**Assignment 5**

**Problem Statement 0: Basics of React.js**

**1. What is React and what problem does it solve?**

React is a JavaScript library for building **user interfaces**. It allows developers to create **reusable UI components** and efficiently update the UI when data changes using a virtual DOM.

**2. What are React components and how are they used?**

React components are **building blocks** of a React application. They are **reusable** UI elements, which can be functional or class-based. Components are used by importing and embedding them inside other components.

**3. What is JSX in React?**

JSX (JavaScript XML) is a syntax extension that allows writing HTML-like code inside JavaScript. It makes React components more readable and maintainable.

const element = <h1>Hello, React!</h1>;

**4. What are props in React and how do they differ from state?**

Props (short for "properties") are **immutable** and used to pass data from parent to child components.

State is **mutable** and is used to store component-specific data that may change over time.

function Welcome(props) {

return <h1>Hello, {props.name}!</h1>;

}

**5. What is state in React and how does it work?**

State is a **built-in object** that holds data about a component. It is managed within the component and updates cause re-renders.

import { useState } from "react";

function Counter() {

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

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

}

**6. What are React lifecycle methods, and why are they important?**

Lifecycle methods are functions in **class components** that allow control over different phases of a component’s life cycle.

* **Mounting:** componentDidMount()
* **Updating:** componentDidUpdate()
* **Unmounting:** componentWillUnmount()

In functional components, **useEffect** serves the same purpose.

**7. React Core Concepts**

**Event Handling**

React handles events similar to HTML but uses camelCase for event names.

<button onClick={() => alert("Clicked!")}>Click Me</button>

**Conditional Rendering**

We can render components conditionally.

{isLoggedIn ? <Dashboard /> : <Login />}

**Lists and Keys**

React uses **keys** to optimize list rendering.

const list = items.map((item) => <li key={item.id}>{item.name}</li>);

**Forms**

Forms in React use **controlled components**.

const [text, setText] = useState("");

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

**Hooks**

React Hooks allow functional components to use state and lifecycle features.

* useState for state management
* useEffect for side effects
* useContext for global state
* useReducer for complex state logic

**React Router**

React Router manages navigation between pages.

import { BrowserRouter, Route, Routes } from "react-router-dom";

<BrowserRouter>

<Routes>

<Route path="/" element={<Home />} />

<Route path="/about" element={<About />} />

</Routes>

</BrowserRouter>;

**State Management**

State management techniques include:

* **useState** (local state)
* **useContext** (global state)
* **Redux/Zustand/Recoil** (advanced state management)

**React Context API**

Context API allows passing data without **prop drilling**.

const ThemeContext = createContext("dark");

function App() {

return (

<ThemeContext.Provider value="light">

<ChildComponent />

</ThemeContext.Provider>

);

}

**Optimizing Performance**

* **Using React.memo** for memoization
* **Using useCallback and useMemo** to optimize function calls
* **Lazy loading and code splitting** for faster initial load times

**Problem Statement 1: Star Wars Character App**

