Welcome Geeks

Topics to be covered

- React Basics: Components with States & Props
- Class Components vs. Functional Components
- Passing Data between Components
- Event Handling
- Setting States and Props
- Loading Data into State

Components

- Building blocks of any React App
- Lets you split the UI into independent, reusable pieces.
- Components are like JS functions. They accept arbitrary inputs (props) and return React Elements

Note: Always start components with a capital letter

Class Components

```
class Car extends React.Component {
  render() {
    return <h2>Hi, I am a Car!</h2>;
  }
}
```

```
class Car extends React.Component {
  render() {
    return <h2>I am a {this.props.color} Car!</h2>;
  }
}
```

Functions vs Class Components

- Before React 16.8, Class components were the only way to track state and lifecycle on a React component. Function components were considered "stateless".

- With the addition of Hooks, Function components are now equivalent to Class components. You will never need to use a Class component in React.

- Even though Function components are preferred, there are no current plans on removing Class components from React.

Props

When React sees an element representing a user-defined component, it passes JSX attributes to this component as a single object which is called props.

```
function Welcome (props) {
  return <>
                <h1>Hello, {props.name}</h1>;
                <h2>Hello, {props.batch}</h2>;
          </>
}

<Welcome name=”geeks” batch=”22” />
```

Props are read only

Whether you declare a component as a function or a class, it must never modify its own props.

```
function sum(a, b) {
  return a + b;
}

function withdraw(account, amount) {
  account = account - amount;
}
```

State allows React components to change their output over time in response to user actions, network responses, and anything else, without violating this rule.

Functional Components

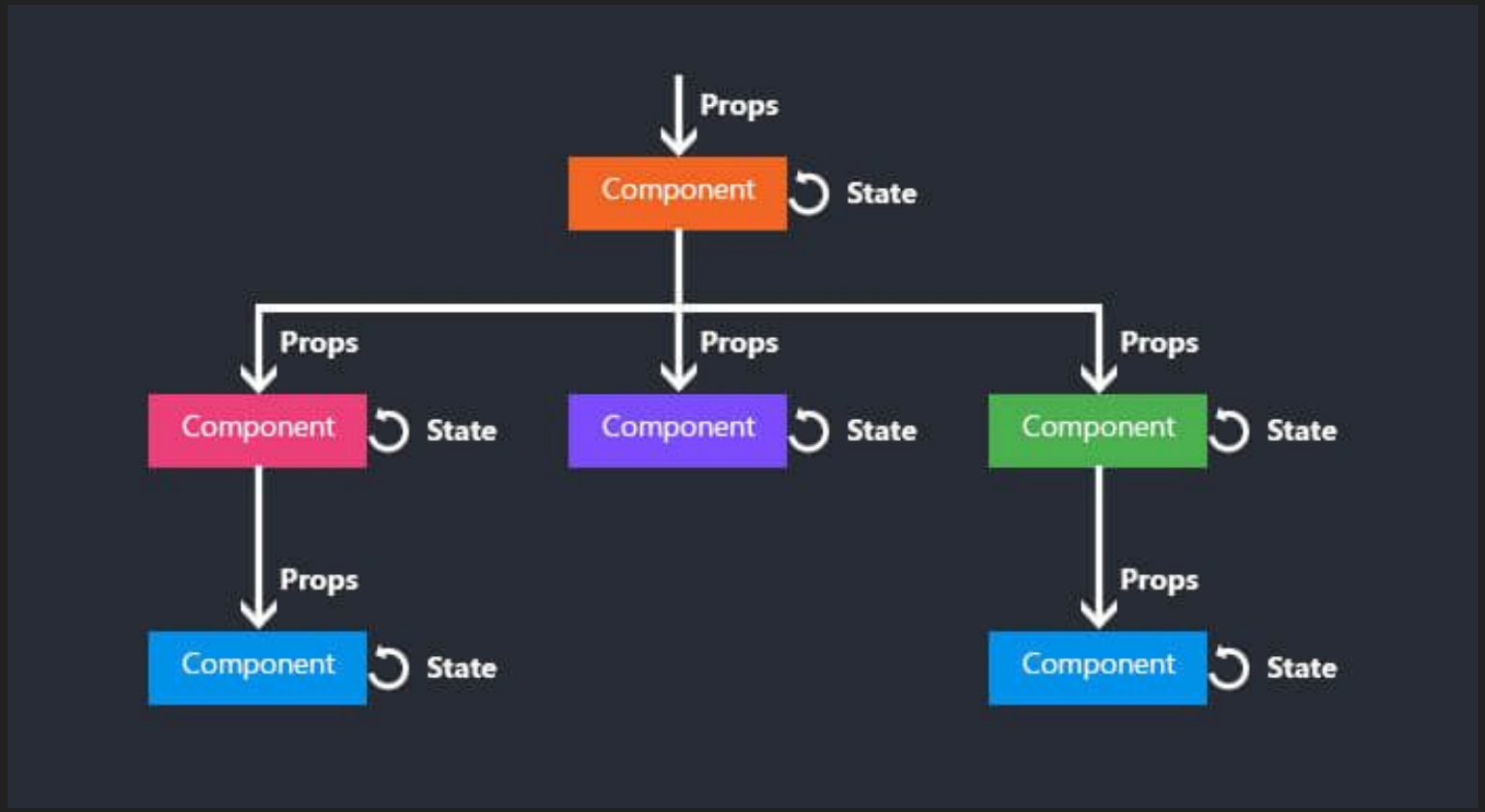
```
function Welcome() {
  return <h1>Hello Geeks</h1>; // React Element (JSX)
}

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

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

State

- React components has a built-in state object.
- The state object is where you store property values that belongs to the component.
- When the state object changes, the component re-renders.


Props
Component
State
Props
Component
State
Props
Component
State
Props
Component
State
Props
Component
State
Props
Component
State

Hooks

- Hooks allow function components to have access to state and other React features. Because of this, class components are generally no longer needed.

- Hooks allow us to "hook" into React features such as state and lifecycle methods.

useState() Hook

useState() - It is a hook that allows you to add React state to functional components.

This returns a pair - the current state value and a function that lets you update it.

React will preserve this state between re-renders.

Declaring state

```
const [text, setText] = useState(“Hello”);

const stateVariable = useState(“Hello”);

const text = stateVariable[0];
const setText = stateVariable[1];
```

Accessing & Updating state

Accessing state

- <h1> {text} </h1>

Updating state

- setText(“Hi”);

useEffect() Hook

- The useEffect Hook allows you to perform side effects in your components.
- Some examples of side effects are: fetching data, directly updating the DOM, and timers.
- useEffect accepts two arguments. The second argument is optional.

useEffect(<function>, <dependency>)

useEffect() Hook

1. No dependency passed:

```
useEffect(() => {
   console.log(“Hello”);
  //Runs on every render
});
```

useEffect() Hook

1. An empty array:

```
useEffect(() => {
  //Runs only on the first render
}, []);
```

useEffect() Hook

3. Props or state values:

```
useEffect(() => {
  //Runs on the first render
  //And any time any dependency value changes
}, [prop, state]);
```