**WEEK 6**

**Superset ID : 6439908**

**Hands-on 2**

**1 Explain React components**

A React **component** is an isolated, reusable block of UI logic—think of it like a super‑charged JavaScript function that returns JSX (rendered HTML)

Components encapsulate UI behavior and appearance, and can receive inputs (props) and maintain internal state.

**2 Identify the differences between components and JavaScript functions**

|  |  |  |
| --- | --- | --- |
| **Feature** | **React Component** | **Plain JavaScript Function** |
| Naming convention | Must start with a capital letter (e.g. MyComponent) | No such restriction |
| Props & lifecycle | Accepts props, managed by React, lifecycle-aware | Regular functions don’t have React lifecycle |
| Invocation | React itself calls/instantiates the component | You manually invoke it like any JS function |
| Return value | Returns JSX to be rendered into DOM | May return anything; not rendered by React |

**3 Identify the types of components**

React supports two main component types:

**Class Components** – ES6 classes extending React.Component

**Function Components** – JavaScript functions (or arrow functions) returning JSX

**Server Components** – Function components that run only on the server (used with frameworks like Next.js)

**4 Explain class component**

Defined by extending React.Component,

e.g.:

class MyComponent extends React.Component {

render() {

return <div>Hello, {this.props.name}</div>;

}

}

**Benefits:**

Can maintain internal state via this.state and update with this.setState()

Full lifecycle method support: componentDidMount, componentDidUpdate, componentWillUnmount, etc.

**5 Explain function component**

Defined simply as a function:

function Greeting(props) {

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

}

With React 16.8+, you can use **Hooks** like useState, useEffect to handle state and lifecycle within functional components

Advantages:

Cleaner syntax, less boilerplate

No this, easier to understand and test

Often more performant and optimized (smaller memory overhead)

**6 Define component constructor**

The **constructor** is a special method in class components, called before mounting:

constructor(props) {

super(props);

this.state = { count: 0 };

this.handleClick = this.handleClick.bind(this);

}

Purpose:

Initialize this.state

Bind instance methods to this (if not using arrow class fields)

**7 Define render() function**

Every class component **must** implement a render() method.

It returns JSX or null, representing what should be rendered to the DOM:

render() {

return <div>Count is {this.state.count}</div>;

}

React calls render() on mount and whenever props or state change (subject to shouldComponentUpdate)

Functional components don’t need a render() method: the function itself is treated as the render stage

**OUTPUT:**



