`useState` is a React Hook that allows you to add state to functional components. It lets you declare a state variable and a function to update it, similar to how state is managed in class components. The state is preserved across re-renders of the component.

Here's a breakdown of how `useState` works:

1. **Importing `useState`**: You need to import `useState` from the 'react' package.
2. **Declaring State**: You call `useState` inside your functional component to declare a state variable and a function to update it. `useState` takes the initial state as an argument.
3. **Using State**: You can use the state variable in your component and update it using the provided function.

Here's a basic example:

```javascript
import React, { useState } from 'react';

function Counter() {
  // Declare a state variable 'count' and a function 'setCount' to update it
  const [count, setCount] = useState(0);

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

export default Counter;
```

Detailed Explanation:

1. **Initial State**:
   ```javascript
   const [count, setCount] = useState(0);
   ```
   This line declares a state variable `count` and a function `setCount` to update it. The initial state of `count` is set to `0`.

2. **Using the State Variable**:
   ```javascript
   <p>You clicked {count} times</p>
   ```
   The current value of `count` is used in the JSX.

3. **Updating the State**:
   ```javascript
   <button onClick={() => setCount(count + 1)}>
     Click me
   </button>
   ```
   The `setCount` function is called when the button is clicked, updating `count` to `count + 1`. This causes the component to re-render and display the updated count.

Benefits of `useState`:
- **Simplifies State Management**: It makes adding state to functional components straightforward.
- **No Need for Class Components**: You can manage state without using class components, making the code cleaner and more readable.

- **Encapsulation**: Each call to `useState` is independent, allowing for multiple state variables in a single component.

Example with Multiple State Variables:
```javascript
import React, { useState } from 'react';

function UserInfo() {
  const [name, setName] = useState('');
  const [age, setAge] = useState(0);

  return (
    <div>
      <input
        type="text"
        value={name}
        onChange={(e) => setName(e.target.value)}
        placeholder="Enter your name"
      />
      <input
        type="number"
        value={age}
        onChange={(e) => setAge(Number(e.target.value))}
        placeholder="Enter your age"
      />
      <p>
        Name: {name}, Age: {age}
      </p>
    </div>
  );
}

export default UserInfo;
```

In this example, `useState` is used to manage two state variables, `name` and `age`, demonstrating how you can handle multiple pieces of state within a single functional component.