**Exercise 2**

### Explain React components

**React Components** are the fundamental building blocks of any React application. Think of them as independent, reusable pieces of UI (User Interface). Just as a house is built from bricks, a React application is built from components.

Each component is a self-contained unit that encapsulates:

* **Logic:** The JavaScript code that defines its behavior (e.g., what happens when a button is clicked).
* **State:** Data that can change over time within the component, causing it to re-render.
* **Markup (UI):** The JSX (HTML-like syntax) that defines what the component looks like.

**Key characteristics of React Components:**

1. **Reusability:** Once you create a component, you can use it multiple times throughout your application, or even in different projects. For example, a Button component can be used for various actions, each with different text or styles.
2. **Modularity:** Components allow you to break down a complex UI into smaller, manageable, and isolated pieces. This makes development, debugging, and maintenance much easier.
3. **Encapsulation:** Each component manages its own state and logic, making it independent of other components unless explicitly connected via props.
4. **Composition:** You build complex UIs by composing (nesting) smaller components together. A Header component might contain a Logo component and a Navigation component.
5. **Declarative:** You describe *what* you want your component to render given its current state and props, and React efficiently handles updating the actual DOM to match that description.

**Example Analogy:**

Imagine building a car.

* The entire car is your **React application**.
* Individual parts like the Engine, Wheel, Seat, and SteeringWheel are your **React components**.
* Each part (component) has its own specific function and design.
* You assemble these parts (compose components) to create the complete car (application).
* If you need another car, you can reuse the Wheel component without rebuilding it from scratch.

### Identify the differences between components and JavaScript functions

While React components (especially functional components) look like JavaScript functions, there are fundamental differences in their purpose and how React treats them:

|  |  |  |
| --- | --- | --- |
| Feature | Standard JavaScript Function | React Component (especially Functional Component) |
| **Purpose** | Performs a specific task or calculation; returns a value. | Renders a piece of UI; returns JSX (React elements). |
| **Return Value** | Can return any JavaScript data type (string, number, object, etc.). | Must return JSX (a React element or null). |
| **Naming Convention** | Typically camelCase (e.g., calculateSum). | Typically PascalCase (e.g., MyComponent, Button). This is crucial for React to differentiate them from HTML tags. |
| **Reactivity/Rendering** | Executes once when called; no inherent mechanism for re-execution based on data changes. | Re-renders automatically when its state or props change. This is the core of React's dynamic UI. |
| **DOM Interaction** | Directly interacts with the DOM if written to (e.g., document.createElement). | **Does not directly interact with the DOM.** Returns JSX, and React's Virtual DOM handles efficient updates to the actual DOM. |
| **Lifecycle Methods/Hooks** | No built-in lifecycle management. | Has lifecycle (for class components) or Hooks (for functional components) to manage behavior at different stages (e.g., useEffect for side effects). |
| **State Management** | No built-in state management. | Can manage internal state (using useState Hook or this.state in class components). |
| **this context** | this context depends on how the function is called. | this context is typically bound correctly in class components; not relevant for functional components. |

**In essence:** A JavaScript function is a general-purpose block of code. A React Component is a *specialized* JavaScript function (or class) designed specifically for describing and rendering UI, with built-in mechanisms for reactivity, state management, and lifecycle handling provided by the React library.

### Identify the types of components

Historically, React applications primarily used two main types of components:

1. **Class Components**
2. **Function Components**

With the introduction of Hooks in React 16.8, **Function Components** have become the preferred way to write new React components due to their simplicity and ability to handle state and side effects. However, you will still encounter **Class Components** in older codebases.

### Explain class component

**Class Components** are JavaScript ES6 classes that extend from React.Component. They are the traditional way to define components in React and were necessary for managing state and lifecycle methods before the introduction of Hooks.

**Key characteristics of Class Components:**

* **extends React.Component:** They must inherit from React.Component to be recognized as React components.
* **render() Method:** They **must** contain a render() method, which is responsible for returning the JSX that describes the component's UI.
* **this.state:** They can have internal, mutable state managed using this.state and updated with this.setState().
* **Lifecycle Methods:** They have access to various lifecycle methods (e.g., componentDidMount, componentDidUpdate, componentWillUnmount) that allow you to execute code at specific points in a component's life (when it's mounted, updated, or unmounted).
* **this.props:** Props passed to the component are available via this.props.

**Example of a Class Component:**

JavaScript

import React from 'react';  
  
class Greeting extends React.Component {  
 // 1. Define state (optional)  
 constructor(props) {  
 super(props); // Must call super(props)  
 this.state = {  
 message: 'Hello',  
 count: 0  
 };  
 }  
  
 // Example of a custom method  
 incrementCount = () => {  
 this.setState(prevState => ({  
 count: prevState.count + 1  
 }));  
 };  
  
 // 2. The render() method is mandatory  
 render() {  
 // Access props using this.props  
 // Access state using this.state  
 return (  
 <div>  
 <h1>{this.state.message}, {this.props.name}!</h1>  
 <p>You clicked {this.state.count} times.</p>  
 <button onClick={this.incrementCount}>Click Me</button>  
 </div>  
 );  
 }  
}  
  
export default Greeting;

**Pros of Class Components (historical context):**

* Were the only way to manage state and lifecycle before Hooks.
* Provided a clear structure for complex components.

**Cons of Class Components (compared to functional components with Hooks):**

* More verbose and boilerplate code.
* this binding issues can be confusing.
* Logic reuse across components can be harder without render props or higher-order components.

### Explain function component

**Function Components** are JavaScript functions that accept props as an argument and return React elements (JSX). Before Hooks, they were often called "stateless functional components" because they couldn't manage their own state or use lifecycle methods.

However, with the introduction of **Hooks** in React 16.8, Function Components can now do everything a Class Component can, including managing state and performing side effects. This has made them the preferred way to write components in modern React development due to their simplicity and cleaner syntax.

**Key characteristics of Function Components:**

* **Plain JavaScript Function:** They are just standard JavaScript functions.
* **Accept props:** They receive an object of props as their first argument.
* **Return JSX:** They must return JSX, which describes the component's UI.
* **Hooks:** They utilize Hooks (e.g., useState, useEffect, useContext) to manage state, perform side effects, and access other React features.
* **No this keyword:** Since they are functions, there's no this context to worry about, making them less prone to this binding issues.

**Example of a Function Component:**

JavaScript

import React, { useState, useEffect } from 'react'; // Import Hooks  
  
function Greeting(props) {  
 // 1. Use the useState Hook for state management  
 const [count, setCount] = useState(0);  
 const [message, setMessage] = useState('Hello');  
  
 // 2. Use the useEffect Hook for side effects (like componentDidMount, componentDidUpdate)  
 useEffect(() => {  
 console.log('Component rendered or count changed!');  
 // This runs after every render where 'count' has changed.  
 // Cleanup function can be returned if needed (like componentWillUnmount)  
 return () => {  
 console.log('Cleanup before next effect or unmount');  
 };  
 }, [count]); // Dependency array: effect runs only when 'count' changes  
  
 const incrementCount = () => {  
 setCount(prevCount => prevCount + 1);  
 };  
  
 // 3. Return JSX  
 return (  
 <div>  
 <h1>{message}, {props.name}!</h1> {/\* Access props directly \*/}  
 <p>You clicked {count} times.</p>  
 <button onClick={incrementCount}>Click Me</button>  
 </div>  
 );  
}  
  
export default Greeting;

**Pros of Function Components (especially with Hooks):**

* **Simpler and less verbose:** Less boilerplate code compared to class components.
* **Easier to read and write:** Cleaner syntax.
* **Better logic reuse:** Hooks allow for easier extraction and reuse of stateful logic.
* **No this binding issues:** Eliminates a common source of confusion in React.
* **Improved performance (potential):** Hooks can sometimes lead to more optimized renders.

### Define component constructor

The **component constructor** is a special method in **Class Components** that is called when a component instance is created (mounted) in the DOM. It's the first method to be executed in the lifecycle of a class component.

**Key points about the constructor:**

* **Purpose:** Primarily used for two things:
  1. **Initializing local state:** Setting up the initial this.state of the component.
  2. **Binding event handler methods:** Binding this context for custom methods that will be used as event handlers (though arrow functions for class methods often negate the need for explicit binding).
* **super(props):** When defining a constructor for a React component, you **must** call super(props) as the first statement within it. This calls the constructor of the parent React.Component class, ensuring that this.props is correctly initialized before your constructor code runs.
* **Receives props:** The constructor receives props as an argument, allowing you to initialize state based on initial props.
* **Should not cause side effects:** The constructor should be a pure function that only initializes state and binds methods. Avoid performing side effects (like data fetching or subscriptions) directly in the constructor; use lifecycle methods like componentDidMount for that.

**Example:**

JavaScript

import React from 'react';  
  
class MyClassComponent extends React.Component {  
 constructor(props) {  
 // 1. Always call super(props) first!  
 super(props);  
  
 // 2. Initialize state  
 this.state = {  
 value: props.initialValue || 'default', // Initializing state from props  
 isLoading: true  
 };  
  
 // 3. Bind event handlers (less common with arrow functions for methods)  
 // this.handleClick = this.handleClick.bind(this);  
 }  
  
 // ... other methods and render()  
 render() {  
 return <div>{this.state.value}</div>;  
 }  
}

**Note:** Function components **do not have constructors**. State initialization in function components is handled using the useState Hook.

### Define render() function

The render() function is a mandatory method in **Class Components** in React. It is the heart of a React component, as its sole responsibility is to describe the UI that the component should display based on its current props and state.

**Key aspects of the render() function:**

* **Mandatory for Class Components:** Every class component **must** have a render() method.
* **Returns JSX:** The render() method must return a React element (typically JSX), which React then uses to create or update the actual DOM. It can also return null, booleans, or strings if you don't want to render anything or just render text.
* **Pure Function (mostly):** The render() function should ideally be a **pure function**. This means:
  + It should not modify the component's state (use this.setState() only in event handlers or lifecycle methods like componentDidMount).
  + It should return the same output given the same inputs (props and state).
  + It should not perform side effects (like data fetching, direct DOM manipulation, or setting up subscriptions). Side effects should be handled in lifecycle methods (for class components) or the useEffect Hook (for function components).
* **Called by React:** React calls the render() method whenever the component's props or state change. This is how React determines what needs to be updated in the UI.

**Example:**

JavaScript

import React from 'react';  
  
class Product extends React.Component {  
 constructor(props) {  
 super(props);  
 this.state = {  
 quantity: 1  
 };  
 }  
  
 // This is a custom method, not render()  
 increaseQuantity = () => {  
 this.setState(prevState => ({  
 quantity: prevState.quantity + 1  
 }));  
 };  
  
 // The mandatory render() method  
 render() {  
 // Access props and state here to determine what to render  
 const { name, price } = this.props; // Destructuring props  
 const { quantity } = this.state; // Destructuring state  
  
 return (  
 <div className="product-card">  
 <h2>{name}</h2>  
 <p>Price: ${price}</p>  
 <p>Quantity: {quantity}</p>  
 <button onClick={this.increaseQuantity}>Add One</button>  
 <p>Total: ${price \* quantity}</p>  
 </div>  
 );  
 }  
}

**Note:** Function components don't have a separate render() method. The function body itself acts as the

render logic, directly returning JSX.

**Home.js**

import React,{Component} from "react";

class Home extends Component {

    render() {

        return (

            <div className="container">

                <h1>Welcome to the Home Page</h1>

            </div>

        );

    }

}

export default Home;

**About.js:**

import React,{Component} from "react";

class About extends Component {

    render() {

        return (

            <div className="container">

                <h1>Welcome to the About Page</h1>

            </div>

        );

    }

}

export default About;

**Contact.js**

import React,{Component} from "react";

class Contact extends Component {

    render() {

        return (

            <div className="container">

                <h1>Welcome to the Contact Page</h1>

            </div>

        );

    }

}

export default Contact;

**App.js:**

import logo from './logo.svg';

import './App.css';

import Contact from './Contact';

import Home from './Home';

import About from './About';

function App() {

  return (

    <div className="App">

    <Home/>

    <About/>

    <Contact/>

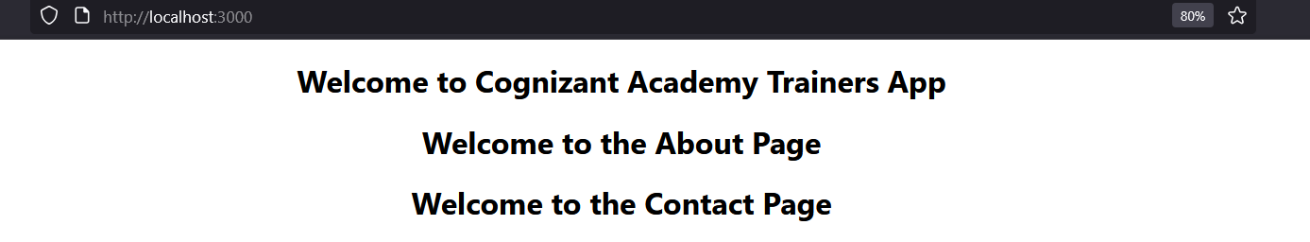
    </div>

  );

}

export default App;

**Output:**

****