**Explain React Events**

**React events** are actions that can be triggered as a result of user actions or system-generated events in React applications. React uses its own event system built on top of native DOM events.

**Key Characteristics:**

* **Cross-browser compatibility**: React normalizes events across different browsers
* **Synthetic event system**: React wraps native events in SyntheticEvent objects
* **Event delegation**: React uses a single event listener on the document root
* **Camel case naming**: Event names use camelCase convention (e.g., onClick, onChange)

**Common React Events:**

* **Mouse events**: onClick, onDoubleClick, onMouseOver, onMouseOut
* **Keyboard events**: onKeyDown, onKeyUp, onKeyPress
* **Form events**: onChange, onSubmit, onFocus, onBlur
* **Touch events**: onTouchStart, onTouchMove, onTouchEnd

**Basic Example:**

const Button = () => {  
 const handleClick = () => {  
 alert('Button clicked!');  
 };  
  
 return <button onClick={handleClick}>Click Me</button>;  
};

**Event with Parameters:**

const ListItem = ({ item }) => {  
 const handleClick = (itemId) => {  
 console.log(`Item ${itemId} clicked`);  
 };  
  
 return (  
 <li onClick={() => handleClick(item.id)}>  
 {item.name}  
 </li>  
 );  
};

**Explain About Event Handlers**

**Event handlers** are functions that are executed in response to specific events in React components. They define what should happen when an event occurs.

**Types of Event Handlers:**

**1. Inline Event Handlers:**

const Component = () => {  
 return (  
 <button onClick={() => alert('Clicked!')}>  
 Click Me  
 </button>  
 );  
};

**2. Function Declaration Event Handlers:**

const Component = () => {  
 function handleClick() {  
 alert('Button clicked!');  
 }  
  
 return <button onClick={handleClick}>Click Me</button>;  
};

**3. Arrow Function Event Handlers:**

const Component = () => {  
 const handleClick = () => {  
 alert('Button clicked!');  
 };  
  
 return <button onClick={handleClick}>Click Me</button>;  
};

**4. Class Component Event Handlers:**

class Component extends React.Component {  
 handleClick = () => {  
 alert('Button clicked!');  
 }  
  
 render() {  
 return <button onClick={this.handleClick}>Click Me</button>;  
 }  
}

**Event Handler with State:**

const Counter = () => {  
 const [count, setCount] = useState(0);  
  
 const handleIncrement = () => {  
 setCount(count + 1);  
 };  
  
 const handleDecrement = () => {  
 setCount(count - 1);  
 };  
  
 return (  
 <div>  
 <p>Count: {count}</p>  
 <button onClick={handleIncrement}>+</button>  
 <button onClick={handleDecrement}>-</button>  
 </div>  
 );  
};

**Form Event Handlers:**

const Form = () => {  
 const [inputValue, setInputValue] = useState('');  
  
 const handleChange = (event) => {  
 setInputValue(event.target.value);  
 };  
  
 const handleSubmit = (event) => {  
 event.preventDefault();  
 console.log('Form submitted with:', inputValue);  
 };  
  
 return (  
 <form onSubmit={handleSubmit}>  
 <input   
 type="text"   
 value={inputValue}  
 onChange={handleChange}  
 />  
 <button type="submit">Submit</button>  
 </form>  
 );  
};

**Define Synthetic Event**

**Synthetic Events** are React's wrapper around native DOM events. They provide a consistent API across different browsers and normalize event behavior.

**Key Features:**

* **Cross-browser compatibility**: Same API works across all browsers
* **Event pooling**: React reuses event objects for performance (React 16 and below)
* **Native event access**: Can access original event via nativeEvent property
* **Prevents default**: Use preventDefault() method
* **Stops propagation**: Use stopPropagation() method

**SyntheticEvent Properties:**

* type: Event type (e.g., 'click', 'change')
* target: Element that triggered the event
* currentTarget: Element that the event handler is attached to
* preventDefault(): Prevents default browser behavior
* stopPropagation(): Stops event bubbling
* nativeEvent: Access to original DOM event

**Basic Example:**

const Component = () => {  
 const handleClick = (event) => {  
 console.log('Event type:', event.type); // 'click'  
 console.log('Target element:', event.target);  
 console.log('Current target:', event.currentTarget);  
   
 // Prevent default behavior  
 event.preventDefault();  
   
 // Stop event propagation  
 event.stopPropagation();  
 };  
  
 return <button onClick={handleClick}>Click Me</button>;  
};

**Form SyntheticEvent Example:**

const Input = () => {  
 const handleChange = (event) => {  
 console.log('Input value:', event.target.value);  
 console.log('Event type:', event.type); // 'change'  
 };  
  
 const handleKeyDown = (event) => {  
 if (event.key === 'Enter') {  
 console.log('Enter key pressed');  
 event.preventDefault();  
 }  
 };  
  
 return (  
 <input   
 type="text"  
 onChange={handleChange}  
 onKeyDown={handleKeyDown}  
 placeholder="Type here..."  
 />  
 );  
};

**Accessing Native Event:**

const Component = () => {  
 const handleClick = (event) => {  
 // SyntheticEvent properties  
 console.log('Synthetic event:', event.type);  
   
 // Native DOM event  
 console.log('Native event:', event.nativeEvent);  
 console.log('Native event type:', event.nativeEvent.type);  
 };  
  
 return <button onClick={handleClick}>Click Me</button>;  
};

**Identify React Event Naming Convention**

React follows specific naming conventions for events that differ from standard HTML event attributes.

**Naming Rules:**

* **CamelCase format**: All event names use camelCase
* **"on" prefix**: All events start with "on"
* **Descriptive names**: Event names clearly describe the action

**Common Event Name Conversions:**

|  |  |  |
| --- | --- | --- |
| **HTML Event** | **React Event** | **Description** |
| onclick | onClick | Mouse click event |
| onchange | onChange | Input value change |
| onsubmit | onSubmit | Form submission |
| onmouseover | onMouseOver | Mouse hover enter |
| onmouseout | onMouseOut | Mouse hover exit |
| onkeydown | onKeyDown | Key press down |
| onkeyup | onKeyUp | Key release |
| onfocus | onFocus | Element receives focus |
| onblur | onBlur | Element loses focus |
| onload | onLoad | Resource loading complete |

**Mouse Events:**

const MouseEvents = () => {  
 return (  
 <div  
 onClick={() => console.log('Click')}  
 onDoubleClick={() => console.log('Double click')}  
 onMouseDown={() => console.log('Mouse down')}  
 onMouseUp={() => console.log('Mouse up')}  
 onMouseOver={() => console.log('Mouse over')}  
 onMouseOut={() => console.log('Mouse out')}  
 onMouseMove={() => console.log('Mouse move')}  
 >  
 Mouse Event Example  
 </div>  
 );  
};

**Keyboard Events:**

const KeyboardEvents = () => {  
 return (  
 <input  
 onKeyDown={(e) => console.log('Key down:', e.key)}  
 onKeyUp={(e) => console.log('Key up:', e.key)}  
 onKeyPress={(e) => console.log('Key press:', e.key)}  
 />  
 );  
};

**Form Events:**

const FormEvents = () => {  
 return (  
 <form onSubmit={(e) => e.preventDefault()}>  
 <input  
 onChange={(e) => console.log('Change:', e.target.value)}  
 onFocus={() => console.log('Focus')}  
 onBlur={() => console.log('Blur')}  
 />  
 <button type="submit">Submit</button>  
 </form>  
 );  
};

**Touch Events (Mobile):**

const TouchEvents = () => {  
 return (  
 <div  
 onTouchStart={() => console.log('Touch start')}  
 onTouchMove={() => console.log('Touch move')}  
 onTouchEnd={() => console.log('Touch end')}  
 >  
 Touch me on mobile  
 </div>  
 );  
};

**Complete Example with Multiple Events:**

const EventDemo = () => {  
 const [message, setMessage] = useState('');  
  
 return (  
 <div>  
 <button  
 onClick={() => setMessage('Button clicked')}  
 onMouseOver={() => console.log('Mouse over button')}  
 onMouseOut={() => console.log('Mouse out of button')}  
 >  
 Hover and Click Me  
 </button>  
   
 <input  
 onChange={(e) => setMessage(`Typing: ${e.target.value}`)}  
 onFocus={() => console.log('Input focused')}  
 onBlur={() => console.log('Input blurred')}  
 placeholder="Type here..."  
 />  
   
 <p>{message}</p>  
 </div>  
 );  
};