

Object and Array State

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

State Object Mutation is BAD

- 📌 React relies on state immutability to detect changes efficiently.
- 📌 Mutating state directly doesn't trigger re-renders because React compares the old and new state using shallow equality (reference comparison).

```
const [user, setUser] = useState({ name: "John", age: 25 });  
  
// ❌ Directly mutating the state  
user.age = 26;  
setUser(user);
```



React won't detect a change because `user` still has the same reference in memory. No re-render happens!

Therefore, although objects in React state are technically mutable, you should treat them **as if** they were immutable—like numbers, booleans, and strings. Instead of mutating them, you should always replace them.

Updating arrays without mutation

- **Arrays in React state are read-only** – avoid mutating them with methods like `push()` and `pop()`.

- **Do not modify items directly** (e.g., `arr[0] = 'bird'`); instead, always create a new array.
- **Use non-mutating methods** like `filter()` , `map()` , and `concat()` to generate a new array.
- **Pass the new array** to the state-setting function to ensure proper re-renders.

	avoid (mutates the array)	prefer (returns a new array)
adding	<code>push</code> , <code>unshift</code>	<code>concat</code> , <code>[...arr]</code> spread syntax
removing	<code>pop</code> , <code>shift</code> , <code>splice</code>	<code>filter</code> , <code>slice</code>
replacing	<code>splice</code> , <code>arr[i] = ...</code> assignment	<code>map</code>
sorting	<code>reverse</code> , <code>sort</code>	copy the array first

Insert an Item At the End of the Array

```
setMyList(prevList => [...prevList, newItem]);
```

Update an Item in Array

```
setMyList(myList.map(artwork => {
  if (artwork.id === artworkId) {
    // Create a *new* object with changes
    return { ...artwork, seen: nextSeen };
  } else {
    // No changes
    return artwork;
  }
}));
```

Delete an Item in Array

```
setMyList(myList.filter(item => item.id !== deletId));
```




Reverse the Array

```
function handleClick() {  
  const nextList = [...list];  
  nextList.reverse();  
  setList(nextList);  
}
```

Insert an element at a particular index

```
const nextArtists = [  
  // Items before the insertion point:  
  ...artists.slice(0, insertAt),  
  // New item:  
  { id: nextId++, name: name },  
  // Items after the insertion point:  
  ...artists.slice(insertAt)  
];
```

Assignments

- Add a search functionality in the Todo App
- Add a Dropdown to filter the Items.
 - All Todos
 - Completed Todos
 - Incomplete Todos
- Add a dropdown to assign Labels to the Todos. Show this label in all todos. Change the color of the Todo card based on the Label.
 -  Important
 -  Priority
 -  General
- Create a new List to show the completed tasks below the pending tasks.