Reference

How to use the `useState` hook in React! Each example demonstrates a different aspect of managing state in functional components.

- **Basic Counter**: Shows how to increment and decrement a count using `useState`.
- **Form Input State**: Demonstrates how to manage the state of a form input, updating it as the user types.
- **Toggle Visibility**: Illustrates how to toggle the visibility of an element, like showing or hiding a paragraph.
- **Managing Multiple Pieces of State**: Shows how to manage multiple state variables (e.g., name and age) within the same component.
- **State with Initial Value from Props**: Explains how to initialize a state variable with a value passed as a prop, useful for setting initial values.
- **Updating State Based on Previous State**: Highlights the best practice of using a functional update when the new state depends on the previous state, ensuring accuracy in asynchronous updates.

1. Basic Counter

This is a simple counter example that increments and decrements a count.

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

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}

export default Counter;
```

2. Form Input State

This example shows how to manage the state of a form input field.

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

function Form() {
  const [name, setName] = useState('');

  const handleChange = (e) => {
    setName(e.target.value);
```

```
  };

  return (
    <div>
      <input type="text" value={name} onChange={handleChange} />
      <p>Your name is: {name}</p>
    </div>
  );
}

export default Form;
```

3. Toggle Visibility

This example demonstrates how to toggle the visibility of an element.

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

function Toggle() {
  const [isVisible, setIsVisible] = useState(true);

  return (
    <div>
      <button onClick={() => setIsVisible(!isVisible)}>
        {isVisible ? 'Hide' : 'Show'} Text
      </button>
      {isVisible && <p>This is some text that can be toggled.</p>}
    </div>
  );
}

export default Toggle;
```

4. Managing Multiple Pieces of State

You can manage multiple pieces of state by calling `useState` multiple times.

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

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

  return (
    <div>
      <div>
        <label>Name: </label>
        <input type="text" value={name} onChange={(e) =>
setName(e.target.value)} />
      </div>
      <div>
        <label>Age: </label>
        <input type="number" value={age} onChange={(e) =>
setAge(parseInt(e.target.value))} />
```

```
      </div>
      <p>
        Name: {name}, Age: {age}
      </p>
    </div>
  );
}

export default MultiState;
```

5. State with Initial Value from Props

You can initialize state with a value derived from props.

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

function Greeting({ initialName }) {
  const [name, setName] = useState(initialName);

  return (
    <div>
      <p>Hello, {name}!</p>
      <input type="text" value={name} onChange={(e) =>
setName(e.target.value)} />
    </div>
  );
}

export default Greeting;
```

6 .Updating State Based on Previous State

When updating state based on the previous state, it's best to use a functional update to ensure you have the most recent state.

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

function Counter() {
  const [count, setCount] = useState(0);

  const increment = () => {
    setCount((prevCount) => prevCount + 1);
  };

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={increment}>Increment</button>
    </div>
  );
}
```

```
export default Counter;
```