National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Web Engineering	**Course Code:**	SE3003
	Program:	BS (Software Engineering)	**Semester:**	Spring 2025
	Section:	BSE-6A	**Total Marks:**	15
	Date:		**Weight:**	
	Exam Type:	Quiz 3	**Roll No:**	
Student Name:				

Q1: What will be the value of count when the button is clicked once in the code below? Identify and fix any missing or incorrect code in the following component. **(3)**

Value of Count: 1

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

  function handleClick() {
    setCount(count + 1);  setCount(prevCount => prevCount + 1);
    setCount(count + 1);  setCount(prevCount => prevCount + 1) // For count+2
  }
  return (
    <div>
      <p>Count: {count} </p>  Display count value
      <button onClick={handleClick}>Increment</button> Add onClick event listener
    </div>
  );
}
```

Q2: How do useEffect and useMemo differ in terms of their purpose and execution in React? **(2)**

1. **Purpose:**
   useEffect: Used for performing side effects (API calls etc)
   useMemo: To optimize expensive calculations and return a memoized value
2. **Execution:**
   useEffect: runs after render + re-runs when dependencies change
   useMemo: runs during render + recalculates only if dependencies change

Q3: Create a **Login** Form Component with **username** and **password** input fields, two-way data binding using useState and a submit button that alerts the username when clicked and prevents default behaviour. **(10)**

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

function LoginForm() {
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");
  const [error, setError] = useState("");

  function handleSubmit(event) {
    event.preventDefault(); // Prevent default form behaviour (submission)

    if (!username || !password) {
      setError("Both fields are required!"); // Validation check
      return;
    }

    alert(`Logged in as: ${username}`); // or something similar
    setError(""); // Clear error after successful submission
  }

  return (
    <div>
      <h2>Login</h2>
      {error && <p style={{ color: "red" }}>{error}</p>} {/* Display error */}

      <form onSubmit={handleSubmit}>
        <input
          type="text"
          placeholder="Username"
          value={username}
          onChange={(e) => setUsername(e.target.value)}
        />
        <br />
        <input
          type="password"
          placeholder="Password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
        />
        <br />
        <button type="submit">Login</button>
      </form>
    </div>
  );
}
```

```
export default LoginForm;
```

National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Web Engineering	**Course Code:**	SE3003
	Program:	BS (Software Engineering)	**Semester:**	Spring 2025
	Section:	BSE-6B	**Total Marks:**	15
	Date:		**Weight:**	
	Exam Type:	Quiz 3	**Roll No:**	
Student Name:				

Q1: Modify the component to conditionally render either "Welcome, User!" or "Please log in." based on the loggedIn state. Ensure that clicking the button toggles the message correctly. **(3)**

```
function StatusMessage() {
  const [loggedIn, setLoggedIn] = useState(false);

  function toggleLogin() {
    setLoggedIn(prev => !prev); // Toggle state
  }

  return (
        <div>
        <p>{loggedIn ? "Welcome, User!" : "Please log in."}</p>
        <button onClick={toggleLogin}>Toggle Login</button>
        </div>
  )
}
```

Q2: What is the difference between React.memo and useMemo? In which scenario would you use each of them? **(2)**

React.memo is a higher-order component (HOC) while **useMemo** is a React Hook.
React.memo prevents unnecessary re-renders of a component by memoizing the entire component while **useMemo** memoizes the return value of a function to avoid expensive recalculations.

Scenario:

React.memo: When a functional component re-renders frequently but its props remain

unchanged
useMemo: When a function returns complex data that doesn’t need to be recalculated every render

Q3: Create a **Product** component that takes **name** and **price** as props and displays them inside a <div>. In the **App** component, define an array of products with at least two objects, each containing name and price. Render multiple Product components dynamically inside the App component. **(10)**

```
import React from "react";

// Product Component

function Product({ name, price }) {
  return (
    <div className="product">
      <h3>{name}</h3>
      <p>Price: ${price}</p>
    </div>
  );
}

// App Component

function App() {
  const products = [
    { id: 1, name: "Laptop", price: 1000 },
    { id: 2, name: "Smartphone", price: 500 },
  ];

  return (
    <div>
      <h2>Product List</h2>
      {products.map((product) => (
        <Product key={product.id} name={product.name} price={product.price} />
      ))}
    </div>
  );
}

export default App;
```

National University of Computer and Emerging Sciences, Lahore Campus

	Course Name:	Web Engineering	**Course Code:**	SE3003
	Program:	BS (Software Engineering)	**Semester:**	Spring 2025
	Section:	BSE-6C	**Total Marks:**	15
	Date:		**Weight:**	
	Exam Type:	Quiz 3	**Roll No:**	
Student Name:				

Q1: How many times will the useEffect run? Fix the code to increment count on every button click and run useEffect every time the value of count changes. **(3)**

useEffect runs once initially not infinitely due to [ ] (dependency)

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

  useEffect(() => {
    console.log("Count incremented");
  }, [count]);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increase</button>
    </div>
  );
}
```

Q2: How does useState differ from useMemo in terms of functionality and performance optimization? **(2)**

useState:

Manages component state (stores and updates values)

Triggers re-renders when the state updates
useMemo:
Memoizes expensive calculations to avoid unnecessary recalculations
Recalculates only when dependencies change

Q3: Create a **Contact** component that takes **name** and **email** as props and displays them inside a <div>. In the **App** component, define an array of two contacts, each with name and email. Inside the App component render multiple Contact components dynamically. **(10)**

```
import React from "react";

// Contact Component

function Contact({ name, email }) {
  return (
    <div className="contact">
      <h3>{name}</h3>
      <p>{email}</p>
    </div>
  );
}

// App Component

function App() {
  const contacts = [
    { id: 1, name: "John Doe", email: "john@example.com" },
    { id: 2, name: "Jane Smith", email: "jane@example.com" },
  ];

  return (
    <div>
      <h2>Contact List</h2>
      {contacts.map((contact) => (
        <Contact key={contact.id} name={contact.name} email={contact.email} />
      ))}
    </div>
  );
}

export default App;
```