Data flow in React

You've just learned how the parent-child relationship can be set up so that data flows from parent to child.

In this reading, you'll learn how to detail the flow of data from parent to child. You will then learn why code samples need to be clear and concise. Finally, you will explore data flow in greater detail by looking at more examples. This should act as a refresher to knowledge gained in previous courses.

Parent-child data flow

In React, data flow is a one-way street. Sometimes it's said that the data flow is unidirectional. Put differently, the data in React flows from a parent component to a child component. The data flow starts at the root and can flow to multiple levels of nesting, from the root component (parent component) to the child component, then the grandchild component, and further down the hierarchy.

A React app consists of many components, organized as a component tree. The data flows from the root component to all the components in the tree structure that require this data, using props.

Props are immutable (cannot be changed).

The two main benefits of this unidirectional data flow are that it allows developers to:

1. comprehend the logic of React apps more quickly and
2. simplify the data flow.

Here's a practical example of this:

Imagine that the parent component passes a prop (name) to the child component. The child component then uses this prop to render the name in the UI.

Parent component:

```
function Dog() {
    return (
        <Puppy name="Max" bowlShape="square" bowlStatus="full" />
    );
};
```

Child component:

```
function Puppy(props) {
    return (
        <div>
            {props.name} has <Bowl bowlShape="square" bowlStatus="full" />
        </div>
    );
};
```

Grandchild component:

```
function Bowl(props) {
    return (
        <span>
            {props.bowlShape}-shaped bowl, and it's currently {props.bowlStatus}
        </span>
    );
};
```

Having data move through props in only one direction makes it simpler to understand the logic of how the components interact. If data were moving everywhere, all the time, then it would be much harder to comprehend its logical flow. Any optimization you tried to implement would likely not be as efficient as it could be, especially in modern React.