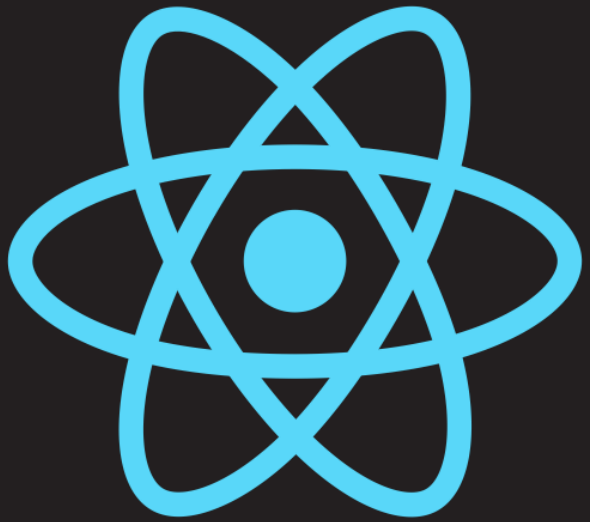
State & Props

What is State?



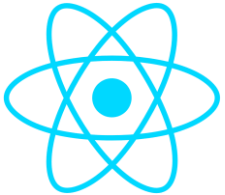
LEARNING OBJECTIVES

- Learn about state which allows you to describe changes to your UI using data
- Learn about the fundamentals of the Hooks API that lets you incorporate state in a Function component
- Learn about the fundamentals of the Hooks API that lets you incorporate state in a Function component
- Learn to debug props using typechecking

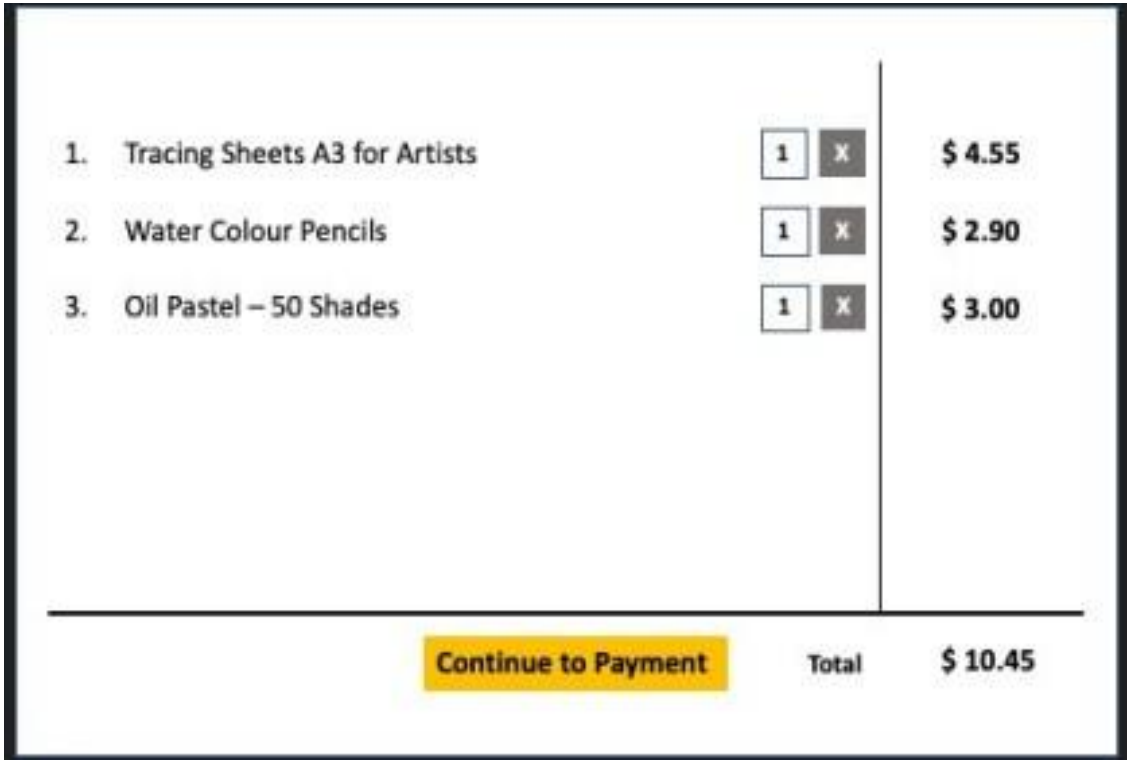
What is State?

DECLARATIVE PROGRAMMING

{...data...}

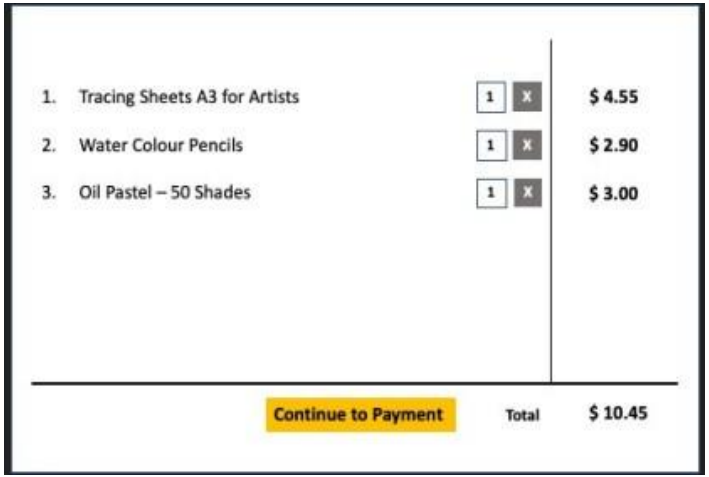


USER INTERFACE FOR A CHECK OUT SCREEN ON AN ONLINE SHOPPING SITE



CHECK OUT SCREEN

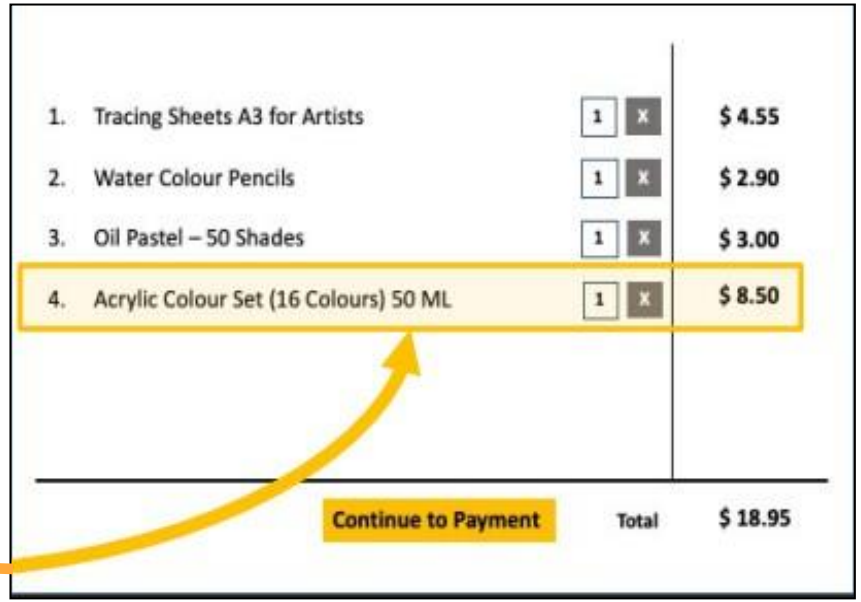
```
cart = [
  {
    product: "Tracing Sheets A3 for Artists",
    quantity: 1,
    cost: 4.55
  },
  {
    product: "Water Colour Pencils",
    quantity: 1,
    cost: 2.9
  },
  {
    product: "Oil Pastels - 50 Shades",
    quantity: 1,
    cost: 3.0
  }
];
```



Array of products with the per unit cost and quantity purchased.
This is the data that drives the interface.

CHECK OUT SCREEN

```
cart = [
  {
    product: "Tracing Sheets A3 for Artists",
    quantity: 1,
    cost: 4.55
  },
  {
    product: "Water Colour Pencils",
    quantity: 1,
    cost: 2.9
  },
  {
    product: "Oil Pastels - 50 Shades",
    quantity: 1,
    cost: 3.0
  },
  {
    product: "Acrylic Colour Set (16 Colours) 50 ML",
    quantity: 1,
    cost: 8.5
  }
];
```



State

Adding another product is as simple as adding the product to the array on the left.
It automatically update the UI on the right.

STATE IN REACT

React uses the concept of state, which essentially describes the contents of the interface.

Whenever the state is changed → Interface automatically updates

You declaratively update the interface by simply updating the underlying state.

STATE IN REACT

✓ It makes easy to imagine application in terms of Data

✗ Not in terms of technical nitty gritties

Document Object Model or DOM manipulation, selectors.

ℹ The UI plays what the state describes.

REACT COMPONENTS

- Class Components
- Function Components

CLASS COMPONENTS

- State management is a built-in feature.
- No Extra Tools needed.

CLASS COMPONENTS

```
class Greet extends Component {
  constructor() {
    super();
    this.state = {
      greeting: "Howdy!"
    };
  }
  render() {
    return <div>{this.state.greeting} partner!</div>;
  }
}
```

Initializing state in the constructor

CLASS COMPONENTS

```
class Greet extends Component {
  state = {
    greeting: "Howdy!"
  };
  render() {
    return <div>{this.state.greeting} partner!</div>;
  }
}
```

Class field syntax for initializing state

Create-react-app based workflow

CLASS COMPONENTS

```
class Greet extends Component {
  state = {
    greeting: "Howdy!"
  };
  render() {
    return <div>{this.state.greeting} partner!</div>;
  }
}
```

State properties are accessible in instance methods including the render method using *this.state.propertyName*

Create-react-app based workflow

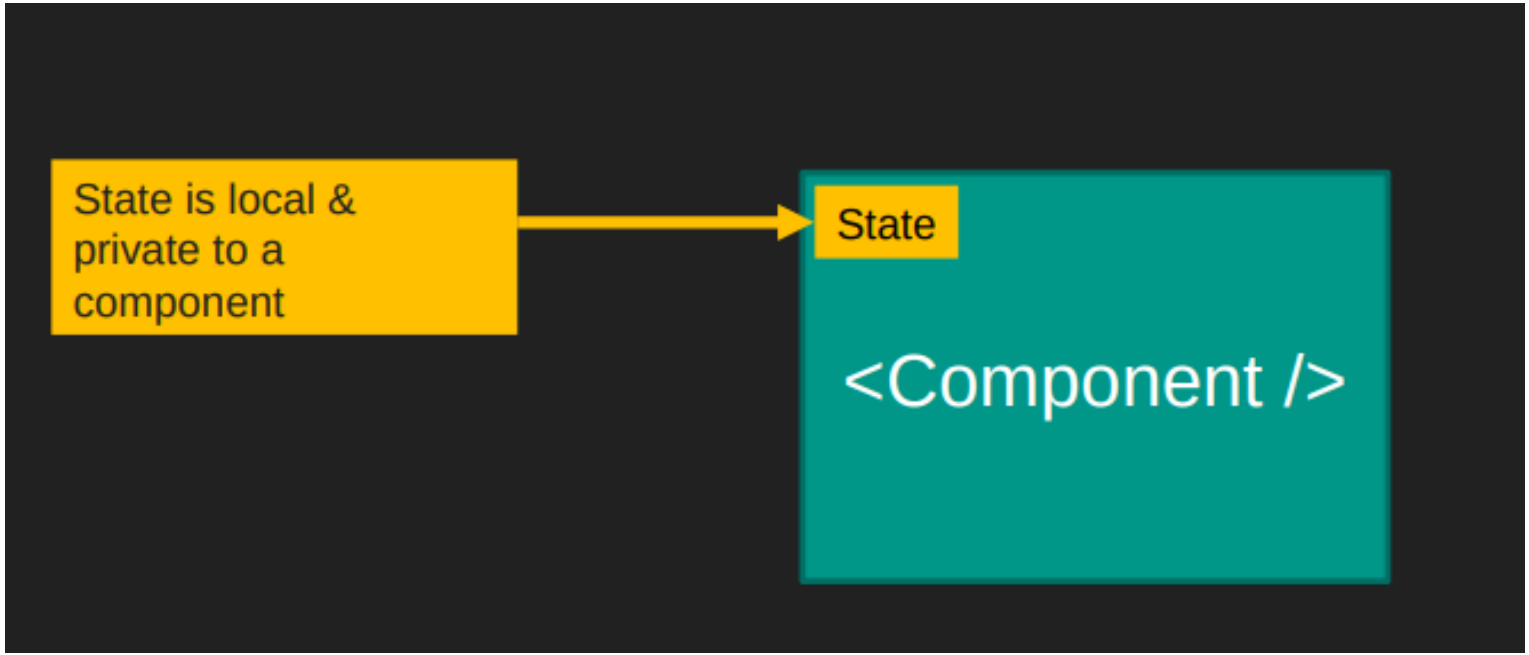
CLASS COMPONENTS

```
class Greet extends Component {
  state = {
    greeting: "Howdy!"
  };
  render() {
    return <div>{this.state.greeting} partner!</div>;
  }
}
```

1. When this updates

2. This updates automatically

CLASS COMPONENTS



Global State Management with Redux

- You can share a component's state with another component is by using props.
- In this case the other component needs to be a child.

CLASS COMPONENTS

```
class Panel extends Component {
  state = {
    isOpen: false
  };
  update = () => {
    this.setState({
      isOpen: !this.state.isOpen
    })
  }
  render() {
    return <div onClick={() => this.update()}>{this.props.children}</div>;
  }
}
```

Use `SETSTATE`() to manipulate state variables

SITUATION: WHERE MULTIPLE INVOCATIONS ARE MADE TO setState

```
class Currency extends Component {
  state = {
    currencyRate: 0
  };
  update = () => {
    this.setState({
      currencyRate: this.state.currencyRate + 2
    });

    this.setState({
      currencyRate: this.state.currencyRate + 20
    });

    this.setState({
      currencyRate: this.state.currencyRate + 30
    })
  };
  render() {}
}
```

- React will batch together such updates
- Do not rely on the value of state to compute the next value of state

CURRENT VALUE OF STATE

What if you do want to rely on the current value of state for computing the next value of state?

CALLBACK NOTATION

```
THIS.SETSTATE((STATE, PROPS) => ({
      CURRENCYRATE: STATE.CURRENCYRATE + 30
}));
```

This callback notation allows to rely on the current value of state or props for computing the next value of state.

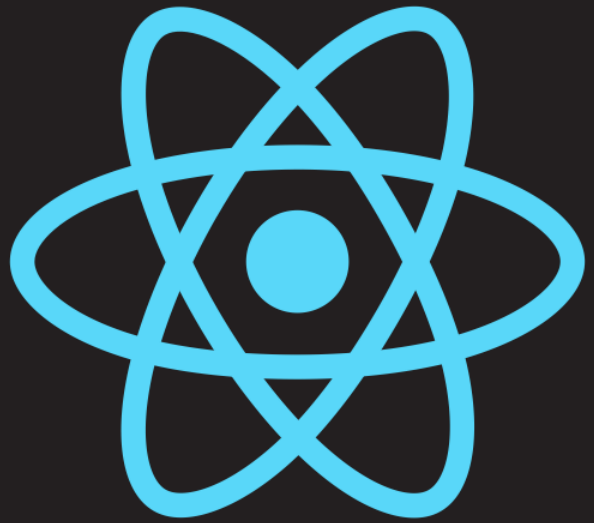
BEST PRACTISES

- Keep state at the nearest logical parent component
- Data can be passed down to child components using props or the Context API
- Keeping state at the nearest parent helps you manage and reason with state data changes logically

Hands-On

State & Props

Stateful
Functions Components with Hooks



FUNCTION COMPONENTS

- Inherently stateless functions with no state management
- Accept data using props

FUNCTION COMPONENTS

Props

React Element

```
const NotificationPanel = ({message}) => (<div>{message}</div>)
```

Function component: Standard JavaScript functions that accept a single prop argument and returns a React element

FUNCTION COMPONENTS

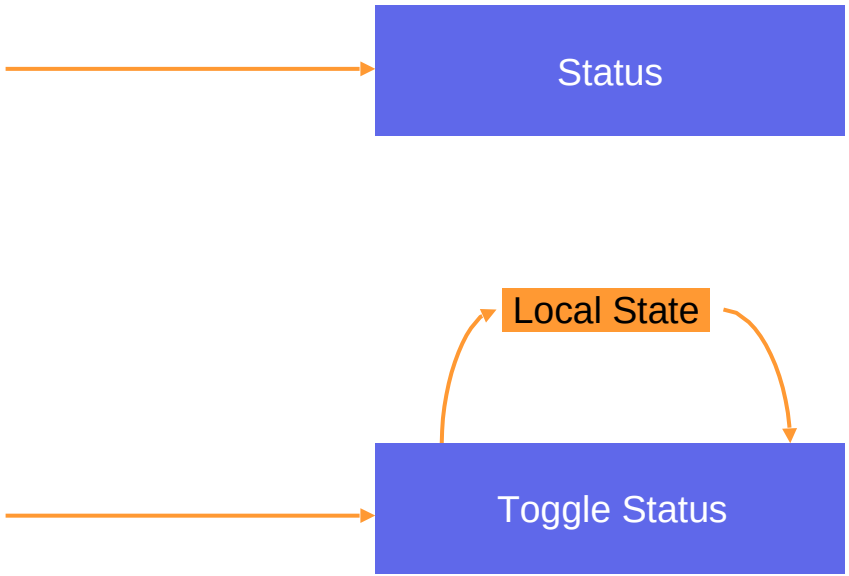
Props

React Element

```
const NotificationPanel = ({message}) => (<div>{message}</div>)
```

State

REASON TO UPGRADE YOUR FUNCTION COMPONENT TO INCORPORATE STATE



Traditionally this would've involved refactoring the function component to a class component so that local state may be used.

Refactoring Code?

Isn’t as easy as it sounds.

HOOKS API

React 16.8 introduced the Hooks API

Allows you to incorporate state in a function component.

No need to refactor Function → Class Components

Hands On

SUMMARY

- Hooks are simple to understand, which is why you might end up writing a lot more function components with hooks than class components in your projects.
- And that is perfectly fine. React does not enforce or express a strong opinion on the type of components that you should build.

OBJECTIVE OF THE COMPONENT

Class Components

- If state & lifecycle methods are required from the get-go

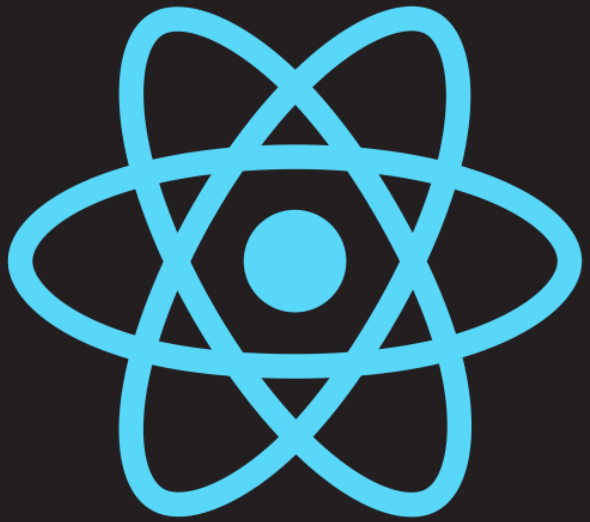
Function Components

- If want to start simple and gradually add state & lifecycle as and when needed
- Hooks API is purely optional and can be plugged in when needed!

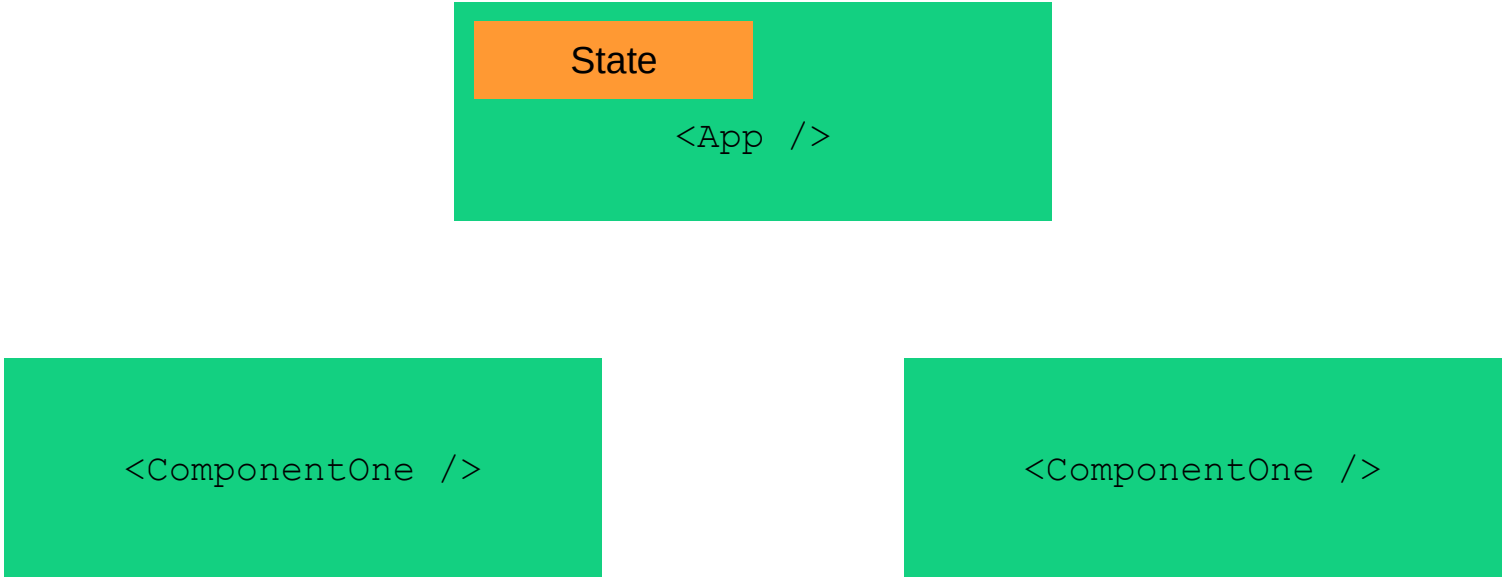
Picking between function and class components depends on the objective of the component.

State & Props

What are Props?

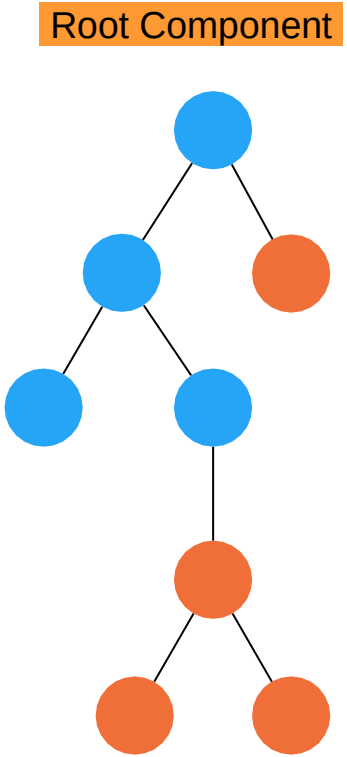


WHAT ARE PROPS?



State is local and private to a component.
How do we share it with other components down the hierarchy?

PROPS



- React apps are made up of several components in a hierarchy
- Props is one of the multiple techniques to share data between the components

PROPS

- Props offer a simple way for parent components to pass data to child components.
- In terms of usage, props can be bound to dynamic data using a pair of curly braces.
- Inside a class component, props can be accessed using this.props.propName.

PROPS IN CLASS COMPONENTS

```
<GetWeather location={locationToSearch} />
```

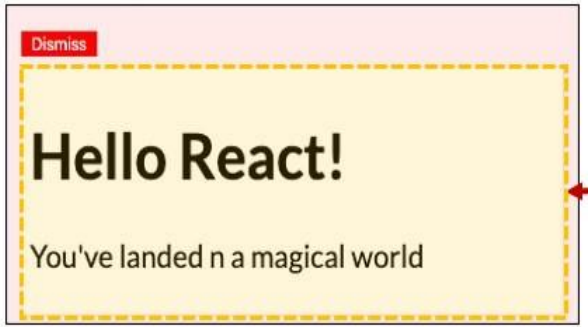
Using the prop

```
class GetWeather extends Component {
  state = {
    temperature: 0,
    conditions: ""
  };
  componentDidMount = () => {
    weatherService(this.props.location)
      .then(({temp, conditions}) =>
        this.setState({temperature: temp, conditions})
      )
      .catch(error => logError(error));
  };
  render() {
    return (
      <>
        <div className="location">{this.props.location}</div>
        <div className="temperature">Temperature: {this.state.temperature}</div>
        <div className="conditions">Conditions: {this.state.conditions}</div>
      </>
    );
  }
}
```

CHILDREN PROP

```
<SuperDialog>
  <div>
    <Hello name={this.state.name} />
    <p>You've landed n a magical world</p>
  </div>
</SuperDialog>;
```

SuperDialog Component



CHILDREN PROP

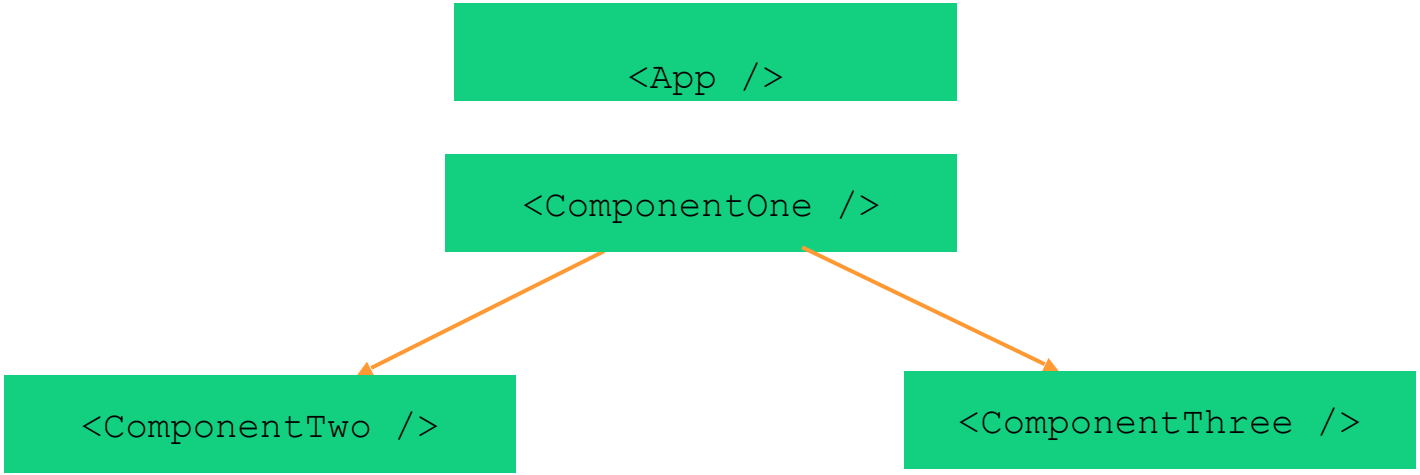
```
const SuperDialog = ({children}) => {
  const [show, setShow] = useState(true);
  return show ? (
    <div className="super-dialog">
      <button onClick={() => setShow(false)}>Dismiss</button>
      <div className="super-content">{children}</div>
    </div>
  ) : null;
};
```

```
<SuperDialog>
  <div>
    <Hello name={this.state.name} />
    <p>You've landed n a magical world</p>
  </div>
</SuperDialog>;
```

PROPS

5 important points to keep in mind with Props

1. TOP-DOWN & ONE-WAY DATA FLOW ARCHITECTURE



- Props enable you to implement a top-down and one-way data flow architecture
- Data generated/created at the parent is passed down to a child component using props
- State should ideally be located at the nearest logical parent component

2. PROPS ARE READ_ONLY

```
class BadComponent extends Component {
  render() {
    this.props.code = 1910;
    return <div>{this.props.code}</div>
  }
}
```

- Props are read-only and must never be mutated
- Components must act like pure functions and never mutate props
- Store prop data in state if you need to modify before consumption

3. PROPS DEFAULT TO TRUE IF NO VALUE IS PASSED

```
const UpperCaseComponent = ({isUpperCase, text}) => {
  return isUpperCase ? <div>{text.toUpperCase()}</div> : <div>{text}</div>
}
```

```
<UpperCaseComponent text={"chocolate cake"} isUpperCase/>
```

This will default to true

4. SETTING DEFAULT VALUES WHEN PROPS ARE NOT SET

```
const UpperCaseComponent = ({isUpperCase, text}) => {
  return isUpperCase ? <div>{text.toUpperCase()}</div> : <div>{text}</div>
}
```

```
UpperCaseComponent.defaultProps = {
  text: "Hello there!"
}
```

Sets the default value of the *text* prop as “Hello there!”

5. STATIC VS DYNAMIC DATA IN PROPS

```
<EmployeeCard data="{name:'Logan Roy', employeeCode: 1, title: 'Chairman'}" />
```

This will be treated as a static String

```
<EmployeeCard data={{name:'Logan Roy', employeeCode: 1, title: 'Chairman'}} />
```

This will be treated as a dynamic object

How to send data back up to a parent component?

SENDING DATA BACK UP TO THE PARENT

By implementing a function in a prop

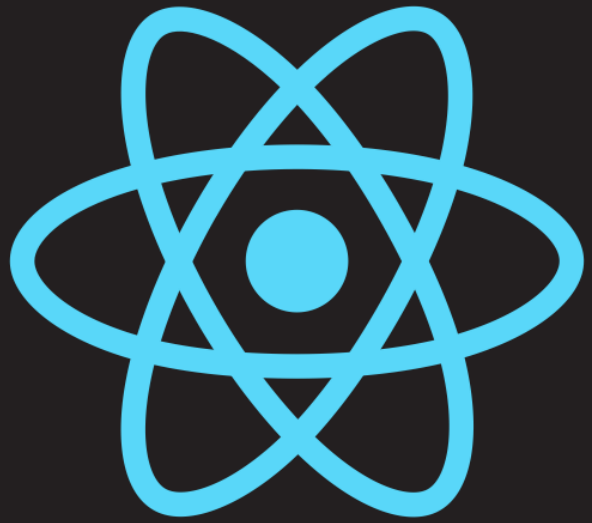
SUMMARY

```
<EmployeeCard data={{name:'Logan Roy', employeeCode: 1, title:
'Chairman'}} />
```

- Props let you implement one-way data flow conduits
- They let parent components pass data to children
- Helps you create reusable components

State & Props

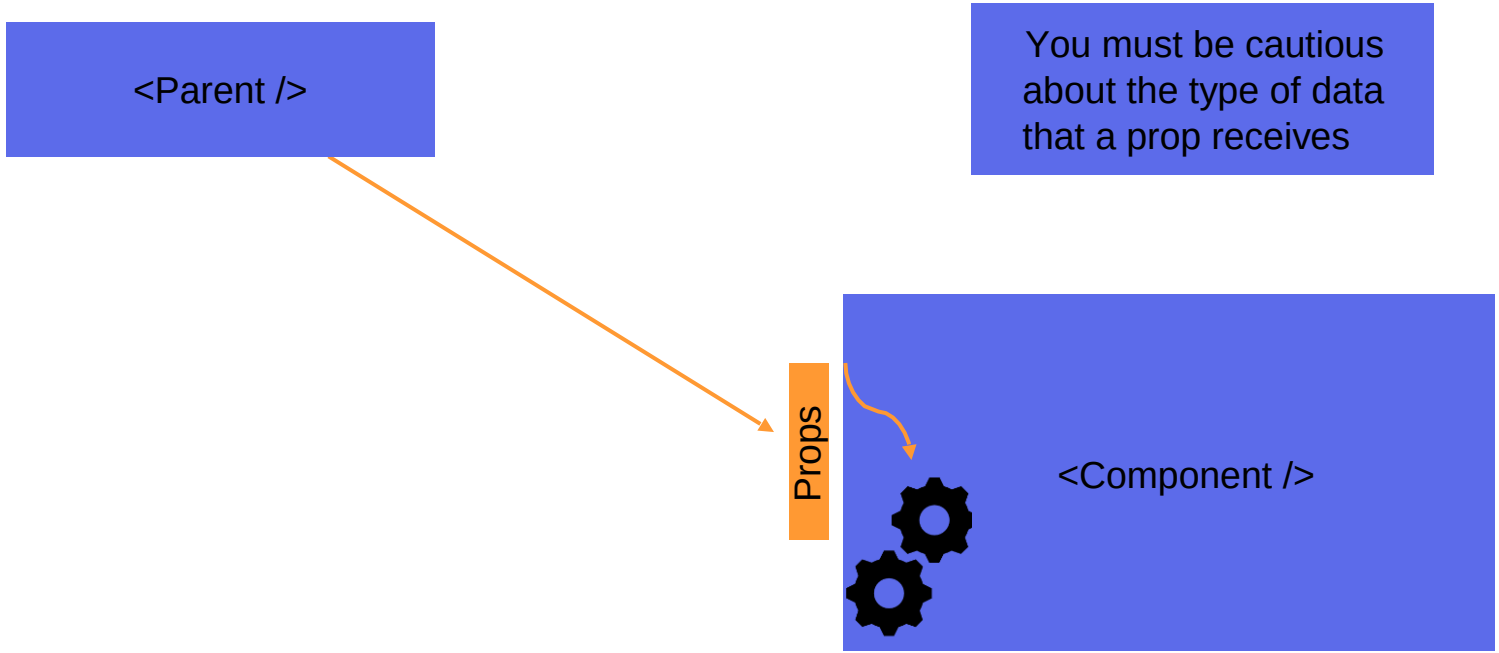
Type Checking with Prop Types



PROPS

- Props let you access data from parent components
- You can pass Strings, Numbers, Objects, Arrays, Functions and more

PROPS ULTIMATELY FEED TO A COMPONENT'S INTERNAL LOGIC



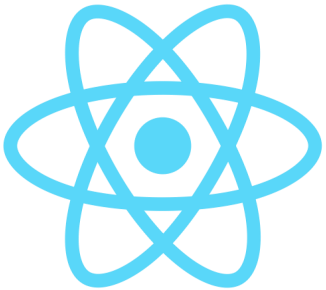
THE COLLECTION PROP

```
<Music COLLECTION={[ARRAY]} />
```

The collection prop must be an Array. If you pass a String, the Music component might throw an error!

It becomes important to type-check data that flows in through props.

A PACKAGE OF VALIDATORS FOR PROPS



`prop-types`

These validators throw appropriate warnings in the console when incompatible data comes in through a prop

Prop validation only works in development because validators are removed when a production build is created!

These validators will issue warnings only in development.

Hands On

PROP-TYPES TO THE RESCUE

- Meaningful warnings help you debug code and fix problems faster.
- WIth validators from the prop-type package, reusable components can throw warning and guide the developer towards using the component correctly.
- Hence, it becomes important to validate props in production.

CUSTOM VALIDATOR – E-MAIL ADDRESSES

```
StudentCard.propTypes = {
  student: PropTypes.shape({
    name: PropTypes.string.isRequired,
    email: function(props, propName, componentName) {
      if (!/^\w+@[a-zA-Z_]+?\.[a-zA-Z]{2,3}$/.test(props[propName])) {
        return new Error(
          `Invalid Prop - ${propName} supplied to ${componentName}. Expected E-                    Mail IDs`
        );
      }
    },
    id: PropTypes.number.isRequired,
    active: PropTypes.bool.isRequired,
    courses: PropTypes.array.isRequired
  })
};
```

TYPECHECKING DATA

Typechecking data that comes through props can help you avoid headaches!

thank you!