React Hooks Will Definitely Hook You In

React has become a staple for many web developers around the globe. Mainly because using the react libraries makes our JavaScript code looks nicer all while making our code more modular. Developers have been using classes to create their products for the longest time. However, a couple of years ago (mainly in 2019) React introduced a new mechanic called "Hooks." This new tool was released to the public in version 16.8 as a better solution to classes by making use of these little things called "states." Using these new states helped developers prevent methods or functions reaching lines of the double digits, all while improving functionality in the process.

Introducing hooks also introduced (more like brought more attention to) states. Which are the keys to success when using hooks. States are a lot like literal triggers or switches. States determine how your components are rendered and ties into the coupling (object-oriented term for "the degree of interdependence between software") for of your overall code.

Two the main hooks that developers would use are useState() and setState(). useState() returns a state value as well as a function to update it. An example is shown below:

```
const [state, setState] = useState(initialState);
```

During the render sequence, the state that's returned (state) is the same value that was first passed in as an argument (initial state), according to the React API.

setState() is a function that is used to update the current state. It accepts a new state value and enqueues a re-render of the component according to the React API. An example is shown below:

```
setState(newState);
```

The value returned by useState() will always be the most recent state after applying updates, according to the React API.

Why You Should Use Hooks Over Classes

Now that we have the essentials of hooks taken care of, why should we use Hooks over classes? One things for sure is that classes can get pretty bulky and reimplementing and reformatting classes can be mind-numbingly tedious for most. Introducing hooks not only decrease the line count for your code tremendously, but it also makes the code more cohesive and readable by eliminating a lot of the tedium.

Functional components don't have to be refactored into a class component as the application develops. React components usually start off depending on properties and then become classes with a having state. Functional components have the ability of tapping into the state. This means that the developer can use hooks within the functional component. Below is an example of a component that shows a label with count, courtesy of Dilantha Prasanjith from her blog on "Bits and Pieces":

```
export function ShowCount(props) {
  return (
    <div>
      <h1> Count : {props.count} </h1>
    </div>
  );
}
```

The component will now be tracked through mouse clicks. This will only affect the current component. Now all that's left is to introduce our state to the component with this **class based** approach (provided by Dilantha Prasanjith):

```
export class
ShowCount
extends
React.Component
{
                    constructor(props) {
                      super(props);
                      this.state = {
                        count: 0
                      };
                    }
                    componentDidMount() {
                      this.setState({
```

```
      count: this.props.count
    })
  }

  render() {
    return (
      <div>
        <h1> Count : {this.state.count} </h1>
      </div>
    );
  }

}
```

This is what the code looks like after using hooks instead of classes (provided by Dilantha Prasanjith):

```
export function ShowCount(props) {
  const [count, setCount] = useState();

  useEffect(() => {
    setCount(props.count);
  }, [props.count]);

  return (
    <div>
      <h1> Count : {count} </h1>
    </div>
  );
}
```

Looks a lot cleaner doesn't it? `useEffect()` is another hook used with this example. All it's doing is that it's calling `setCount` to track the number of clicks that are being called into the component.

React Hooks allow you to not use the keyword "this" anymore. "This" can be very vague when it comes to class implementation and expository use of the developers code. The keyword usually introduces confusion because its meant to tell the renderer or the compiler to only use the instance of the method or function that is within a specific class, or "this" class to be more exact. The example below shows the difference between Hooks (left) and classes (right) for the count example (provided by Dilantha Prasanjith):

```jsx
export function ShowCount(props) {
  const [count, setCount] = useState();

  useEffect(() => {
    setCount(props.count);
  }, [props.count]);

  return (
    <div>
      <h1> Count : {count} </h1>
    </div>
  );
}
```

```jsx
export class ShowCount extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }
  componentDidMount() {
    this.setState({
      count: this.props.count
    })
  }

  render() {
    return (
      <div>
        <h1> Count : {this.state.count} </h1>
      </div>
    );
  }

}
```

It's Easier to decouple logic from UI. Thanks to hooks, logic and UI can be separated easily. This eliminates the use for render properties and HOCs (higher-order components). This helps a lot when developers share their components through other websites for others to use. Using open-source tools is as easy to implement as cloning a repository to your desktop. This is better than using classes because of the refactor explanation before, having hooks in the code rather than classes would allow the developer to slot in open-source code like one Lego brick to another. If we used open-source classes, implementing the class would take a lot of variable changing and class management that might be more tedious than writing the code from scratch.

All in all, React Hooks makes code a lot more cohesive and flexible for the average developer. Doing away with the "this " keyword goes a long way as well as making the code more modular for open-source use. Classes would require way more specifications and maintenance if a new function were to be introduced as well as double-digits worth of code in order to perform a simple task, while hooks (with a little understanding) will go a long way towards efficiency.

Sources:

Prasanjith, Dilantha. “6 Reasons to Use React Hooks Instead of Classes.” *Medium*, Bits and Pieces, 14 Sept. 2020, https://blog.bitsrc.io/6-reasons-to-use-react-hooks-instead-of-classes-7e3ee745fe04.

SamarthNehe. “React Hooks vs Class Components.” *Medium*, Medium, 26 May 2021, https://samarthnehe.medium.com/react-hooks-vs-class-components-c344b59f3bc.

“Hooks API Reference.” *React*, https://reactjs.org/docs/hooks-reference.html#usestate.