You use `useState` when you need to add state to a functional component in React. State is essential when your component needs to remember information between renders, respond to user input, or manage dynamic data.

### Situations to Use `useState`

1. **User Input Handling**:
   - For input fields, checkboxes, radio buttons, text areas, and other form elements.
   ```javascript
   function NameInput() {
     const [name, setName] = useState('');

     return (
       <div>
         <input
           type="text"
           value={name}
           onChange={(e) => setName(e.target.value)}
           placeholder="Enter your name"
         />
         <p>Your name is: {name}</p>
       </div>
     );
   }
   ```

2. **Toggle Visibility**:
   - For showing and hiding elements, such as modals, dropdowns, or sections of a page.
   ```javascript
   function ToggleVisibility() {
     const [isVisible, setIsVisible] = useState(false);

     return (
       <div>
         <button onClick={() => setIsVisible(!isVisible)}>
           {isVisible ? 'Hide' : 'Show'} Content
         </button>
         {isVisible && <p>This content is visible</p>}
       </div>
     );
   }
   ```

3. **Counters**:
   - For simple numeric states that increase or decrease.
   ```javascript
   function Counter() {
     const [count, setCount] = useState(0);

     return (
       <div>
         <p>You clicked {count} times</p>
         <button onClick={() => setCount(count + 1)}>Click me</button>
       </div>
     );
   }
   ```

4. **Fetching Data**:
   - To manage the loading state and store fetched data.
   ```javascript
   import { useEffect, useState } from 'react';
   ```

```
   function DataFetcher() {
     const [data, setData] = useState(null);
     const [loading, setLoading] = useState(true);

     useEffect(() => {
       fetch('https://api.example.com/data')
         .then(response => response.json())
         .then(data => {
           setData(data);
           setLoading(false);
         });
     }, []);

     if (loading) {
       return <p>Loading...</p>;
     }

     return <div>Data: {JSON.stringify(data)}</div>;
   }
   ```

5. **Form Submission**:
   - To manage form data and submission state.
   ```javascript
   function Form() {
     const [formData, setFormData] = useState({ name: '', email: '' });

     const handleSubmit = (e) => {
       e.preventDefault();
       console.log('Form submitted:', formData);
     };

     return (
       <form onSubmit={handleSubmit}>
         <input
           type="text"
           value={formData.name}
           onChange={(e) => setFormData({ ...formData, name: e.target.value })}
           placeholder="Name"
         />
         <input
           type="email"
           value={formData.email}
           onChange={(e) => setFormData({ ...formData, email: e.target.value })}
           placeholder="Email"
         />
         <button type="submit">Submit</button>
       </form>
     );
   }
   ```

### Benefits of Using `useState`

1. **Simplicity**: It provides a simple API for managing state in functional components.
2. **Encapsulation**: State is encapsulated within the component, making it easy to manage and reason about.
3. **Readability**: Functional components with hooks can be easier to read and understand compared to class components.
4. **Reusability**: Functional components are more reusable, and hooks can be shared between components using custom hooks.

By using `useState`, you can efficiently manage state in your functional components, enabling dynamic and interactive user interfaces.