1. Explain advantages of React JS regarding web development.

- React JS is a **JavaScript library** for building user interfaces, mainly single-page applications.
- It uses a **component-based structure**, making code reusable and easier to maintain.
- The **Virtual DOM** feature makes rendering faster by updating only the changed parts of the UI.
- React supports **unidirectional data flow**, which improves debugging and code stability.
- It integrates well with other libraries and frameworks, giving flexibility in development.
- Developers get faster productivity with tools like **React Developer Tools** and ecosystem support.
- Companies like **Facebook, Instagram, and Netflix** use React for smooth, dynamic UIs.

2. HTML and JSX.

- **HTML (HyperText Markup Language)** is the standard language for structuring web pages.
- It defines elements like headings, paragraphs, images, and links.
- **JSX (JavaScript XML)** is an extension of JavaScript used in React.
- JSX allows writing HTML-like code inside JavaScript, making UI creation more intuitive.
- Example: <h1>Hello World</h1> can be written directly inside React code using JSX.
- JSX makes code shorter and easier to read compared to traditional React.createElement.
- It combines the **power of JavaScript** with the simplicity of HTML.

3. Explain about JSX in React.

- JSX is a **syntax extension** that lets developers write HTML-like code inside JavaScript.
- It improves readability and makes UI design more natural for web developers.
- JSX is not understood by browsers directly; tools like **Babel** compile it into normal JavaScript.
- Example: {name} allows embedding dynamic JavaScript expressions inside UI.
- JSX supports attributes, styles, and events similar to HTML.
- It ensures components are written in a **clean and structured way**.
- Without JSX, React code becomes long and hard to manage.

4. Explain functionalities of class and function component in React.

- **Class Components**:
  - Introduced earlier in React, they extend React.Component.
  - Can hold **state** and use **lifecycle methods** like componentDidMount.
  - Example: class App extends React.Component { render(){ return <h1>Hello</h1> } }.
- **Function Components**:
  - Simpler, written as normal JavaScript functions.
  - With **React Hooks (useState, useEffect)**, they can manage state and side effects.
  - Example: function App(){ return <h1>Hello</h1> }.
- Today, **function components are preferred** for being lightweight and modern.

5. Directory structure of React.

- A standard React app created with create-react-app has this structure:
  - **node_modules/** → stores all project dependencies.
  - **public/** → contains index.html and static files like images.
  - **src/** → main folder where development happens.
    - **App.js / App.jsx** → main component.
    - **index.js** → entry point, renders app to DOM.
    - **components/** → custom reusable components.
    - **assets/** → images, CSS, fonts.
  - **package.json** → manages dependencies and scripts.
- This structure keeps the project organized and scalable.

6. Uses of JavaScript variables in JSX.

- In JSX, JavaScript variables can be **embedded inside curly braces { }**.
- This allows dynamic rendering of values in the UI.
- Example: const name = "Subham"; <h1>Hello {name}</h1>.
- Variables can hold strings, numbers, arrays, objects, or even functions.
- This helps display **real-time data** in components.
- Using variables in JSX makes the UI interactive and connected with logic.
- It is the main way React connects **JavaScript logic with HTML-like syntax**.

7. DOM structure.

- **DOM (Document Object Model)** is a tree-like structure representing HTML elements.
- Each HTML tag is a **node** in the DOM tree.
- JavaScript can access and manipulate DOM elements using methods like getElementById.
- The DOM updates whenever page content changes.
- It allows event handling, styling, and dynamic page modifications.
- However, frequent DOM updates make the page slower.
- That's why React introduced the **Virtual DOM** for optimization.

8. Virtual DOM.

- The **Virtual DOM** is a lightweight copy of the real DOM maintained by React.
- When data changes, React first updates the Virtual DOM instead of the real one.
- Then, it compares the new and old versions using a process called **diffing**.
- Only the changed parts are updated in the real DOM.
- This makes updates **faster and more efficient** than direct DOM manipulation.
- It improves performance, especially in large applications.
- Virtual DOM is one of the **core reasons React is so fast**.

9. ReactJS vs Traditional JS.

- **Traditional JS**:
  - Directly manipulates the DOM using methods like getElementById.
  - Code gets complex and harder to manage in large projects.
  - No built-in structure for reusable components.

- **ReactJS**:
  - Uses Virtual DOM for **faster performance**.
  - UI is built with **reusable components**, making code clean.
  - JSX makes UI design easier inside JavaScript.
  - Supports modern features like **state, props, and hooks**.
- Overall, React is **scalable and efficient** compared to traditional JS approaches.

.

🟦 Full Stack – React (Set 2)

1. Significance of React to build SPA (Single Page Application).

- SPA means the web app loads a **single HTML page** and dynamically updates content without refreshing.
- React is ideal for SPAs because it uses a **Virtual DOM** to update only the changed parts.
- Navigation in React SPAs is smooth, as pages are updated instantly using **React Router**.
- It improves user experience by making apps fast and responsive like desktop apps.
- Data fetching can be done dynamically with APIs, avoiding full reloads.
- Example: Gmail, Facebook, and Twitter use SPA concepts with React.
- In short, React provides **speed, interactivity, and reusability**, which are the backbone of SPAs.

2. Describe module in React application.

- A **module** is a separate piece of code (file) that performs a specific function in React.
- Modules can be **JavaScript files, components, or libraries** that are imported into a project.
- They help break the project into smaller parts, making it easier to manage.
- Example: Header.js, Footer.js, and Navbar.js can be modules in one app.
- React uses import and export keywords to reuse modules across files.
- This supports **modular development**, where teams can work independently on different parts.
- Modules increase **reusability, scalability, and maintainability** of code.

3. What is props in ReactJS?

- **Props (Properties)** are used to **pass data** from one component (parent) to another (child).
- They are **read-only** and cannot be modified by the child component.
- Props make components **dynamic and reusable** with different data.
- Example: <Greeting name="Subham" /> passes name as a prop.
- Inside the component, props are accessed using props.name.
- Props can hold strings, numbers, arrays, objects, or even functions.
- They are essential for **data flow in React**.

4. Explain about Reusable component.

- A **reusable component** is a piece of UI that can be used multiple times in different places.

- Instead of writing the same code again, one component can handle different data using props.
- Example: A Button component can be reused with different labels like “Submit”, “Login”, “Cancel”.
- Reusable components reduce duplication and improve **code maintainability**.
- They make the project easier to scale, as updates need to be done in one place only.
- React’s component-based design strongly supports reusability.
- This is one reason React is popular for **large-scale applications**.

5. Describe about state.

- **State** is an object in React that stores data which can **change over time**.
- Unlike props, state is **managed within a component itself**.
- When state changes, React **re-renders** the component automatically.
- Example: A counter app uses state to store and update the count value.
- State is usually defined with useState hook in functional components.
- It makes components **interactive and dynamic** by holding user input, API data, etc.
- State is at the heart of React’s reactive programming model.

6. Explain importance about useState.

- useState is a **React Hook** used in functional components to manage state.
- It provides two values: the current state and a function to update it.
- Example: const [count, setCount] = useState(0) creates a counter state.
- It allows components to **remember values** between renders.
- Without useState, functional components would remain static and non-interactive.
- It is important for handling **forms, toggles, API responses, and user input**.
- useState made functional components as powerful as class components.

7. React state vs props.

- **Props**:
  - Passed from **parent to child**.
  - **Read-only**, cannot be changed by the component.
  - Used to make components dynamic and reusable.
- **State**:
  - Managed **inside the component** itself.
  - **Mutable**, can change with user actions or events.
  - Used to store data that affects component behavior.
- Together, state and props define how data flows in a React app.
- Example: Props = initial data, State = live data changes.

8. Props validation in React.

- Props validation ensures that **components receive correct data types**.
- It is done using the prop-types library in React.
- Example:
- Greeting.propTypes = { name: PropTypes.string.isRequired }

- It helps developers catch errors early during development.
- Validation ensures that required props are passed and in the correct format.
- This makes the app more **reliable and less buggy**.
- It is especially useful in large projects with many components.

9. Uses of state in React.

- State is used to **store and update data** dynamically in components.
- It handles data that changes with user input, like form fields or button clicks.
- State is useful for managing UI elements like **toggles, dropdowns, and modals**.
- It stores API responses and allows live updates without refreshing the page.
- Example: A login form stores username and password in state.
- State makes React apps **interactive and responsive**.
- Without state, components would only display static data.

10. React Router in Single Page Application.

- **React Router** is a library used to handle navigation in React SPAs.
- Instead of reloading the page, it **changes the URL and loads components dynamically**.
- It provides components like <BrowserRouter>, <Route>, and <Link> for navigation.
- Example: /home, /about, /contact routes can display different components.
- It keeps the app fast because only necessary components update.
- React Router also supports nested routes and redirects.
- It is essential for building **multi-page feel inside a single-page app**.