# State Updates

If your state in a React component is an array of an object, you must be careful in how you update it.

Do not update arrays or objects directly.

```
const [colors, setColors] = useState(['red', 'green', 'blue']);

const changeColor = () => {
  // Bad!  This directly changes the 'colors' state!
  colors[0] = 'orange';

  setColors(colors);
};
```

Instead, there are special techniques to update arrays and objects by first creating a *new* array or object. Even though this does require a tiny, tiny bit of extra processing power, it allows React to do far less work when re-rendering a component.

## Adding Elements to an Array

You can add elements to the **start** of an array by using the spread syntax.

```
const [colors, setColors] = useState(['red', 'green']);

const addColor = (colorToAdd) => {
  const updatedColors = [colorToAdd, ...colors];
  setColors(updatedColors);
};
```

Add elements to the **end** of an array by reversing the order of elements in `updatedColors`.

```
const [colors, setColors] = useState(['red', 'green']);

const addColor = (colorToAdd) => {
  // Now 'colorToAdd' will be at the end
  const updatedColors = [...colors, colorToAdd];
  setColors(updatedColors);
};
```

Elements can be added at any index by using the `slice` method available on all arrays.

```
const [colors, setColors] = useState(['red', 'green']);

const addColorAtIndex = (colorToAdd, index) => {
  const updatedColors = [
    ...colors.slice(0, index),
    colorToAdd,
    ...colors.slice(index),
  ];
  setColors(updatedColors);
};
```

The `slice` method can be used to add elements at the start or end of an array as well.

## Removing Elements From An Array

Elements can be removed from an array by using the `filter` method.

The `filter` method can remove elements by index.

```
const [colors, setColors] = useState(['red', 'green', 'blue']);

const removeColorAtIndex = (indexToRemove) => {
  const updatedColors = colors.filter((color, index) => {
    return index !== indexToRemove;
  });

  setColors(updatedColors);
};
```

`filter` can also remove elements by value.

```
const [colors, setColors] = useState(['red', 'green', 'blue']);

const removeValue = (colorToRemove) => {
  const updatedColors = colors.filter((color) => {
    return color !== colorToRemove;
  });

  setColors(updatedColors);
};
```

## Changing Elements

Objects in an array can be modified by using the `map` function.

```
const [books, setBooks] = useState([
  { id: 1, title: 'Sense and Sensibility' },
  { id: 2, title: 'Oliver Twist' },
]);

const changeTitleById = (id, newTitle) => {
  const updatedBooks = books.map((book) => {
    if (book.id === id) {
      return { ...book, title: newTitle };
    }

    return book;
  });

  setBooks(updatedBooks);
};
```

## Changing Properties In Objects

Properties in an object can be changed or added by using the spread syntax (the `...`).

```
const [fruit, setFruit] = useState({
  color: 'red',
  name: 'apple',
});

const changeColor = (newColor) => {
  const updatedFruit = {
    ...fruit,
    color: newColor,
  };

  setFruit(updatedFruit);
```

## Removing Properties In Objects

Properties in an object can be removed by using destructuring.

```
const [fruit, setFruit] = useState({
  color: 'red',
  name: 'apple',
});

const removeColor = () => {
  // `rest` is an object with all the properties
  // of fruit except for `color`.
  const { color, ...rest } = fruit;

  setFruit(rest);
};
```