅ Welcome, <strong>{user.name}</strong>!</p>

          <button onClick={logout}>Logout</button>

        </>

      ) : (

        <>

          <input

            type="text"

            placeholder="Enter your name"

            value={input}

            onChange={(e) => setInput(e.target.value)}

          />

          <button onClick={() => login(input)}>Login</button>

        </>

      )}

    </div>

  );

}

export default App;