React Native - Basics - Props

- Dr Nick Hayward

A brief intro to the basics of *Props* in React Native app development.

Contents

Intro

`props` in React, and React Native in this instance, are parameters we may pass as a component is created.

Such props enable most components to be customised as they're created.

`props` usage

We can use `props` to pass variables, for example, within a component. More often, we will use `props` to pass values and variables between components.

In custom components, such usage of `props` helps abstract component structure to help reuse within an app.

e.g.

```
// import React, Component module as Component from base React
import React, { Component } from 'react';
// import Text as Text &c. from React Native
import { AppRegistry, Text, View } from 'react-native';

// custom abstracted component - expects props for text `output`
class OutputText extends Component {
  render() {
    return (
      // render passed props `output` value
      <Text>{this.props.output}</Text>
    );
  }
}

// default component - use View container render OutputText message with passed
props...
export default class WelcomeMessage extends Component {
  render() {
    return (
      // View container - render Text output from OutputText component
      <View style={{alignItems: 'center'}}>
        // JSX embed OutputText component - pass value for props `output`
        <OutputText output='welcome to the basic tester...' />
      </View>
    );
  }
}
```

In this example, we define the required imports for React and React Native, including existing components we need for this basic app.

- `AppRegistry` - entry point for JavaScript to enable a React Native app to run...
  - added as part of `init` command for React Native apps

- `Text` - used to display text within an app
- `View` - a UI container for displaying content (basic requirement for UI development with React Native)
    - supports layout structures with flexbox, style, touch, accessibility...

Then, we define our required custom components. One abstracted for broader re-use, the other for use in the current specific app.

`OutputText` is the abstracted component, which accepts `props` as part of the output for a standard `Text` component. As the `render()` function is called for this component, it returns text output with the value of the passed props.

`WelcomeMessage` is a custom component, which is also set as the default export for the module. If the export is not explicitly set, this component (class in JS) will be called at execution.

As this component is executed, it returns a standard `View` container with its own defined `style` props. The custom JSX uses the abstracted `OutputText` component to define the UI rendering. In this example, we set the expected `props` of `output` on the `OutputText` JSX.

In effect, as the JSX is executed, it calls the custom component `OutputText`, which renders a standard `Text` component with the value from the passed `props`.

So, we end up with a `View` container, which includes rendered text for `welcome to the basic tester...`.

The benefit is that we can now re-use the `OutputText` component whenever we need to output text using a passed `props` value.