Arrays and Objects as State

`useState()` can manage any type of JS value

- including arrays and objects

React assumes the state only changes when you call the setter function.

- This means arrays and objects can be a problem
- You can **mutate** these by changing an element/property
    - without calling the setter function
- This would confuse React

Solution: Don't do that

Treat arrays and objects in state as **immutable**

- No React confusion

But how do you change the state?

- pass a NEW array/object to the setter function

Updating array in state example

```
function SomeComponent() {
  const [todos, setTodos] = useState(['nap','sleep','rest']);
  const [newTodo, setNewTodo] = useState('');
  return (
    <div>
      <TodoList list={todos}/>
      <input
        value={newTodo}
        onInput={ (e) => setNewTodo(e.target.value) }
      />
      <button
        onClick={ () => setTodos( [...todos, newTodo]);
      />
    </div>
  );
}
```

Setting a new array

- Setting the state to a new array using `[]`
- Using the **spread** operator (`...`)
    - fills new array with contents of existing array
    - copies array

https://beta.reactjs.org/learn/updating-arrays-in-state

Replacing array mutations for state update

Changing an element:

- DO NOT set the element to a new value
- DO copy the array, change copy, set state to copy

Adding an element:

- DO NOT use `.push()` or `.unshift()`
- DO use spread (`...`) or `.slice()` (to copy array)

Removing an element:

- DO NOT use `.pop()` or `.shift()`
- DO use `.slice()` or alter a copy

Updating object in state example

```
function SomeOtherComponent() {
  const [student, setStudent] = useState({
    name: 'Jorts', grade: '87'
  });
  const [grade, setGrade] = useState(student.grade);
  return (
    <div>
      <div>Name: {student.name}</div>
      <div>
        Grade:
        <input
          value={grade}
          onInput={ (e) => setGrade(e.target.value) }
        />
      </div>
      <button
        onClick={ () => setStudent( {...student, grade });
      />
    </div>
  );
}
```

Setting a new object

- Setting the state to a new object using `{}`
- Using the **spread** operator (`...`)
    - fills new object with existing object contents
    - copies object

https://beta.reactjs.org/learn/updating-objects-in-state

Any key/value pairs after spread op override key/values in copied object

More about Object copying

```
onClick={ () => setStudent( {...student, grade });
```

Remember this is the same as saying:

```
onClick={ () => setStudent({
  ...student,
  grade: grade,
});
```

`grade` property gets the value of the `grade` variable

- and here, overrides any `grade` key/value pair in `student`

Replacing object mutations for state update

Changing an element:

- DO NOT set the element to a new value
- DO copy the object, change copy, set state to copy

Adding an element:

- DO NOT define the new property value
- DO use spread (`...`) (to copy object)

Removing an element:

- DO NOT use `delete` on object property
- DO alter a copy, set new state as copy

This can feel daunting

But the rules itself is straight-forward

- Do not change an array/object that is in state
- set state to a new array/object
    - that was set from the existing array/object
    - and has the changes