

[LEARN REACT](#) >

Adding Interactivity

Some things on the screen update in response to user input. For example, clicking an image gallery switches the active image. In React, data that changes over time is called *state*. You can add state to any component, and update it as needed. In this chapter, you'll learn how to write components that handle interactions, update their state, and display different output over time.

In this chapter

- [How to handle user-initiated events](#)
- [How to make components “remember” information with state](#)
- [How React updates the UI in two phases](#)
- [Why state doesn't update right after you change it](#)
- [How to queue multiple state updates](#)
- [How to update an object in state](#)
- [How to update an array in state](#)

Responding to events

React lets you add *event handlers* to your JSX. Event handlers are your own functions that will be triggered in response to user interactions like clicking, hovering, focusing on form inputs, and so on.

Built-in components like `<button>` only support built-in browser events like

`onClick`. However, you can also create your own components, and give their event handler props any application-specific names that you like.

App.js

[Download](#)[Reset](#)

```
1  export default function App() {
2    return (
3      <Toolbar
4        onPlayMovie={() => alert('Playing!')}
5        onUploadImage={() => alert('Uploading!')}
6      />
7    );
8  }
9
10 function Toolbar({ onPlayMovie, onUploadImage }) {
11   return (
12     <div>
13       <Button onClick={onPlayMovie}>
14         Play Movie
15       </Button>
16       <Button onClick={onUploadImage}>
17         Upload Image
18       </Button>
19     </div>
20   );
21 }
22
23 function Button({ onClick, children }) {
24   return (
25     <button onClick={onClick}>
26       {children}
27     </button>
28   );
29 }
30
```

[Show less](#)



Ready to learn this topic?

Read [Responding to Events](#) to learn how to add event handlers.

Read More

State: a component's memory

Components often need to change what's on the screen as a result of an interaction. Typing into the form should update the input field, clicking “next” on an image carousel should change which image is displayed, clicking “buy” puts a product in the shopping cart. Components need to “remember” things: the current input value, the current image, the shopping cart. In React, this

kind of component-specific memory is called *state*.

You can add state to a component with a `useState` Hook. *Hooks* are special functions that let your components use React features (state is one of those features). The `useState` Hook lets you declare a state variable. It takes the initial state and returns a pair of values: the current state, and a state setter function that lets you update it.

```
const [index, setIndex] = useState(0);  
const [showMore, setShowMore] = useState(false);
```

Here is how an image gallery uses and updates state on click:

App.js data.js

Reset

```
1  import { useState } from 'react';  
2  import { sculptureList } from './data.js';  
3  
4  export default function Gallery() {  
5    const [index, setIndex] = useState(0);  
6    const [showMore, setShowMore] = useState(false);  
7  
8    function handleNextClick() {  
9      setIndex(index + 1);  
10   }  
11  
12   function handleMoreClick() {  
13     setShowMore(!showMore);  
14   }  
15  
16   let sculpture = sculptureList[index];  
17   return (  
18     <>  
19     <button onClick={handleNextClick}>  
20       Next  
21     </button>
```

```
--      , .....
22      <h2>
23        <i>{sculpture.name} </i>
24        by {sculpture.artist}
25      </h2>
26      <h3>
27        ({index + 1} of {sculptureList.length})
28      </h3>
29      <button onClick={handleMoreClick}>
30        {showMore ? 'Hide' : 'Show'} details
31      </button>
32      {showMore && <p>{sculpture.description}</p>}
33      <img
34        src={sculpture.url}
35        alt={sculpture.alt}
36      />
37    </>
38  );
39 }
40
```

Show less

Ready to learn this topic?

Read [State: A Component's Memory](#) to learn how to remember a value and update it on interaction.

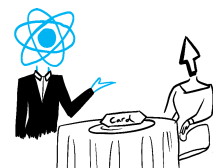
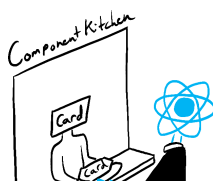
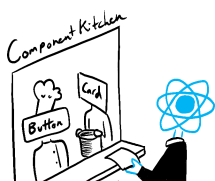
Read More

Render and commit

Before your components are displayed on the screen, they must be rendered by React. Understanding the steps in this process will help you think about how your code executes and explain its behavior.

Imagine that your components are cooks in the kitchen, assembling tasty dishes from ingredients. In this scenario, React is the waiter who puts in requests from customers and brings them their orders. This process of requesting and serving UI has three steps:

1. **Triggering** a render (delivering the diner's order to the kitchen)
2. **Rendering** the component (preparing the order in the kitchen)
3. **Committing** to the DOM (placing the order on the table)





Trigger



Render



Commit

Ready to learn this topic?

Read [Render and Commit](#) to learn the lifecycle of a UI update.

[Read More](#)

State as a snapshot

Unlike regular JavaScript variables, React state behaves more like a snapshot. Setting it does not change the state variable you already have, but instead triggers a re-render. This can be surprising at first!

```
console.log(count); // 0
setCount(count + 1); // Request a re-render with 1
console.log(count); // Still 0!
```

React works this way to help you avoid subtle bugs. Here is a little chat app. Try to guess what happens if you press “Send” first and *then* change the recipient to Bob. Whose name will appear in the `alert` five seconds later?

[App.is](#)

Download

Reset

```
1 import { useState } from 'react';
2
3 export default function Form() {
4   const [to, setTo] = useState('Alice');
5   const [message, setMessage] = useState('Hello');
6
7   function handleSubmit(e) {
8     e.preventDefault();
9     setTimeout(() => {
10       alert(`You said ${message} to ${to}`);
11     }, 5000);
12   }
13
14   return (
15     <form onSubmit={handleSubmit}>
16       <label>
17         To: { ' '}
18         <select
19           value={to}
20           onChange={e => setTo(e.target.value)}>
21           <option value="Alice">Alice</option>
22           <option value="Bob">Bob</option>
23         </select>
24       </label>
25       <textarea
26         placeholder="Message"
27         value={message}
28         onChange={e => setMessage(e.target.value)}
29       />
30       <button type="submit">Send</button>
31     </form>
32   );
33 }
34
```

[Show less](#)

Ready to learn this topic?

Read [State as a Snapshot](#) to learn why state appears “fixed” and unchanging inside the event handlers.

[Read More](#)

Queueing a series of state updates

This component is buggy: clicking “+3” increments the score only once.

[App.js](#)

Download

Reset

```
1 import { useState } from 'react';
```

```
2
3 export default function Counter() {
4   const [score, setScore] = useState(0);
5
6   function increment() {
7     setScore(score + 1);
8   }
9
10  return (
11    <>
12      <button onClick={() => increment()}>+1</button>
13      <button onClick={() => {
14        increment();
15        increment();
16        increment();
17      }}>+3</button>
18      <h1>Score: {score}</h1>
19    </>
20  )
21 }
22
```

Show less

State as a Snapshot explains why this is happening. Setting state requests a new re-render, but does not change it in the already running code. So `score` continues to be `0` right after you call `setScore(score + 1)`.

```
console.log(score); // 0
setScore(score + 1); // setScore(0 + 1);
console.log(score); // 0
setScore(score + 1); // setScore(0 + 1);
console.log(score); // 0
setScore(score + 1); // setScore(0 + 1);
console.log(score); // 0
```

You can fix this by passing an *updater function* when setting state. Notice how replacing `setScore(score + 1)` with `setScore(s => s + 1)` fixes the “+3” button. This is handy if you need to queue multiple state updates.

App.js

[Download](#)[Reset](#)

```
1  import { useState } from 'react';
2
3  export default function Counter() {
4    const [score, setScore] = useState(0);
5
6    function increment() {
7      setScore(s => s + 1);
8    }
9
10   return (
11     <>
12       <button onClick={() => increment()}>+1</button>
13       <button onClick={() => {
14         increment();
15         increment();
16         increment();
17       }}>+3</button>
```

```
17      // ...  
18      <h1>Score: {score}</h1>  
19    </>  
20  )  
21 }  
22
```

Show less

Ready to learn this topic?

Read [Queueing a Series of State Updates](#) to learn how to queue multiple updates before the next render.

Read More

Updating objects in state

State can hold any kind of JavaScript value, including objects. But you shouldn't change objects and arrays that you hold in the React state directly. Instead, when you want to update an object and array, you need to create a new one (or make a copy of an existing one), and then update the state to use that copy.

Usually, you will use the `...` spread syntax to copy objects and arrays that you want to change. For example, updating a nested object could look like this:

App.js

Download

Reset

```
1  import { useState } from 'react';
2
3  export default function Form() {
4    const [person, setPerson] = useState({
5      name: 'Niki de Saint Phalle',
6      artwork: {
7        title: 'Blue Nana',
8        city: 'Hamburg',
9        image: 'https://i.imgur.com/Sd1AgU0m.jpg',
10     }
11   });
12
13   function handleNameChange(e) {
14     setPerson({
15       ...person,
16       name: e.target.value
17     });
18   }
19
20   function handleTitleChange(e) {
21     setPerson({
22       ...person,
23       artwork: {
```

```
24     ...person.artwork,  
25     title: e.target.value  
26   }  
27   });  
28 }  
29  
30 function handleCityChange(e) {  
31   setPerson({  
32     ...person,  
33     artwork: {  
34       ...person.artwork,  
35       city: e.target.value  
36     }  
37   });  
38 }  
39  
40 function handleImageChange(e) {  
41   setPerson({  
42     ...person,  
43     artwork: {  
44       ...person.artwork,  
45       image: e.target.value  
46     }  
47   });  
48 }  
49  
50 return (  
51   <>  
52     <label>  
53       Name:  
54       <input  
55         value={person.name}  
56         onChange={handleNameChange}  
57       />  
58     </label>  
59     <label>  
60       Title:  
61       <input
```

Show less

If copying objects in code gets tedious, you can use a library like [Immer](#) to reduce repetitive code:

App.js package.json

Reset

```
1  import { useImmer } from 'use-immer';
2
3  export default function Form() {
4    const [person, updatePerson] = useImmer({
5      name: 'Niki de Saint Phalle',
6      artwork: {
7        title: 'Blue Nana',
8        city: 'Hamburg',
9        image: 'https://i.imgur.com/Sd1AgUOm.jpg',
10     }
11   });
12
13   function handleNameChange(e) {
14     updatePerson(draft => {
15       draft.name = e.target.value;
16     });
17   }
18
19   function handleTitleChange(e) {
20     updatePerson(draft => {
21       draft.artwork.title = e.target.value;
22     });
23   }
```



```
--  
24  
25 function handleCityChange(e) {  
26   updatePerson(draft => {  
27     draft.artwork.city = e.target.value;  
28   });  
29 }  
30  
31 function handleImageChange(e) {  
32   updatePerson(draft => {  
33     draft.artwork.image = e.target.value;  
34   });  
35 }  
36  
37 return (  
38   <>  
39     <label>  
40       Name:  
41       <input  
42         value={person.name}  
43         onChange={handleNameChange}  
44       />  
45     </label>  
46     <label>  
47       Title:  
48       <input  
49         value={person.artwork.title}  
50         onChange={handleTitleChange}  
51       />  
52     </label>  
53     <label>  
54       City:  
55       <input  
56         value={person.artwork.city}  
57         onChange={handleCityChange}  
58       />  
59     </label>  
60     <label>  
61       Image:  
62       <input  
63         value={person.artwork.image}
```

```
63         {person.artwork.image}
64         onChange={handleImageChange}
65       />
66     </label>
67     <p>
68       <i>{person.artwork.title}</i>
69       {' by '}
70       {person.name}
71     <br />
72     (located in {person.artwork.city})
```

Show less

Ready to learn this topic?

Read [Updating Objects in State](#) to learn how to update objects correctly.

[Read More](#)

Updating arrays in state

Arrays are another type of mutable JavaScript objects you can store in state and should treat as read-only. Just like with objects, when you want to update an array stored in state, you need to create a new one (or make a copy of an existing one), and then set state to use the new array:

[App.js](#)

Download

Reset

```
1  import { useState } from 'react';
2
3  let nextId = 3;
4  const initialList = [
5    { id: 0, title: 'Big Bellies', seen: false },
6    { id: 1, title: 'Lunar Landscape', seen: false },
7    { id: 2, title: 'Terracotta Army', seen: true },
8  ];
9
10 export default function BucketList() {
11   const [list, setList] = useState(
12     initialList
13   );
14
```

```
15   function handleToggle(artworkId, nextSeen) {
16     setList(list.map(artwork => {
17       if (artwork.id === artworkId) {
18         return { ...artwork, seen: nextSeen };
19       } else {
20         return artwork;
21       }
22     }));
23   }
24
25   return (
26     <>
27       <h1>Art Bucket List</h1>
28       <h2>My list of art to see:</h2>
29       <ItemList
30         artworks={list}
31         onToggle={handleToggle} />
32     </>
33   );
34 }
35
36 function ItemList({ artworks, onToggle }) {
37   return (
38     <ul>
39       {artworks.map(artwork => (
40         <li key={artwork.id}>
41           <label>
42             <input
43               type="checkbox"
44               checked={artwork.seen}
45               onChange={e => {
46                 onToggle(
47                   artwork.id,
48                   e.target.checked
49                 )};
50             }}
51           </>
52           {artwork.title}
53         </label>
54       </li>
55     )}
```

```
55     })}
56     </ul>
57   );
```

Show less

If copying arrays in code gets tedious, you can use a library like [Immer](#) to reduce repetitive code:

App.js package.json

Reset

```
1  import { useState } from 'react';
2  import { useImmer } from 'use-immer';
3
4  let nextId = 3;
5  const initialList = [
6    { id: 0, title: 'Big Bellies', seen: false },
7    { id: 1, title: 'Lunar Landscape', seen: false },
8    { id: 2, title: 'Terracotta Army', seen: true },
9  ];
```

```
10
11 export default function BucketList() {
12   const [list, updateList] = useImmer(initialList);
13
14   function handleToggle(artworkId, nextSeen) {
15     updateList(draft => {
16       const artwork = draft.find(a =>
17         a.id === artworkId
18       );
19       artwork.seen = nextSeen;
20     });
21   }
22
23   return (
24     <>
25       <h1>Art Bucket List</h1>
26       <h2>My list of art to see:</h2>
27       <ItemList
28         artworks={list}
29         onToggle={handleToggle} />
30     </>
31   );
32 }
33
34 function ItemList({ artworks, onToggle }) {
35   return (
36     <ul>
37       {artworks.map(artwork => (
38         <li key={artwork.id}>
39           <label>
40             <input
41               type="checkbox"
42               checked={artwork.seen}
43               onChange={e => {
44                 onToggle(
45                   artwork.id,
46                   e.target.checked
47                 );
48               }}
49             />
```

```
50         {artwork.title}
51       </label>
52     </li>
53   )}
54 </ul>
55 );
56 }
57
```

Show less

Ready to learn this topic?

Read [Updating Arrays in State](#) to learn how to update arrays correctly.

Read More

What's next?

Head over to [Responding to Events](#) to start reading this chapter page by page!

Or, if you're already familiar with these topics, why not read about [Managing State](#)?

PREVIOUS

[Keeping Components Pure](#)

NEXT

[Responding to Events](#)

How do you like these docs?

[Take our survey!](#)

FACEBOOK

Open Source

©2023

Learn React

Quick Start

Installation

Describing the UI

Adding Interactivity

Managing State

API Reference

React APIs

React DOM APIs

Escape Hatches

Community

[Code of Conduct](#)

[Acknowledgements](#)

[Docs Contributors](#)

[Meet the Team](#)

[Blog](#)

More

[React Native](#)

[Privacy](#)

[Terms](#)