Managing Component State

Terms

Asynchronous
Lifting state
Pure component
Strict mode

Summary

- The state hook allows us to add state to function components.
- Hooks can only be called at the top level of components.
- State variables, unlike local variables in a function, stay in memory as long as the component is visible on the screen. This is because state is tied to the component instance, and React will destroy the component and its state when it is removed from the screen.
- React updates state in an asynchronous manner, so updates are not applied immediately. Instead, they're batched and applied at once after all event handlers have finished execution. Once the state is updated, React re-renders our component.
- Group related state variables into an object to keep them organized.
- Avoid deeply nested state objects as they can be hard to update and maintain.
- To keep state as minimal as possible, avoid redundant state variables that can be computed from existing variables.
- A pure function is one that always returns the same result given the same input. Pure functions should not modify objects outside of the function.

- React expects our function components to be pure. A pure component should always return the same JSX given the same input.

- To keep our components pure, we should avoid making changes during the render phase.

- Strict mode helps us catch potential problems such as impure components. Starting from React 18, it is enabled by default. It renders our components twice in development mode to detect any potential side effects.

- When updating objects or arrays, we should treat them as immutable objects. Instead of mutating them, we should create new objects or arrays to update the state.

- Immer is a library that can help us update objects and arrays in a more concise and mutable way.

- To share state between components, we should lift the state up to the closest parent component and pass it down as props to child components.

- The component that holds some state should be the one that updates it. If a child component needs to update some state, it should notify the parent component using a callback function passed down as a prop.

UPDATING OBJECTS

```
const [drink, setDrink] = useState({
  title: 'Americano',
  price: 5
});

setDrink({ ...drink, price: 2 });
```

UPDATING NESTED OBJECTS

```
const [customer, setCustomer] = useState({
  name: 'John',
  address: {
    city: 'San Francisco',
    zipCode: 94111
  }
});

setCustomer({
  ...customer,
  address: { ...customer.address, zipCode: 94112 },
});
```


UPDATING ARRAYS

```
const [tags, setTags] = useState(['a', 'b']);

// Adding
setTags([...tags, 'c']);

// Removing
setTags(tags.filter(tag => tag !== 'a'));

// Updating
setTags(tags.map(tag => tag === 'a' ? 'A' : tag));
```

UPDATING ARRAY OF OBJECTS

```
const [bugs, setBugs] = useState([
  { id: 1, title: 'Bug 1', fixed: false },
  { id: 2, title: 'Bug 2', fixed: false },
]);

setBugs(bugs.map(bug =>
          bug.id === 1 ? { ...bug, fixed: true } : bug));
```

UPDATING WITH IMMER

```
import produce from 'immer';

setBugs(produce(draft => {
  const bug = draft.find(bug => bug.id === 1);
  if (bug) bug.fixed = true;
}));
```

