MISSION READY

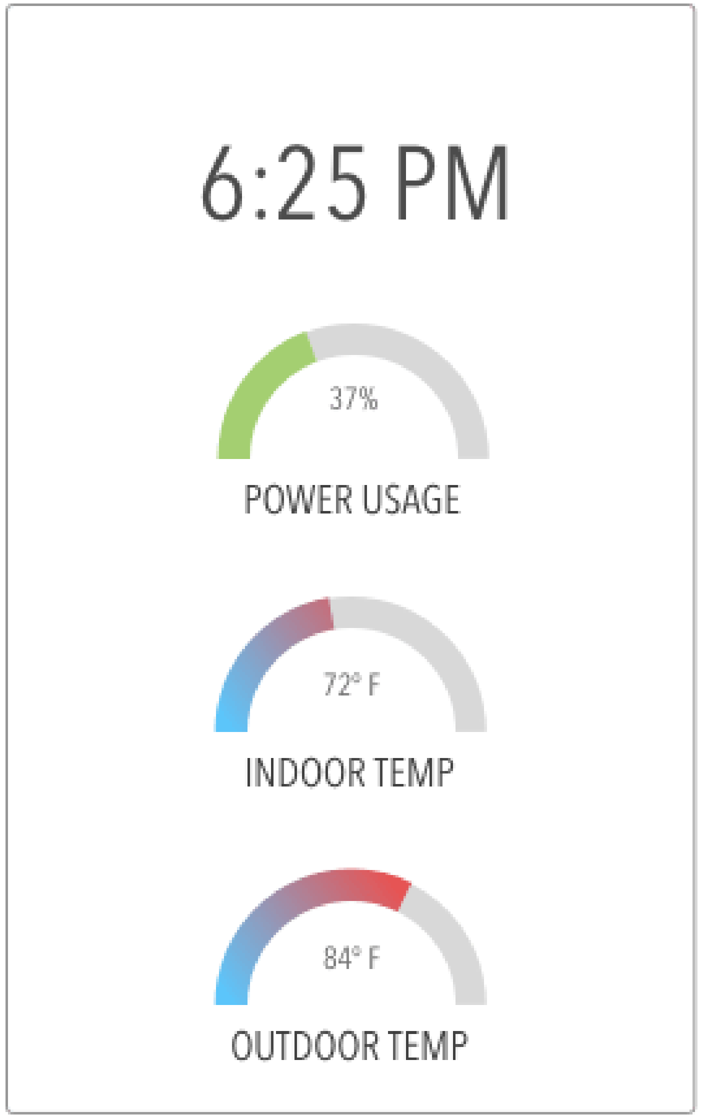
DARE TO **DEVELOP**

React State and Hooks

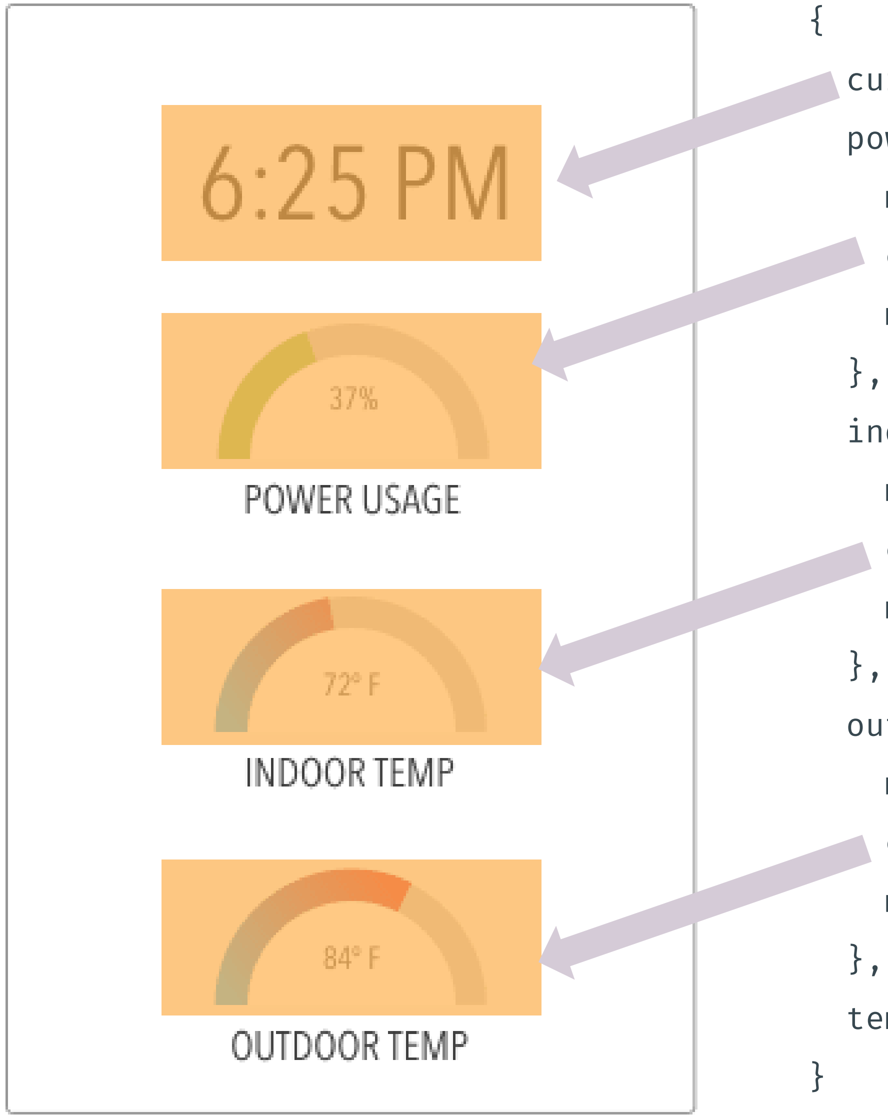
Reuben Simpson

State

- "state" in react is an object that represents the parts of the app that can change.
  - The state is managed within the React component and can be changed to update the information in the DOM.
  - Each component can maintain its own state
- State can change as we interact with the component.
- In functional components we can use the ***useState*** hook to change the value of the state.



Parts that could change over time

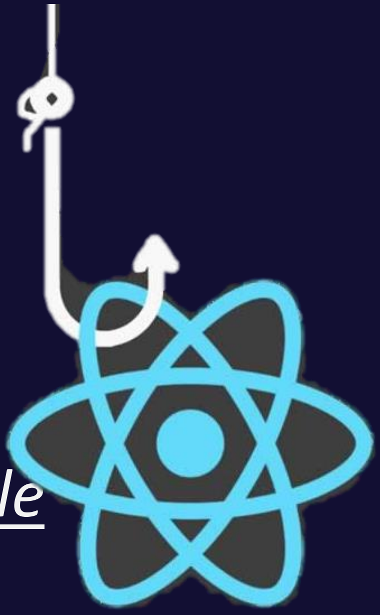


A representation of the state of the app

```
{
  currentTime: "2016-10-12T22:25:42.564Z",
  power: {
    min: 0,
    current: 37,
    max: 100
  },
  indoorTemperature: {
    min: 0.0,
    current: 72.0,
    max: 100.0
  },
  outdoorTemperature: {
    min: -10.0,
    current: 84.0,
    max: 120.0
  },
  tempUnits: "F"
}
```

You *change the state* to *change* how the app *looks*.

What are React Hooks?



- React is a library for building user interfaces.
- Hooks are ***functions*** that let you “hook into” React *state* and *lifecycle* features from function components
- *Hooks* are a new addition in React 16.8. They let you use state and other React features without writing a class.
- Hooks are JavaScript functions, but they impose two additional rules
  - Only call Hooks at the ***top level***.
    - Don’t call Hooks inside loops, conditions, or nested functions.
  - Only call Hooks from ***React function components***. Don’t call Hooks from regular JavaScript functions.

https://reactjs.org/docs/hooks-overview.html

React Hooks (History)

- Hooks are new built-in functions in React that lets you use state and other React features ***without writing a class***.
  - Officially in early February 2019
- Basic Hooks
  - useState
  - useEffect
- **useEffect** replaces ***componentDidMount***, ***componentDidUpdate***, and ***componentWillUnmount*** with a unified API.

https://reactjs.org/docs/react-component.html#componentdidmount

Additional Hooks

- useContext
- useReducer
- useCallback
- useMemo
- useRef
- useImperativeHandle
- useLayoutEffect
- useDebugValue

https://reactjs.org/docs/hooks-reference.html

Functional Component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

Class Component

```
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

useState hook

- Usable in function components to add some local state to it.

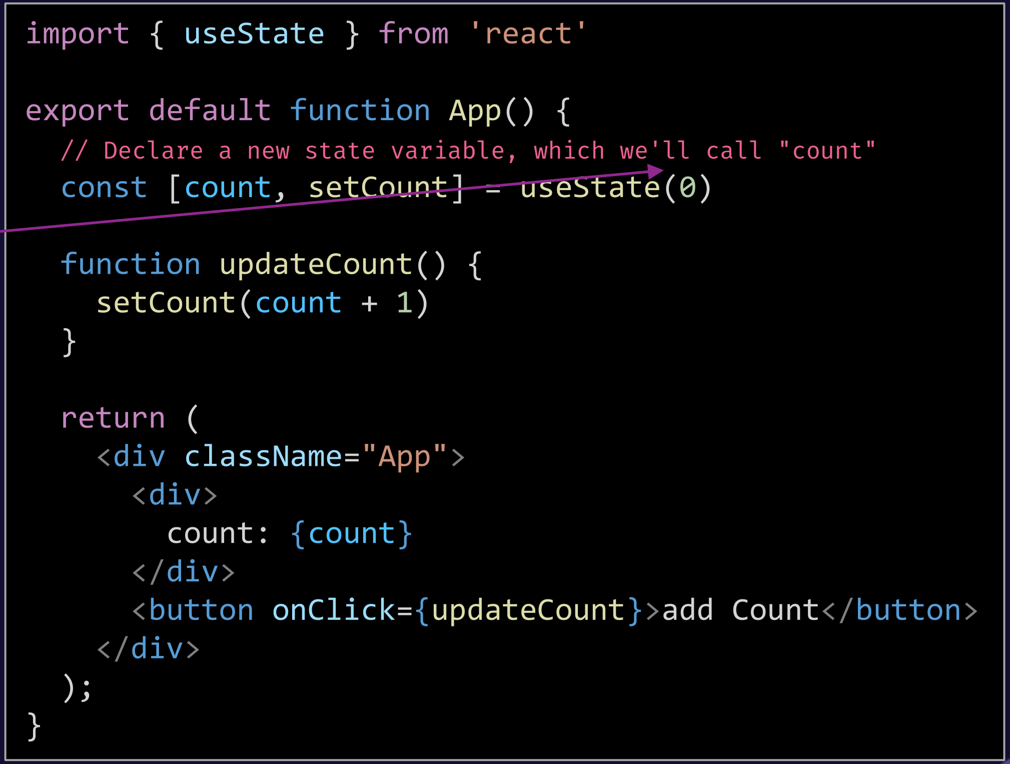
```
   const [state, setState] = useState(initialState);
```

- useState returns a pair of values
  - A stateful ***value*** (state)
  - A ***function*** that lets you update it (setState)
- Only argument to useState is the initial state.
  - The initial state argument is only used during the first render.

Counter example

- Let’s look at this simple app that increments a counter in our browser

- Import useState
- Declare count and setter
- Set initial value
- Create function to update counter
- Show count in the browser
- Create button that executes updateCount



```
import { useState } from 'react'

export default function App() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0)

  function updateCount() {
    setCount(count + 1)
  }

  return (
    <div className="App">
      <div>
        count: {count}
      </div>
      <button onClick={updateCount}>add Count</button>
    </div>
  );
}
```

https://reactjs.org/docs/hooks-state.html

Exercise 1

- Create an additional counter and button that starts at 100 and subtracts by 1 whenever a button is pressed.

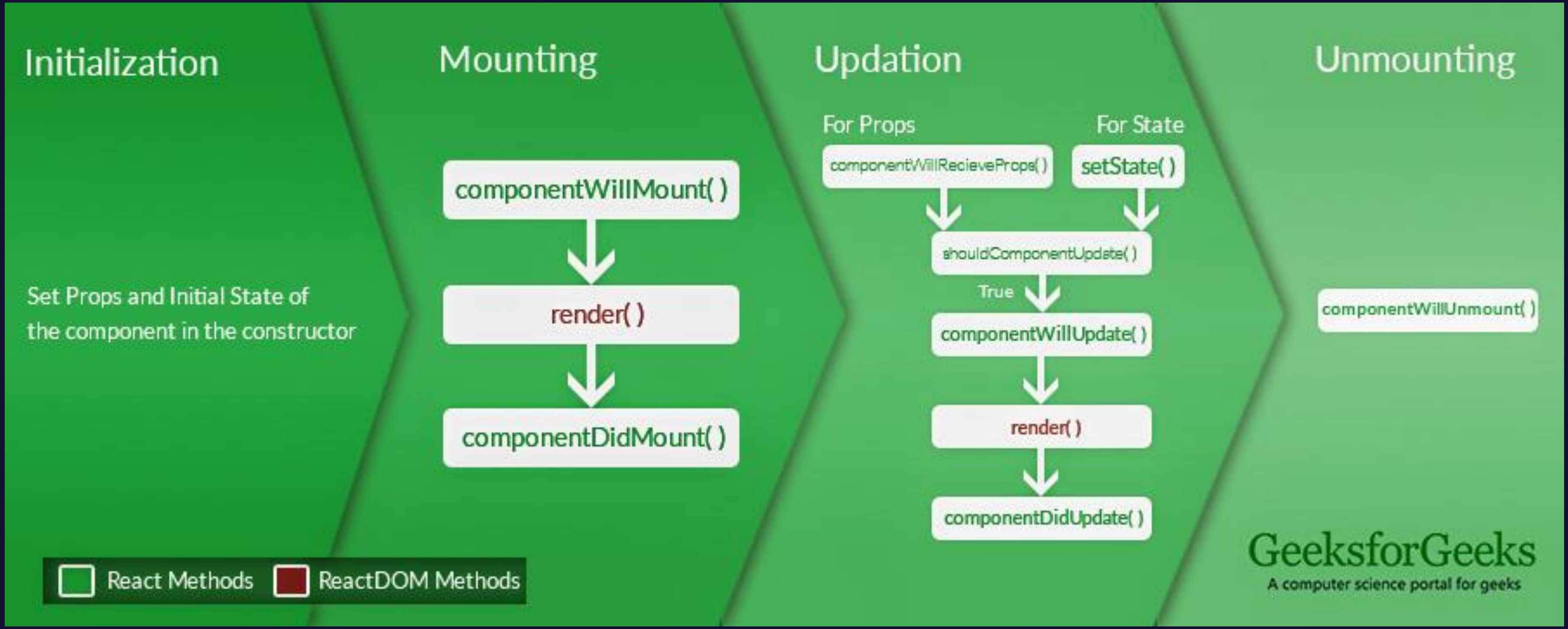
useEffect hook

- The useEffect hook allows us to perform an action (side effects) any time there is a change to the state/props.
    - Some examples of side effects are: fetching data (APIs), timers.
- It takes in two parameters
    - A ***function*** that will run whenever the dependencies (if any) are changed
    - A set of ***dependencies***, that allow the function to run only when one of the dependencies is changed
- You can use multiple useEffect statements in your component.

https://dev.to/colocodes/6-use-cases-of-the-useeffect-reactjs-hook-282o

Life cycle phases of a react component

- **Mounting** that is putting inserting elements into the DOM.
- **Updating**, which involves methods for updating components in the DOM.
- **Unmounting**, that is removing a component from the DOM.



https://www.geeksforgeeks.org/reactjs-lifecycle-components/

useEffect Hook – the syntax

```
1. useEffect(() => {}) – One argument
2. useEffect(() => {}, []) – Two arguments
```

`() => {}` is the ***mandatory*** function that will run when the Hook gets activated.

`[]` is an optional set of dependencies which decide when the hooks gets activated.

When does `useEffect` hook Run?

1. ***After every render***
   - This is if you do NOT pass the second argument.
   - Example, if a user is composing a message, a copy of the draft to the server.

```
useEffect(() => {
        // put ‘every update' code here
        });
```

When does `useEffect` hook Run?

2. *On state change*

- Include an ***array*** of all the ***state variables*** to be watched as the second argument.
- Example, to validate your input field, live filtering of a list (https://mui.com/components/autocomplete/#country-select).

```
function YourComponent() {
  const [state, setState] = useState();
  useEffect(() => {
    // code to run when state changes
  }, [state]);
}
```

When does `useEffect` hook Run?

3. ***Once on mounting***

   - If you pass an ***empty array*** as the second argument.
   - Example, to fetch API data

```
useEffect(() => {
        // put 'run once' code here
        }, []);
```

When does useEffect hook Run?

4. *On props change*

- Include an ***array*** of all ***props*** to be monitored as the second argument.
- Example, if a fetched API result is updated in a parent element. Or, if an API needs to be called based on the parent data change.

```
function YourComponent({ someProp }) {
  useEffect(() => {
    // code to run when someProp changes
  }, [someProp]);
}
```

When does `useEffect` hook Run?

5. *On unmount*

- If you do NOT pass the second argument and return a cleanup function.

```
useEffect(() => {
  return () => {
    // put unmount code here
  };
});
```

useEffect – Example 1

- We can now add a useEffect to our counter example that will log something to the console, every time the state is changed (the count is updated)

```
import { useState, useEffect } from 'react'

export default function App() {
  const [count, setCount] = useState(0)

  useEffect(() => {
    console.log(`the count is at ${count}`)
  })

  function updateCount() {
    setCount(count + 1)
  }

  return (
    <div className="App">
      <div>
        count: {count}
      </div>
      <button onClick={updateCount}>add Count</button>
    </div>
  );
}
```

useEffect – Example 1 continued

- Let’s add a second state variable. This variable is going to **sum** up the total of all the count values so far
- E.g.

button pressed once:	count = 1, sum = 1
button pressed twice:	count = 2, sum = 3
button pressed 3 times:	count = 3, sum = 6 …

- But useEffect runs every time the state changes, so now we are in an infinite loop… Let’s look at how to fix that

```
const [count, setCount] = useState(0)
const [sum, setSum] = useState(0)

useEffect(() => {
  console.log(`the count is at ${count}`)
  setSum(sum + count)
  console.log(`the sum is at ${sum}`)
})
```

useEffect – Example 1 continued...

- To fix the infinite loop we can add a dependency to the useEffect function.
- We add the dependency *count* so useEffect only runs when the count is updated and not when the sum is updated

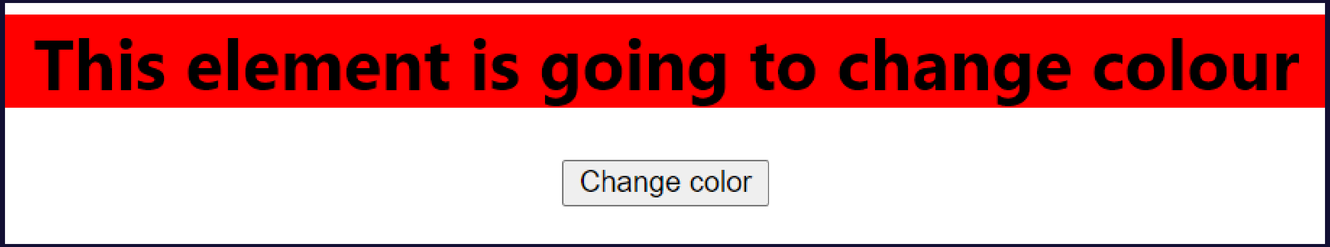
```
const [count, setCount] = useState(0)
const [sum, setSum] = useState(0)

useEffect(() => {
  console.log(`the count is at ${count}`)
  setSum(sum + count)
  console.log(`the sum is at ${sum}`)
}, [count])
```

Color changer

- Let’s look at another example that uses these hooks
- Let’s add an element in our JSX that has a set background color
- Then, we can add a button that will change the background color when it is clicked

```
return (
    <div className="App">
      <h1 style={{background: color}}>This element is going to change colour</h1>
      <button onClick={changeColor}>Change color</button>
    </div>
  );
```


This element is going to change colour

Color changer continued…

- First, we can import useState from react and declare our variable and the setter

```
const [colour, setColour] = useState("red")
```

- Then, we can create the changeColor function to change our color on each button press

```
function changeColour() {
    switch (colour) {
      case "red":
        setColour("blue")
        break;
      case "blue":
        setColour("green")
        break;
      case "green":
        setColour("orange")
        break;
      case "orange":
        setColour("yellow")
        break;
      default:
        setColour("red")
    }
  }
```

Exercise 2

- Add a useEffect hook to the colour changer app, that logs the colour of the h1 to the console, whenever it is changed, be sure to add the appropriate dependencies

- Example: "colour of the h1 tag changed to {colour}"

MISSION READY
www.missionreadyhq.com

DARE TO
DEVELOP

Thank you | Reuben Simpson