The useState Hook in React

React has two types of components, functional and class.

Class components extend from [React.Component] and have state and lifecycle methods. They look like this:

```
Class Example extends React.Component{
    constructor(props) {
        super(props);
        this.state = {
            text: ""
        };
    }
    componentDidMount(){
        //Do something when the component mounts.
    }
    render(){
        return <div>{this.state.text}</div>
    }
}
```

While functional components simply accept arguments as component props and return valid JSX:

```
function Example(props) {
  return <div>{props.text}</div>;
}
//Or alternatively
const Example = (props) => {
  return <div>{props.text}</div>;
};
```

The above examples are very similar, except the functional component cannot use state variables. This is where the useState Hook comes into play. Hooks were introduced in React 16.8, they are functions that add lifecycle methods and state variables to functional components. This is exciting because that means we can replace class components with more readable functional components and maintain functionality. This guide is focused specifically on the useState Hook, which allows us to access state variables in a functional component. Hooks should start with `use`, to make them easy to identify.

Get and Set State with useState

The useState Hook allows you to add get state variables and set state variables in functional components. While state in a class is always an object, state with Hooks can be any type. Each piece of state can hold just a single value.

To import the useState Hook, simply type:

```
import React, { useState } from "react";
```

The useState Hook always returns an array where the first element is the state variable and the second is a function that updates the value of the variable.

```
const Example = () => {
  const exampleState = useState("");
  const example = exampleState[0]; //This is the value ''
```

```
  const setExample = exampleState[1]; // This is a function
};
```

It is good practice to use array destructuring with useState to make it simpler and more readable:

```
const Example = () => {
  const [example, setExample] = useState("");
  //Now you can access the value with example, and update it with setExample.
  return (
    <div>
      <input
        type={"text"}
        value={example}
        onChange={(e) => {
          setExample(e.target.value);
        }}
      />
      <p>{example}</p>
    </div>
  );
};
```

useState rules

All Hooks have to follow two rules.

- Only call Hooks at the Top Level
- Only call Hooks from React Functions

The first rule means that you cannot use Hooks inside conditionals, loops or nested functions because React relies on the order useState Hooks are called to get values correctly. The second rule means that you cannot call Hooks in class components or regular JavaScript functions. If you violate either of these rules you'll get an error.

```
const GoodExample = () => {
  const [good, setGood] = useState("This is a good example.");
};
const BadExample = () => {
  const handleBad = () => {
    const [badExample, setBadExample] = useState(
      "Don't call Hooks inside nested functions."
    );
  };
  if (badExample) {
    const [anotherBadExample, setAnotherBadExample] = useState(
      "Don't call Hooks inside conditionals or loops."
    );
  }
};
```

A functional component can have many calls to useState. When the component is rendered for the first time and useState is executed, the initial value is read. However, if the value is changed by the set function, subsequent renders of the component use the updated state value.

Conclusion

The useState Hook allows you to use state variables in functional components. This means that you can replace class components with more readable functional components while keeping state variables. The easiest way to write them is like this:

```
import React, { useState } from "react";
const Example = () => {
  const [example, setExample] = useState("");
};
```

All Hooks have to follow the same two rules, and there's an ESLint plugin to help enforce them.

Below is an example of useState Hooks using a variety of types:

```
const Example = () => {
  const [string, setString] = useState("This is a string example.");
  const [numberExample, setNumber] = useState(0);
  const [arrayExample, setArray] = useState([]);
  const [objectExample, setObject] = useState({});
  const [boolExample, setBool] = useState(false);
  const [nullExample, setNull] = useState(null);
};
```