**Question 1: How are events handled in React compared to vanilla JavaScript? Explain the concept of synthetic events.**

**Event Handling in React vs. Vanilla JavaScript**

1. **Event Binding:**
   * In **vanilla JavaScript**, events are added using methods like addEventListener() directly on DOM elements.
   * In **React**, event handlers are passed as props (e.g., onClick={handleClick}) to JSX elements.
2. **Event Naming:**
   * In vanilla JavaScript, event names are lowercase (e.g., onclick, onmouseover).
   * In React, event names use camelCase (e.g., onClick, onMouseOver).
3. **Event Handling:**
   * In vanilla JavaScript, you usually call event.preventDefault() manually in an event listener.
   * In React, events are wrapped in a **Synthetic Event** system, but you still call event.preventDefault() as usual.
4. **Event Binding in Classes:**
   * In vanilla JavaScript, binding is not an issue when using regular functions.
   * In React class components, event handlers need to be explicitly bound (this.handleClick = this.handleClick.bind(this);) or defined as arrow functions to preserve this context.

**Synthetic Events in React**

* **Definition:**  
  Synthetic events are React’s wrapper around native browser events. They provide a unified API across different browsers to ensure consistency.
* **Benefits of Synthetic Events:**
  + Works across all browsers.
  + Improves performance through **event pooling** (events are reused to reduce memory consumption).
  + Provides the same event properties as native events (e.g., event.target, event.type).
* **Example of Synthetic Events:**

function Button() {

function handleClick(event) {

console.log("Button clicked!", event.type); // Synthetic event

}

return <button onClick={handleClick}>Click Me</button>;

}

**Question 2: What are some common event handlers in React.js? Provide examples of onClick, onChange, and onSubmit.**

In React.js, event handlers are used to handle user interactions like clicks, typing in a form, submitting a form, etc. React uses a camelCase naming convention for events (e.g., onClick, onChange, onSubmit), and you pass a function as the event handler.

**Here are some common event handlers with examples:**

**1. onClick**

Used to handle click events on buttons or other clickable elements.

import React from 'react';

function ClickExample() {

const handleClick = () => {

alert('Button was clicked!');

};

return (

<button onClick={handleClick}>

Click Me

</button>

);

}

export default ClickExample;

**2. onChange**

Used to handle changes in input fields (text, checkbox, select, etc.).

import React, { useState } from 'react';

function ChangeExample() {

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

const handleChange = (event) => {

setText(event.target.value);

};

return (

<div>

<input type="text" value={text} onChange={handleChange} />

<p>You typed: {text}</p>

</div>

);

}

export default ChangeExample;

**3. onSubmit**

Used to handle form submission.

import React, { useState } from 'react';

function SubmitExample() {

const [inputValue, setInputValue] = useState('');

const handleSubmit = (event) => {

event.preventDefault(); // Prevents page reload

alert(`Form submitted with input: ${inputValue}`);

};

return (

<form onSubmit={handleSubmit}>

<input

type="text"

value={inputValue}

onChange={(e) => setInputValue(e.target.value)}

/>

<button type="submit">Submit</button>

</form>

);

}

export default SubmitExample;

**Question 3: Why do you need to bind event handlers in class components?**

Great question! In React class components, you often need to bind event handlers to ensure that the keyword this inside the handler refers to the correct component instance.

**🔧 Why Binding is Necessary**

In JavaScript, how this behaves depends on how a function is called, not where it's defined. When you pass a class method like this.handleClick directly to an event handler, this might become undefined or refer to the wrong context when the function is invoked.

So, without binding, this inside your method won't point to the class instance, and you'll get errors like:

TypeError: Cannot read property 'setState' of undefined

**✅ Example Without Binding (Will Throw Error)**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = { count: 0 };

}

handleClick() {

this.setState({ count: this.state.count + 1 }); // ❌ `this` is undefined here

}

render() {

return <button onClick={this.handleClick}>Click Me</button>;

}

}

**✅ Correct Ways to Bind**

**1. Bind in the Constructor**

constructor(props) {

super(props);

this.state = { count: 0 };

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

}

**2. Use Arrow Function as Class Property (Modern Syntax)**

handleClick = () => {

this.setState({ count: this.state.count + 1 });

}

**3. Use Arrow Function in JSX (not recommended for performance)**

<button onClick={() => this.handleClick()}>Click Me</button>