# WELCOME TO scope

Welcome to our second React Native meeting! We'll be going a little more in depth about how to make interactive applications.

QR Code for Attendance

</>

Please fill out this attendance form! At the end of the semester, prizes for participation may or may not be awarded based upon attendance and other secret to-be-determined factors :^)

# Announcements

*Dues:*

- Venmo $30 to @scopeusc

*This week:*

- Thursday 7pm PT: trivia night and opening ceremony of Scope cup
- Join the discord before the meeting!
- Office hours beforehand at 6:30pm to go over today's lesson

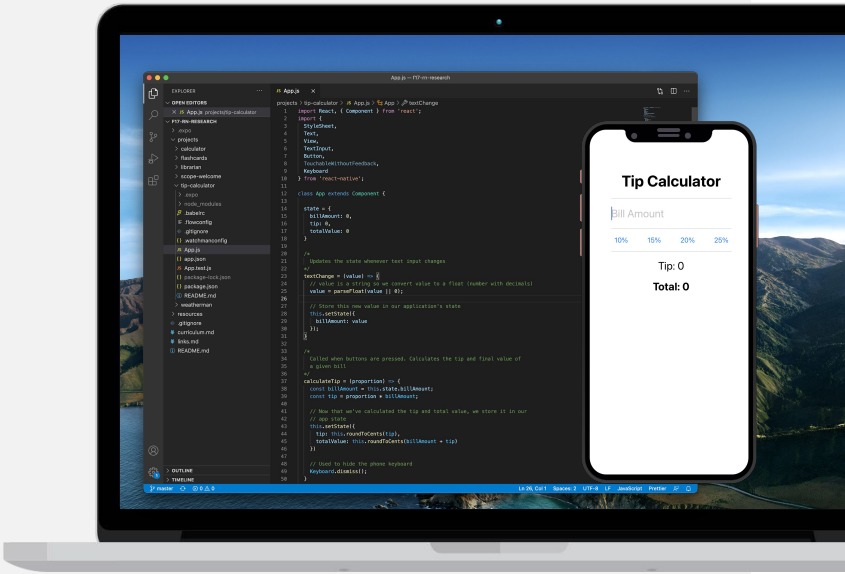
# TODAY'S LESSON :^)

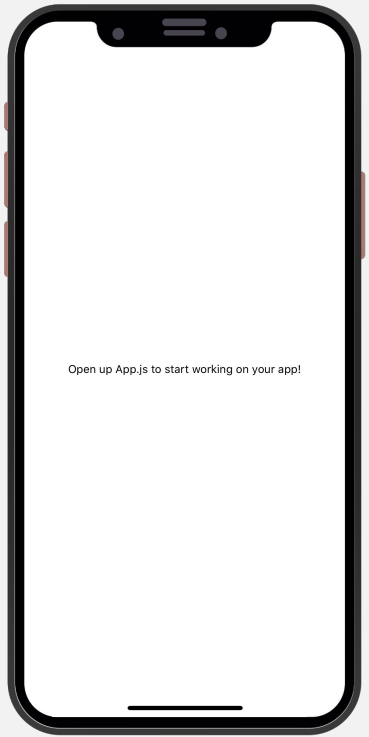
*What you'll build:*

A tip calculator application with a text field, a little bit more interactive and customizable!

*What you'll learn:*

How to implement text fields in React Native, and creating functional buttons.

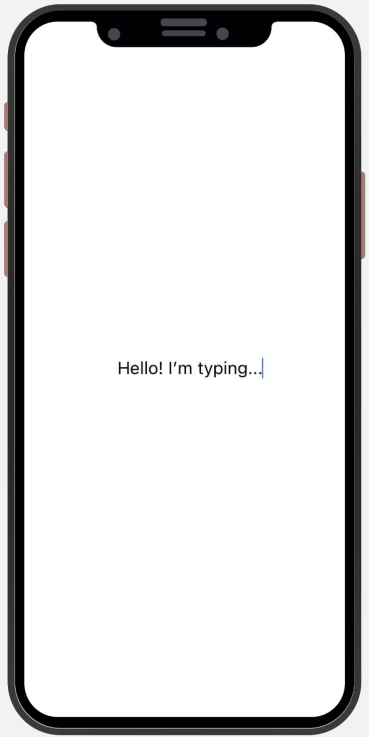





# CREATE NEW PROJECT

*Open a terminal, navigate to your Scope folder and run the following commands:*

```
1.  expo init TipCalculator
2.  cd TipCalculator
3.  npm start
```




# TEXT FIELD

*A new UI element for today! You can enter text in here.*

```
<TextInput placeholder="No text yet!"/>
```

# SETTING UP

First, we need to take several steps to set up our App.js before we start adding UI to the view.

1

Export a class, not a function

Replace the line

```
export default function App() {
```

With:

```
export default class App extends Component {
```

This allows us to place functions within the App class, rather than App being a singular function.

2

Import Component

In the

```
import React from 'react';
```

line, add `React, { Component } from 'react';` so that we are importing Component from React. A component is a type of class that provides additional features.

3

Create the render() method

Surround the `return ( );` method with the render method. Add `render(){` on line 6 before the 'return', and add a closing curly brace after return's closing parenthesis on line 11 (before the semicolon). The return method is required to implement the Component.

# ADD ELEMENTS

Now we can add the TextInput field, and start to add the logic for calculation of tip.

1

Import the UI components

Add

```
TextInput, Button, TouchableWithoutFeedback, Keyboard
```

To the list of React element imports at the top of App.js, so the line reads::

```
import { StyleSheet, Text, View, TextInput, Button,
TouchableWithoutFeedback, Keyboard} from
'react-native';
```

2

Add a TextInput

After changing the `</Text>` element to read “Tip Calculator”, add a TextInput element

```
<TextInput autoFocus={true} keyboardType='numeric'
placeholder="Bill Amount" onChangeText={this.textChange}
/>
```

3

Add a State for tracking the bill and tip amounts

We need a state for the App class so we can track the bill and tip amounts, since it’s a variable that will update the view. Add the following to the App class:

```
constructor(){
   super()
   this.state = {
     billAmount: 0,
   tip: 0,
   totalValue: 0
   }
 };
```

# FUNCTIONS!

Let's add some functions to calculate the tip. But I believe in you guys! I've given you the headers, but take some time to try and figure it out on your own. Talk with your groups.

```
/*
  Updates the state whenever text input changes
  value is a string so we convert value to a float (number with
  decimals), and store it to the state
 */
textChange = (value) => {
  value = parseFloat(value || 0);
  // what do we need to be doing here?
 }

 /*
  Called when buttons are pressed. Calculates the tip and final value of
  a given bill and stores it to state
 */
 calculateTip = (proportion) => {
  // we are given a percentage to tip, I want you to calculate it and
update appropriately
  // Used to hide the phone keyboard
  Keyboard.dismiss();
 }

 /*
  Helper function that rounds numbers to two decimals (like cent values)
  A nice mathematical way to round a number to 2 digits
 */
 roundToCents = (amount) => {
  // there's an easy mathematical way to do this, or nice function you
can find :') try the math one first!
  return result;
 }
```

# FUNCTIONS!

Here's how we did it. Questions?

1

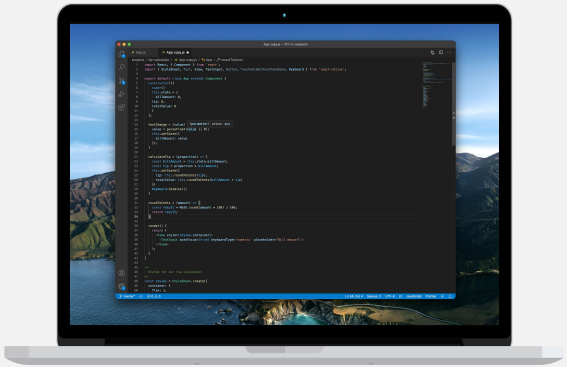
```
/*
  Updates the state whenever text input changes
  value is a string so we convert value to a float (number with
  decimals), and store it to the state
 */
textChange = (value) => {
  value = parseFloat(value || 0);
  this.setState({
    billAmount: value
  });
 }

 /*
  Called when buttons are pressed. Calculates the tip and final value of
  a given bill and stores it to state
 */
 calculateTip = (proportion) => {
  const billAmount = this.state.billAmount;
  const tip = proportion * billAmount;
  this.setState({
    tip: this.roundToCents(tip),
    totalValue: this.roundToCents(billAmount + tip)
  })
  // Used to hide the phone keyboard
  Keyboard.dismiss();
 }

 /*
  Helper function that rounds numbers to two decimals (like cent values)
  A nice mathematical way to round a number to 2 digits
 */
 roundToCents = (amount) => {
     const result = Math.round(amount * 100) / 100;
  return result;
 }
```

# CODE

*Your current code should read as shown:*



```
import React, { Component } from 'react';
import { StyleSheet, Text, View, TextInput, Button,
TouchableWithoutFeedback, Keyboard } from 'react-native';

export default class App extends Component {

 constructor(){
   super()
   this.state = {
     billAmount: 0,
   tip: 0,
   totalValue: 0
   }
 };
 textChange = (value) => {
   value = parseFloat(value || 0);
   this.setState({
     billAmount: value
   });
 }

 calculateTip = (proportion) => {
   const billAmount = this.state.billAmount;
   const tip = proportion * billAmount;
   this.setState({
     tip: this.roundToCents(tip),
     totalValue: this.roundToCents(billAmount + tip)
   })
   Keyboard.dismiss();
 }

 roundToCents = (amount) => {
   const result = Math.round(amount * 100) / 100;
   return result;
 }

 render() {
   return (
     <View style={styles.container}>
       <TextInput autoFocus={true} keyboardType='numeric'
         placeholder="Bill Amount" onChangeText={this.textChange}
         />
     </View>
   );
 }
}
```

# BUTTONS!

So now we need some buttons to calculate the tip. Below the TextInput element, add another encapsulating view with three buttons. The onPress prop is telling each button what function to call each time the button is pressed. Try and add it in for yourself, paying close attention to syntax. Hint: you'll need an arrow function.

```
<View>
  <Button
    title="10%"
    onPress={}
  />

  <Button
    title="15%"
    onPress={}
  />

  <Button
    title="20%"
    onPress={}
  />

  <Button
    title="25%"
    onPress={}
  />
</View>
```

1

# BUTTONS!

So now we need some buttons to calculate the tip. Below the TextInput element, add another encapsulating view with three buttons. The onPress prop is telling each button what function to call each time the button is pressed. We are using the arrow function again so we can call our function using JavaScript.

```
<View>
  <Button
    title="10%"
    onPress={() => {
      this.calculateTip(0.1);
    }}
  />

  <Button
    title="15%"
    onPress={() => {
      this.calculateTip(0.15);
    }}
  />

  <Button
    title="20%"
    onPress={() => {
      this.calculateTip(0.2);
    }}
  />

  <Button
    title="25%"
    onPress={() => {
      this.calculateTip(0.25);
    }}
  />
</View>
```

# RESPONSE

Finally, we need some text to display the output of our application. We can add this right in below the view for the buttons, but inside the root view that the render method is returning. Try and figure out how to access the correct variables.

1

```
<Text>Tip: {}</Text>
<Text>Total: {}</Text>
```

# RESPONSE

Finally, we need some text to display the output of our application. We can add this right in below the view for the buttons, but inside the root view that the render method is returning.

1

```
<Text>Tip: {this.state.tip}</Text>
<Text>Total: {this.state.totalValue}</Text>
```

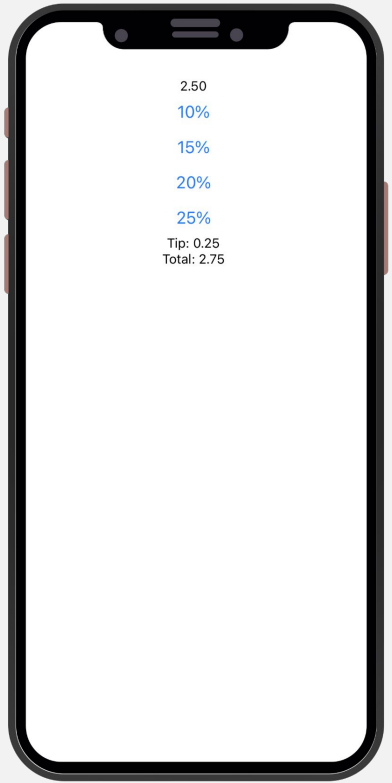
# RUN THE APP!

*In your terminal/command prompt:*

```
npm start
```

*Once Expo DevTools opens in the browser, either:*

1. Scan the QR code with your phone to open Expo Go
2. Click “Run on iOS Simulator” or “Run on Android device/emulator”
3. App is now fully functional, but kinda ugly



# CODE

*Your current code*

*should read as shown:*

```
import React, { Component } from 'react';
import { StyleSheet, Text, View, TextInput, Button,
TouchableWithoutFeedback, Keyboard } from 'react-native';

export default class App extends Component {
 constructor(){
   super()
   this.state = {
     billAmount: 0,
   tip: 0,
   totalValue: 0
   }
 };
 textChange = (value) => {
   value = parseFloat(value || 0);
   this.setState({
     billAmount: value
   });
 }

 calculateTip = (proportion) => {
   const billAmount = this.state.billAmount;
   const tip = proportion * billAmount;
   this.setState({
     tip: this.roundToCents(tip),
     totalValue: this.roundToCents(billAmount + tip)
   })
   Keyboard.dismiss();
 }

 roundToCents = (amount) => {
   const result = Math.round(amount * 100) / 100;
   return result;
 }
```

```
 render() {
   return (
     <View style={styles.container}>
       <TextInput autoFocus={true} keyboardType='numeric'
placeholder="Bill Amount" onChangeText={this.textChange}
/>
       <View>
         <Button
           title="10%"
           onPress={() => {
             this.calculateTip(0.1);
           }}
         />

         <Button
           title="15%"
           onPress={() => {
             this.calculateTip(0.15);
           }}
         />

         <Button
           title="20%"
           onPress={() => {
             this.calculateTip(0.2);
           }}
         />

         <Button
           title="25%"
           onPress={() => {
             this.calculateTip(0.25);
           }}
         />
       </View>
       <Text>Tip: {this.state.tip}</Text>
       <Text>Total: {this.state.totalValue}</Text>
     </View>
   );
 }
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
   alignItems: 'center',
   justifyContent: 'flex-start',
   paddingTop: 60
 }
});
```

# STYLING

So, the app works, but it's not designed very well. Now's your chance to add some styling! Since some of you may be new, we're going to review some basic elements first.

1

### Defining a style

You can define a style in the constant stylesheet at the bottom (or use external CSS, and import it in). To add a style, give it a name and some properties, like so:

```
title: {
   fontSize: 40,
   fontWeight: '700',
   textAlign: 'center',
   marginBottom: 20
 },
```

2

### Set the style

Now, you need to tell your UI elements which style to use. This can be done by using the style attribute for an element, for example:

```
<Text style={styles.title}>Tip Calculator</Text>
```

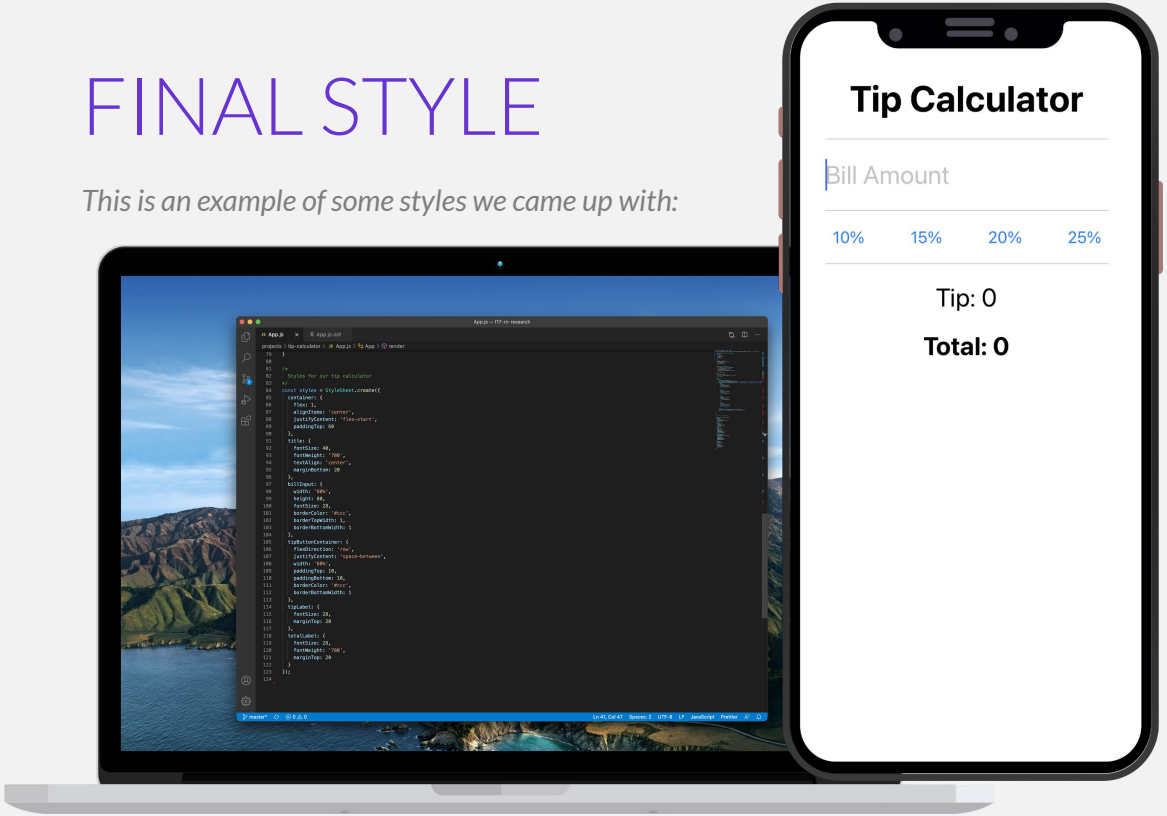
3

### Get to customizing!

Here are some challenges, depending on how comfortable you feel. Change up some of the colors and fonts; try and add an external stylesheet (CSS); check out external styling APIs like Bootstrap, and see if you can figure out how to bring that import in .

# FINAL STYLE

*This is an example of some styles we came up with:*




```
const styles = StyleSheet.create({
 container: {
   flex: 1,
   alignItems: 'center',
   justifyContent: 'flex-start',
   paddingTop: 60
 },
 title: {
   fontSize: 40,
   fontWeight: '700',
   textAlign: 'center',
   marginBottom: 20
 },
 billInput: {
   width: '80%',
   height: 80,
   fontSize: 28,
   borderColor: '#ccc',
   borderTopWidth: 1,
   borderBottomWidth: 1
 },
 tipButtonContainer: {
   flexDirection: 'row',
   justifyContent: 'space-between',
   width: '80%',
   paddingTop: 10,
   paddingBottom: 10,
   borderColor: '#ccc',
   borderBottomWidth: 1
 },
 tipLabel: {
   fontSize: 28,
   marginTop: 20
 },
 totalLabel: {
   fontSize: 28,
   fontWeight: '700',
   marginTop: 20
 }
});
```

# WRAP UP

Hopefully today, you learned a bit more how to make React Native apps interactive and stylish!