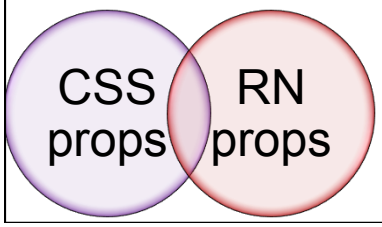
Styling React Native

tl;dr

- Native components receive a prop called style which should resolve to a plain JavaScript object which contains keys that look like CSS but are definitely not CSS.
- We can create that style object any way you want, raw, inline, imported, but preferably with StyleSheet.create()
- These can even be applied dynamically using JavaScript logic

React Native styles are **not** CSS

- They are simple JavaScript objects
- camel-cased, and not kebab-cased



The keys/props may look like CSS, but ...

- Not all CSS properties are supported
- There are some properties that CSS doesn't support

```
<Anything style={someStyleHere} />
```

You don't use a special language for defining styles. You just style your application using JavaScript. All the core components accept a prop named *style*.

Styles are only applied on the component itself

There are no classes, no ids, no CSS selectors

```
let big={ fontSize: 80 };

<Text style={big}>Lorem</Text>
```

```
let big={ fontSize: 80 };
let Text={fontFamily:'cochin'};
let #prodLabel={color:'red'};

<Slider class="big" />
<Text class="big" />
<Text class="smaller" />
<Text id="prodLabel" />
```

You can apply arrays of styles

```
export const MyComponent () => (
 <Text style={[bang, big, shout]}>
  I don't understand the question,
  and I won't respond to it.
 </Text>
);
```

These would "upsert" (Values in the later ones clobber earlier ones if they already exist)

And combine inline with external

```
export const MyComponent () => (
 <Text style={[big,{color:'red'}]}>
  I don't understand the question,
  and I won't respond to it.
 </Text>
);
```

These would "upsert" (Values in the later ones clobber earlier ones if they already exist)

How to define styles

You define styles in 4 ways

1. Inline
2. Inside the component - raw
3. Inside the component - with StyleSheet.create()
4. Separate stylesheet

1. Inline

```
<Text style={{ color:'red' }}>
I am going to oblige and answer the nice officer's
questions because I am an honest man with no
secrets to hide.
</Text>
<View style={{ flex: 1, textAlign:'left' }}>
```

Note the double-curly braces

2. Raw styles

These are easier to write but slower at runtime

```
const big = {
 fontSize: 50,
 fontWeight: 'bold'
};

<Text style={big}>
There are so many poorly chosen words in
that sentence.
</Text>
```

MyComponent.js

```
const styles = StyleSheet.create({
 container: {
  flex: 1,
  backgroundColor: '#ffa1d6',
  alignItems: 'center',
  justifyContent: 'center',
 },
 hdr: {
  fontSize: 25,
  color: '#000000'
  textAlign: 'right',
 },
});
// Your component might go here
```

4. StyleSheet.create()

And here's how to apply them

MyComponent.js

```
// Your styles object might be created here.
function MyComponent() {
 return <View style={styles.container}>
  <Text style={styles.hdr}>Cool App</Text>
  <Text>
   There are so many poorly chosen words
   in that sentence.
  </Text>
 </View>
}
```

4. Separate stylesheet

myStyles.js

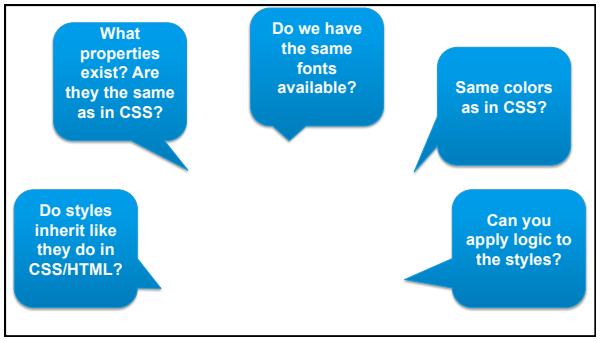
```
export const styles =
StyleSheet.create({
 container: {
  flex: 1,
  alignItems: 'left',
 },
 headerText: {
  fontSize: 25,
 },
};
```

... sort of. It's just an object. So export from one file and import it into your component.

MyComponent.js

```
import {styles} from './myStyles';
function MyComponent() {
 return (
  <View style={styles.container}>
  ...
  </View>
}
```

How do these styles work?



Styles are seldom inherited!

- (Developer) React components are designed with strong isolation in mind: You should be able to drop a component anywhere in your application, trusting that as long as the props are the same, it will look and behave the same way. Text properties that could inherit from outside of the props would break this isolation.
- It's easier to list where they *are* and assume that everywhere else is not.
- <Text> embedded inside a <Text> is.

- We believe that this more constrained way to style text will yield better apps:
- (Implementor) The implementation of React Native is also simplified. We do not need to have a fontFamily field on every single element, and we do not need to potentially traverse the tree up to the root every time we display a text node. The style inheritance is only encoded inside of the native Text component and doesn't leak to other components or the system itself.

So how do you inherit?

Pass styles down as props

- or -

Create a styled component and use it at different levels

```
const MyText =
 ({contents}) => (
  <Text style={{fontSize:20}}>{contents}</Text>)
```

Properties

backgroundColor
color
fontSize
fontWeight
pad
alignContent,
alignItems,
alignSelf,
aspectRatio,
backfaceVisibility,
backgroundColor,
borderBottomColor,
borderBottomLeftRadius,
borderBottomRightRadius,
borderBottomWidth,
borderColor,
borderLeftColor,
borderLeftWidth,
borderRadius,
borderRightColor,
borderRightWidth,
borderStyle,
borderTopColor,
borderTopLeftRadius,
borderTopRightRadius,
borderTopWidth,
borderWidth,
bottom,
color,
decomposedMatrix,
direction,
display,
elevation,
flex,
flexBasis,
flexDirection,
flexGrow,
flexShrink,
flexWrap,
fontFamily,
fontSize,
fontStyle,
fontVariant,
fontWeight,
height,
includeFontPadding,
justifyContent,
left,
letterSpacing,
lineHeight,
margin,
marginBottom,
marginHorizontal,
marginLeft,
marginRight,
marginTop,
marginVertical,
maxHeight,
maxWidth,
minHeight,
minWidth,
opacity,
overflow,
overlayColor,
padding,
paddingBottom,
paddingHorizontal,
paddingLeft,
paddingRight,
paddingTop,
paddingVertical,
position,
resizeMode,
right,
rotation,
scaleX,
scaleY,
shadowColor,
shadowOffset,
shadowOpacity,
shadowRadius,
textAlign,
textAlignVertical,
textDecorationColor,
textDecorationLine,
textDecorationStyle,
textShadowColor,
textShadowOffset,
textShadowRadius,
tintColor,
top,
transform,
transformMatrix,
translateX,
translateY,
width,
writingDirection,
zIndex

There's a great cheatsheet of React Native style properties here: http://bit.ly/RNStyles

Fonts on Android

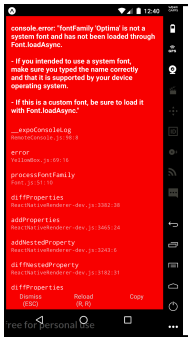
- normal
- notoserif
- sans-serif
- sans-serif-light
- sans-serif-thin
- sans-serif-condensed
- sans-serif-medium
- serif
- Roboto
- monospace

Fonts on iOS

- Academy Engraved
- Al Nile
- American Typewriter
- Apple Color Emoji
- AppleSDGothicNeo
- Arial
- Arial Hebrew
- Avenir
- Avenir-Roman
- Bangla Sangam MN
- Baskerville
- Bodoni 72
- Bodoni Ornaments
- Bradley Hand
- Chalkboard SE
- Chalkduster
- Cochin
- Copperplate
- Courier
- Damascus
- Devanagari Sangam
- Didot
- DiwanMishafi
- Euphemia UCAS
- Farah
- Futura
- Geeza Pro
- Georgia
- Gill Sans
- Gujarati Sangam MN
- Gurmukhi MN
- Heiti SC
- Helvetica
- Hiragino Mincho ProN
- Hiragino Sans
- Hoefler Text
- Iowan Old Style
- Kailasa
- Kannada Sangam MN
- Khmer Sangam MN
- Kohinoor Bangla
- Lao Sangam MN
- Malayalam Sangam
- Marker Felt
- Menlo
- Mishafi
- Noteworthy
- Optima
- Oriya Sangam MN
- Palatino
- Papyrus
- Party LET
- PingFang HK
- Savoye LET
- Sinhala Sangam MN
- Snell Roundhand
- Symbol
- Tamil Sangam MN
- Telugu Sangam MN
- Thonburi
- Times New Roman
- Trebuchet MS
- Verdana
- Zapf Dingbats
- Zapfino.

Fonts on both Android and iOS

- …



console.error: "fontFamily 'Optima' is not a system font and has not been loaded through Font.loadAsync.

- If you intended to use a system font, make sure you typed the name correctly

To use an unsupported font is an error.

So how do you use fonts?

```
label: {
 color: 'blue',
 fontFamily: Platform.OS === 'ios' ? 'Optima' : 'serif',
 fontSize: 20,
 fontWeight: 'bold',
},
alert: {
 color: 'red',
 fontFamily: Platform.select({
  ios:'San Francisco',
  android:'sans-serif'}),
 fontSize: 15,
},
```

You use Platform to pick Platform-specific fonts

What if I want to have the same font on both platforms?

- You can't. Well ... not natively.

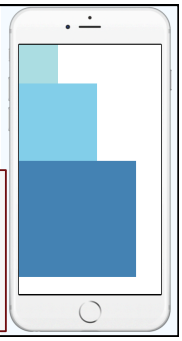
1. Obtain a custom font
2. Install the same font on both platforms using font.loadAsync
3. Use that font

How to use cross-platform custom fonts: http://bit.ly/RNCustomFonts

Sizes are unitless and independent of pixel density

They represent pixels, just not *actual* pixels.

```
<View style={{width: 50, height: 50,
  backgroundColor: 'powderblue'}} />
 <View style={{width:100, height:100,
  backgroundColor: 'skyblue'}} />
 <View style={{width:150, height:150,
  backgroundColor: 'steelblue'}} />
</View>
```



What values can color take on?

'red'	All the named colors from the W3C spec - about 100 of them
'#fff'	#rgb
'#fa75ed'	#rrggbb
'rgb(255, 100, 0)'	Red/Green/Blue
'rgba(255, 100, 0)'	RGB with alpha channel
'hsl(360, 100%, 50%)'	Hue, saturation, lightness
'hsla(360, 100%, 50%, 0.75)'	HSL with alpha channel

You can conditionally style

```
state = {
 danger: true,
}
render() {
 const color = this.state.danger ? 'red' : 'black';
 return (
  <View>
   <Text style={{ color: color }}>
    I like being with you. Really? Did nothing cancel?
   </Text>
  </View>
 );
}
```

It's just JavaScript, so apply logic.

You can conditionally style

```
const Product = (product) => {
 const style = [
  product.defaultStyle,
  product.isInStock && {backgroundColor:'green'},
  product.isOnSale && {borderColor:'red', borderWidth:2}
 ];
 return (
  <View style={style}>
    <Text>{product.description}</Text>
  <View>
 )
}
```

It's just JavaScript, so apply logic.

tl;dr

- Native components receive a prop called style which should resolve to a plain JavaScript object which contains keys that look like CSS but are definitely not CSS.
- We can create that style object any way you want, raw, inline, imported, but preferably with StyleSheet.create()
- These can even be applied dynamically using JavaScript logic