**STATE**

There are two types of data that control a component: props and state. Props are set by the parent and they are fixed throughout the lifetime of a component. For data that is going to change, we have se state.

In general, we should initialize state in the constructor, and then call setState when you want to change it.

For Example, let’s say we want to make text that blinks all the time. The text itself gets set once when the blinking component gets created, so the text itself is a prop. The “whether the text is currently on or off” changes over time, so that should be kept in state.

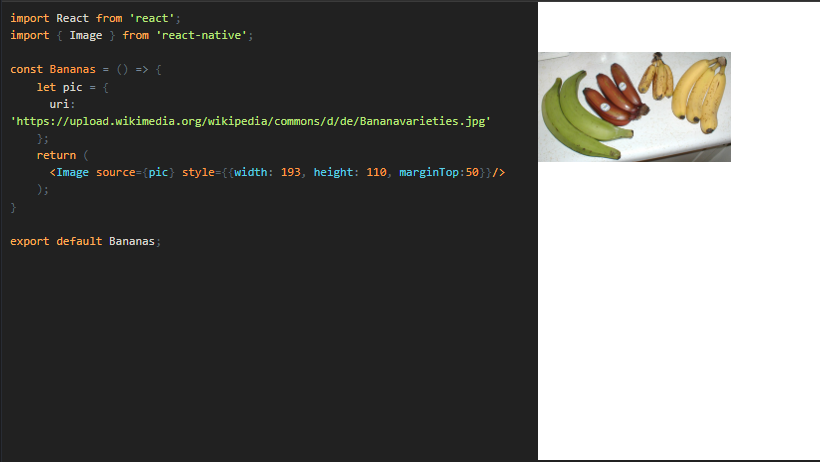


In a real application, you probably won't be setting state with a timer. You might set state when you have new data from the server, or from user input. You can also use a state container like [Redux](https://redux.js.org/) or [MobX](https://mobx.js.org/) to control your data flow. In that case you would use Redux or MobX to modify your state rather than calling setState directly.

**PROPS**

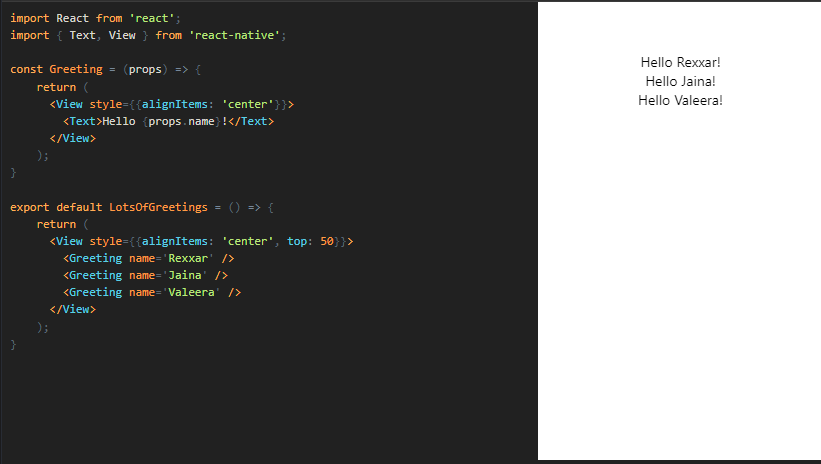
Most components can be customized when they are created, with different parameters. These created parameters are called props, short for properties.

For example, one basic React Native component is the Image. When you create an image, you can use a prop named source to control what image it shows.



Notice the braces surrounding {pic} - these embed the variable pic into JSX. You can put any JavaScript expression inside braces in JSX.

Your own components can also use props. This lets you make a single component that is used in many different places in your app, with slightly different properties in each place by referring to props in your render function. Here's an example:

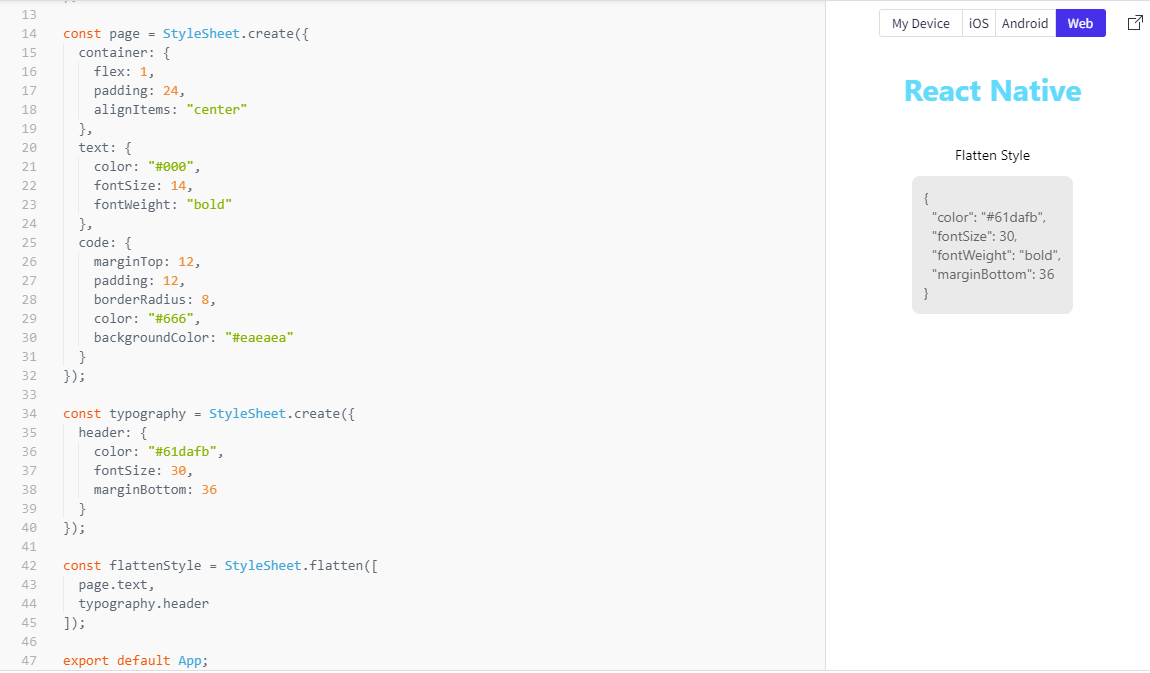


Using name as a prop lets us customize the Greeting component, so we can reuse that component for each of our greetings. This example also uses the Greeting component in JSX, similar to the [Core Components](https://reactnative.dev/docs/intro-react-native-components). The power to do this is what makes React so cool - if you find yourself wishing that you had a different set of UI primitives to work with, you can invent new ones.

**STYLESHEET**

A StyleSheet is an abstraction similar to CSS StyleSheets.

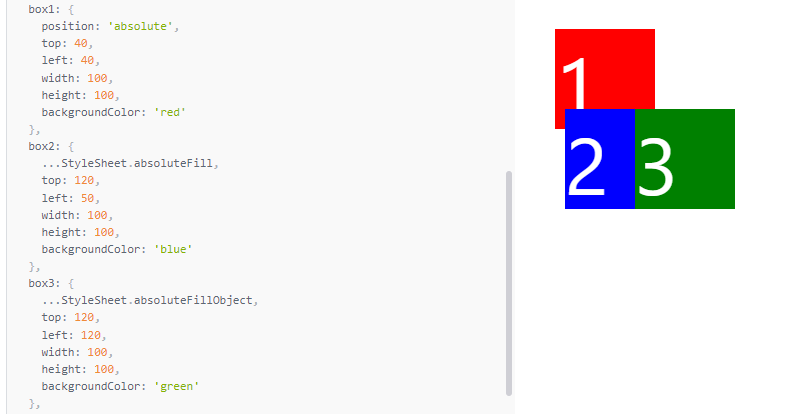
flatten(): Flattens an array of style objects, into one aggregated style object. Alternatively, this method can be used to lookup IDs, returned by StyleSheet.register.



absoluteFill(): A very common pattern is to create overlays with position absolute and zero positioning (position: 'absolute', left: 0, right: 0, top: 0, bottom: 0), so absoluteFill can be used for convenience and to reduce duplication of these repeated styles. If you want, absoluteFill can be used to create a customized entry in a StyleSheet.



absoluteFillObject(): Sometimes you may want absoluteFill but with a couple tweaks - absoluteFillObject can be used to create a customized entry in a StyleSheet.



hairLineWidth():This is defined as the width of a thin line on the platform. It can be used as the thickness of a border or division between two elements. Example:

