upGrad ENTERPRISE

React Components, State and Event Handling

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

Creating and Composing Functional Components

Functional components are JavaScript functions that return JSX to define UI elements. They are the modern way to write React components due to their simplicity and support for hooks.

- **Creating a Functional Component**:
  - Define a function that returns JSX.
  - Use the `export` keyword to make it reusable.
  - Example:

```
function Welcome() {
  return <h1>Hello, World!</h1>;
}
export default Welcome;
```

- **Composing Components**:
  - Components can be nested or combined to build complex UIs.
  - Import and use components like HTML tags.
  - Example:

```
import Welcome from './Welcome';
function App() {
  return (
    <div>
      <Welcome />
      <Welcome />
    </div>
  );
}
export default App;
```

Component Naming and File Conventions

- **Naming**:
  - Use PascalCase for component names (e.g., `MyComponent`).
  - Avoid reserved JavaScript words or generic terms like `Component`.
  - Names should reflect the component's purpose (e.g., `UserProfile`, `Navbar`).
- **File Conventions**:
  - Store each component in its own file, typically in a `components/` folder.
  - File names match the component name (e.g., `MyComponent.jsx`).
  - Use `.jsx` or `.js` extension for React components.
  - Group related components in subfolders (e.g., `components/Navbar/Navbar.jsx`).

- Example structure:

```
src/
  components/
    Welcome.jsx
    Navbar/
      Navbar.jsx
      NavbarItem.jsx
```

What are Events in React?

- Events in React are actions triggered by user interactions (e.g., clicks, typing, mouse movements) or system-generated events.
- React uses synthetic events, a cross-browser wrapper around native DOM events, for consistent behavior.

Event Handling in React

- React events are handled by passing functions as props to elements.
- Event handlers are typically defined within the component.
- Use camelCase for event names (e.g., `onClick`, `onChange`).
- Syntax: `<element eventName={handlerFunction}>`.
- Example: `<button onClick={buttonClick}>`.

Handling DOM Events (Examples)

- onClick:

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

- **onChange** (for inputs):

```
function Input() {
  function handleChange(event) {
    console.log('Input value:', event.target.value);
  }
  return <input type="text" onChange={handleChange} />;
}
```

- **onSubmit** (for forms):

```
function Form() {
  function handleSubmit(event) {
    event.preventDefault();
    console.log('Form submitted');
  }
  return (
    <form onSubmit={handleSubmit}>
      <button type="submit">Submit</button>
    </form>
  );
}
```

Event Handler with Parameters

To pass parameters to an event handler, use an arrow function or bind the handler.

- **Using Arrow Function**:

```
function Item({ id, name }) {
  const handleClick = (itemId) => {
    console.log(`Clicked item with ID: ${itemId}`);
  };
  return <button onClick={() => handleClick(id)}>{name}</button>;
}
```

Working with State in React

What is State in React?

State is a built-in object in React that holds dynamic data for a component. When state changes, React re-renders the component to reflect the updated data.

- **Characteristics**:
  - Local to the component (unless shared via props or context).
  - Managed internally by the component.
  - Immutable directly; updated using setter functions.

State in Functional Components

- State is managed in functional components using the `useState` hook.
- State updates trigger re-renders to keep the UI in sync with data.

useState Hook for State Management

The `useState` hook allows functional components to manage state. It returns an array with the current state value and a setter function.

- **Syntax**:

```
import { useState } from 'react';
function Counter() {
  const [count, setCount] = useState(0); // Initial state: 0
  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
    </div>
  );
}
```

Q: How to handle object State?

- **Example with Object State:**

```
function UserForm() {
  const [user, setUser] = useState({ name: '', age: 0 });
  const handleNameChange = (e) => {
    setUser({ ...user, name: e.target.value });
  };
  return (
    <div>
      <input type="text" value={user.name} onChange={handleNameChange} />
      <p>Name: {user.name}</p>
    </div>
  );
}
```

Props in React

What are Props in React?

Props (short for properties) are read-only inputs passed to components to customize their behavior or rendering. They allow components to be reusable and dynamic.

- **Characteristics**:
  - Immutable within the receiving component.
  - Passed as attributes in JSX.
  - Can include data, functions, or other components.

Passing Props to Components

- Props are passed as attributes in JSX and accessed as an object in the component.
- Example:

```
function Greeting({ name }) {
  return <h1>Hello, {name}!</h1>;
}
function App() {
  return <Greeting name="Alice" />;
}
```

- Passing Multiple Props:

```
function UserCard({ name, age, email }) {
  return (
    <div>
      <p>Name: {name}</p>
      <p>Age: {age}</p>
      <p>Email: {email}</p>
    </div>
  );
}
function App() {
  return <UserCard name="Bob" age={25} email="bob@example.com" />;
}
```

Passing Event Handlers as Props

Event handlers can be passed as props to child components for handling events in a parent component.

- **Example**:

```
function Button({ onClick, label }) {
  return <button onClick={onClick}>{label}</button>;
}
function App() {
  const handleClick = () => {
    console.log('Button clicked from parent!');
  };
  return <Button onClick={handleClick} label="Click Me" />;
}
```

Self Check Questions

Check you knowledge

1. What is the difference between functional and class components?
2. How does the `useState` hook work?
3. How do you pass data from a child to a parent component?
4. How do you prevent default behavior in React events?
5. What is the difference between state and props?
6. How can you optimize event handlers in React?
7. What happens if you update state directly (e.g., `state.value = 10`)?

Q & A

1. **What is the difference between functional and class components?**
    - Functional components are simpler, use hooks, and are preferred in modern React. Class components use lifecycle methods and `this` for state/props.
2. **How does the `useState` hook work?**
    - `useState` declares a state variable and its setter. It preserves state between renders and triggers re-renders on state updates.
3. **How do you pass data from a child to a parent component?**
    - Pass a callback function as a prop to the child, which the child calls with the data.

4. **How do you prevent default behavior in React events?**
    - Use `event.preventDefault()` in the handler (e.g., to prevent form submission).
5. **What is the difference between state and props?**
    - State is internal, mutable data managed by the component. Props are external, immutable data passed to the component.
6. **How can you optimize event handlers in React?**
    - Use `useCallback` to memoize handlers, prevent unnecessary re-renders, and avoid re-creating functions on every render.

7. **What happens if you update state directly (e.g., `state.value = 10`)?**
    - Direct state mutation doesn't trigger a re-render. Always use the setter function (e.g., `setState`) provided by `useState`.

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com