**[This](https://classroom.udacity.com/nanodegrees/nd019/parts/1187639a-82e2-4dd8-9d7b-477af14e5dd7/modules/cc51381c-49d3-43b4-86c3-dd55f9b8a3f6/lessons/c6957bbe-51e4-4d4e-b634-0aee13b4d11b/concepts/47161ebe-3079-48fc-9a80-b595999820d1" \t "_blank)** is where we left off in the last lesson.

**StyleSheet**

Benefits of StyleSheet, as listed in the [**React Native docs**](https://facebook.github.io/react-native/docs/stylesheet):

Code quality

* By moving styles away from the render function, you're making the code easier to understand.
* Naming the styles is a good way to add meaning to the low-level components in the render function.

Performance

* Making a stylesheet from a style object makes it possible to refer to it by ID instead of creating a new style object every time.
* It also allows to send the style only once through the bridge. All subsequent uses are going to refer to an id (not implemented yet).

Another benefit is that StyleSheet validates the content within the style object as well. This means that should there be any errors in any properties or values in your style objects, the console will throw an error during compilation instead of at runtime.

If you wanted to implement more than one style to a component, the style prop can accept styles as an array:

`js

This will be red, then greenLarge `

The above <Text> component will render large green text, as the last style in the array will take precedence. This is a great way to inherit styles!

Properties are written in camelCase (i.e., borderRadius in CSS in JS, but border-radius on the web).

**Flexbox**

React Native leverages a version of **flexbox** to build component layout. This is primarily due to flexbox's ability to provide consistent layouts across different screen sizes.

Flexbox containers comprise of two axes: a **main axis**, as well as a **cross axis**. Some of the more critical properties to consider when building layouts with flexbox include flex-direction, justify-content, and align-items.

By default, every element in React Native has the flexDirection: column declaration. When an element has a flex-direction of column, its *Main* Axis is vertical and its *Cross* Axis is horizontal.

If you give an element a flexDirection: row declaration, the axes switch. The *Main* axis becomes horizontal, while the *Cross* axis becomes vertical. Again, this is crucial because your entire layout is dependent on these two axes.

In order to specify how children align themselves along the *Main* Axis, you'll use the justifyContentproperty.

In order to specify how children align themselves along the *Cross* Axis, you'd use the align-itemsproperty.