Chapter 3: Props

Props, or properties, are data that is passed to child components in a React application. React components render UI elements based on their props and their internal state. The props that a component takes (and uses) defines how it can be controlled from the outside.

Section 3.1: PropTypes

The `prop-types` package allows you to add runtime type checking to your component that ensures the types of the props passed to the component are correct. For instance, if you don't pass a `name` or `isYummy` prop to the component below it will throw an error in development mode. In production mode the prop type checks are not done. Defining `propTypes` can make your component more readable and maintainable.

```
import React, { Component } from 'react';
import PropTypes from 'prop-types';
import { AppRegistry, Text, View } from 'react-native';

import styles from './styles.js';

class Recipe extends Component {
  static propTypes = {
    name: PropTypes.string.isRequired,
    isYummy: PropTypes.bool.isRequired
  }
  render() {
    return (
      <View style={styles.container}>
        <Text>{this.props.name}</Text>
        {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
      </View>
    )
  }
}

AppRegistry.registerComponent('Recipe', () => Recipe);


// Using the component
<Recipe name="Pancakes" isYummy={true} />
```

Multiple PropTypes

You can also have multiple `propTypes` for one props. For example, the name props I'm taking can also be an object, I can write it as.

```
static propTypes = {
  name: PropTypes.oneOfType([
      PropTypes.string,
      PropTypes.object
  ])
}
```

Children Props

There is also a special props called `children`, which is **not** passed in like

```
<Recipe children={something}/>
```

Instead, you should do this

```
<Recipe>
  <Text>Hello React Native</Text>
</Recipe>
```

then you can do this in Recipe's render:

```
return (
  <View style={styles.container}>
    {this.props.children}
    {this.props.isYummy ? <Text>THIS RECIPE IS YUMMY</Text> : null}
  </View>
)
```

You will have a **<Text>** component in your `Recipe` saying `Hello React Native`, pretty cool hum?

And the propType of children is

```
children: PropTypes.node
```

Section 3.2: What are props?

Props are used to transfer data from parent to child component. Props are read only. Child component can only get the props passed from parent using **this.props.keyName**. Using props one can make his component reusable.

Section 3.3: Use of props

Once setup is completed. Copy the code below to `index.android.js` or to `index.ios.js` file to use the props.

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Greeting extends Component {
  render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}

class LotsOfGreetings extends Component {
  render() {
    return (
      <View style={{alignItems: 'center'}}>
        <Greeting name='Rexxar' />
        <Greeting name='Jaina' />
        <Greeting name='Valeera' />
      </View>
    );
  }
}

AppRegistry.registerComponent('LotsOfGreetings', () => LotsOfGreetings);
```

Using props one can make his component generic. For example, you have a Button component. You can pass different props to that component, so that one can place that button anywhere in his view.

source: Props-React Native

Section 3.4: Default Props

defaultProps allows you to set default prop values for your component. In the below example if you do not pass the name props, it will display John otherwise it will display the passed value

```
class Example extends Component {
  render() {
    return (
      <View>
        <Text>{this.props.name}</Text>
      </View>
    )
  }
}


Example.defaultProps = {
  name: 'John'
}
```