HOOKS

Up to now, functional components haven't been able to have state (well, at least not how we've been writing them). One of the upside to hooks is that if you write a functional component, you no longer need to refactor your entire functional component to a class! You can just `useState`!

setState

Previously, if you wanted to have a counter (like we did with our previous class components), you'd have to write a lot of boiler plate:

```
export class ReactClass extends React.Component {
  constructor(props) {
    super(props);

    this.state = {
      timesPressed: 0
    };

    this.handleAction = this.handleAction.bind(this);
  }

  handleAction(action) {
    console.log("Handling the action! ", action);

    this.setState({
      timesPressed: this.state.timesPressed + 1
    });

    console.log("TIMES PRESSED? ", this.state.timesPressed);
  }

  render() {
    return (
      <>
        <button onClick={this.handleAction}> Press ME! </button>
        <div>That dang button got clicked {this.state.timesPressed} times
</div>
      </>
    );
```

```
  }
}

```

However, when you get to use hooks, you can quite easily just write a functional component:

```
import React from "react";

export const FunctionComponent = () => {
  const [clickedTimes, setClickedTimes] = React.useState(0);

  const handleClick = () => {
    setClickedTimes(clickedTimes + 1);
  };

  return (
    <>
      <button onClick={handleClick}> Press ME! </button>
      <div>That dang button got clicked {clickedTimes} times </div>
    </>
  );
};

```

First and foremost, all hooks in React start with `use`. The `useState` hook takes in an initial state value as its parameter. What the function returns is 2 elements to an array that we destructured as `[clickedTimes, setClickedTimes]`. `clickedTimes` is the state variable, and `setClickedTimes` is the function we use to change that specific potion of state.

One potential downside to `useState` is that you're specifically accessing a single instance of state, and you can't `setState` and overwrite multiple portions of state. You'll need to use a lot of functions.

Ultimately, what's happening behind the scenes is that React creates an object to live alongside the function component to keep track of the component.