

# How Do You Update Objects in State?

---

(🔗) [freecodecamp.org/learn/front-end-development-libraries-v9/lecture-working-with-state-and-responding-to-events-in-react/how-do-you-update-objects-in-state](https://freecodecamp.org/learn/front-end-development-libraries-v9/lecture-working-with-state-and-responding-to-events-in-react/how-do-you-update-objects-in-state)



Updating objects in state in React can be tricky if you're used to changing object property values directly.

React treats state as immutable, meaning you should not modify it directly.

Let's look at what happens if you try to change objects in React state directly, and then dive into the correct way to do it.

Let's say you have an object in your component state that represents a user's profile, and you want the user to update their age:

```
import { useState } from "react";

function Profile() {
  const [user, setUser] = useState({
    name: "John Doe",
    age: 31,
    city: "LA",
  });

  // Change user age directly
  const handleAgeChange = (e) => {
    user.age = e.target.value;
    console.log(user);
  };

  return (
    <div>
      <h1>User Profile</h1>
      <p>Name: {user.name}</p>
      <p>Age: {user.age}</p>
      <p>City: {user.city}</p>

      <h2>Update User Age </h2>
      <input type="number" value={user.age} onChange={handleAgeChange} />
    </div>
  );
}

export default Profile;
```

This code will not work because you're directly modifying the `user.age` property.

Even though `console.log(user)` will show the new age in the console, React won't re-render the component to show it in the user interface because you didn't use the setter function, `setUser`.

To update an object directly in the state, you need to use the setter function to create a new object with the updated value. For example:

```
const handleAgeChange = (e) => {
  setUser({
    age: e.target.value,
  });
};
```

That works. But if you look at the page now, the user's age gets updated, but the values for the name and city are lost.

This is because the new object you passed to the setter function only contained a key/value pair for `age`.

To prevent this from happening, you can copy the existing object first, then update only the property you want to update, in this case, `age`.

To do this, you can pass a special function called an updater function to your setter function, `setUser`. The updater function takes the pending state as an argument, here, called `prevUser`, and should return the next state:

```
const handleAgeChange = (e) => {
  setUser((prevUser) => {
    const updatedUser = { ...prevUser, age: e.target.value };
    console.log('Previous State:', prevUser);
    console.log('Updated State:', updatedUser);
    return updatedUser;
  });
};
```

As you can see, we create a new user object called `updatedUser` by using the spread syntax to copy the pending user object, `...prevUser`. We then update the age based on the form input and return `updatedUser` at the bottom of the function as the next state.

Now your project works as expected, and updates to the age input don't affect the user's name or city name. You can also see the previous and updated states in the console.

This is the ideal way to update an object in state, especially when you're not updating all the properties.

To update the remaining `name` and `city` properties, you can write them as separate setting functions and connect them to their respective inputs:

```
const handleNameChange = (e) => {
  setUser((prevUser) => ({
    ...prevUser,
    name: e.target.value,
  }));
};

const handleCityChange = (e) => {
  setUser((prevUser) => ({
    ...prevUser,
    city: e.target.value,
  }));
};
```

Or you can combine them into a single setter function like this:

```
const handleChange = (e) => {
  const { name, value } = e.target;
  setUser((prevUser) => ({
    ...prevUser,
    [name]: value,
  }));
};
```

To make this work, each input field has to have a `name` attribute.

Here's the full code now:

```
import { useState } from "react";

function Profile() {
  const [user, setUser] = useState({ name: "John Doe", age: 31, city: "LA" });

  const handleChange = (e) => {
    const { name, value } = e.target;

    setUser((prevUser) => ({...prevUser, [name]: value}));
  };

  return (
    <div>
      <h1>User Profile</h1>
      <p>Name: {user.name}</p>
      <p>Age: {user.age}</p>
      <p>City: {user.city}</p>

      <h2>Update User Age </h2>
      <input type="number" name="age" value={user.age} onChange={handleChange} />

      <h2>Update User Name </h2>
      <input type="text" name="name" value={user.name} onChange={handleChange} />

      <h2>Update User City </h2>
      <input type="text" name="city" value={user.city} onChange={handleChange} />
    </div>
  );
}

export default Profile;
```

## Questions

---

### How does React treat state in a component?

As mutable, allowing direct modification.

As immutable, meaning it should not be modified directly.

As a global variable accessible in all components.

As a temporary variable that resets on each render.

### How should you update an object stored in React state?

Use the setter function to create a new object.

Modify the object directly in the state.

Use `Object.assign()` without calling the setter.

Use the `push()` method to add properties to the object.

What is the setter function in this code?

```
const [user, setUser] = useState({  
  name: 'John Doe',  
  age: 31,  
  city: 'LA',  
});
```

`user`

`useState`

`age`

`setUser`