App.js

```
// Creating a calculator
/* I wrote the entire code myself but not without help from instructions which I wrote alongside code as comments. The code between
 // Start & // End is the one which I wrote without direct help from instructions*/

// Start
// Import required libraries
import { StatusBar } from 'expo-status-bar';

// Import Dimensions, SafeAreaView and TouchableOpacity from react-native
import { StyleSheet, Text, View, Dimensions, SafeAreaView, TouchableOpacity } from 'react-native';

// Import useState from react, as we'll need it to store our calculator's state
import React, { useState } from 'react';

/* we need to set the width of the button style. To do this we need to calculate a value based on the width of the screen.
Define a new const in the global space which defines the width of buttons based on the correct proportion of the screen width.
Update the button styling to use this height. */
const BUTTON_WIDTH = Dimensions.get('window').width * 0.8;
// End

export default function App()
{
  /*We need React to remember values so that we can make our calculator interactive.
  Firstly, create a new hook for answerValue and setAnswerValue, set the default state to 0.
*/
  const [answerValue, setAnswerValue] = useState(0);

  // This dictates if we should replace what is on screen next time a number is pressed
  const [readyToReplace, setReadyToReplace] = useState(true);

  // Hook to store the value in memory (e.g., the first operand in a calculation)
  const [memoryValue, setMemoryValue] = useState(null);

  // Hook to store the current operator (e.g., +, -, *, /)
  const [operatorValue, setOperatorValue] = useState(null);

  // Start
  const handleNumber = (num) => {
    if (readyToReplace) // check if readyToReplace is true
    {
      setReadyToReplace(false); // if so return the button value. This will cause the calculator to display the pressed button number.
      return num;
    }
    else
    {
      return answerValue.toString() + num; // add an else which simply appends the button value to the end of the answerValue
```

```
    }
    // End
  };

  const calculateEquals = () => {
    // Convert memoryValue and answerValue to floating point numbers
    const previous = parseFloat(memoryValue);
    const current = parseFloat(answerValue);

    // Perform the calculation based on the operatorValue
    // Start
    switch (operatorValue)
    {
      case '+':
        return previous + current;
      case '-':
        return previous - current;
      case '*':
        return previous * current;
      case '/':
        return previous / current;
      default:
        return current;
    }
    // End
  };

  // We need to be able to press buttons, so set up a new function buttonPressed which takes one parameter value.
  const buttonPressed = (value) => {
    if (value === 'C') // This resets everything
    {
      setAnswerValue(0);
      setMemoryValue(null);
      setOperatorValue(null);
      setReadyToReplace(true);
    }
    else if (!isNaN(value)) // check if value is a number
    {
      setAnswerValue(handleNumber(value)); // if it is, set the answerValue to the result of a new function handleNumber
    }
    else if (['+', '-', '*', '/'].includes(value)) // If a value is an operator
    {
      if (operatorValue !== null) //  check if the operator value is 0, Check if there is operator set already
      {
        // If not 0, calculate the result of the previous operation
        const result = calculateEquals(); // if it is not 0, call calculateEquals and save its returns to a new local variable

        // Use this local variable to set the value of the memoryValue later in the statement. This chains the calculations!
        setAnswerValue(result);
```

```
        setMemoryValue(result);
      }
      else
      {
        // If no operator is set, store the current answerValue in memoryValue
        setMemoryValue(answerValue);
      }
      // Prepare to replace the current display value with the next input
      setReadyToReplace(true);
      // Set the current operator to the pressed operator
      setOperatorValue(value);
    }
    else if (value === '=') // Check if the value is '=' (equals)
    {
      // Calculate the result of the current operation and display it
      setAnswerValue(calculateEquals());

      // Reset memoryValue to 0
      setMemoryValue(0);

      // Prepare to replace the current display value with the next input
      setReadyToReplace(true);
    }
    else if (value === '+/-') // Check if the value is '+/-' (negate)
    {
      // Negate the current answerValue
      setAnswerValue(parseFloat(answerValue) * -1);
    }
    else if (value === '%') // Check if the value is '%' (percentage)
    {
      // Convert the current answerValue to a percentage
      setAnswerValue(parseFloat(answerValue) * 0.01);
    }
    else
    {
      // Add a dummy alert to buttonPressed so that you can quickly check if everything is working as expected
      alert(`Button pressed: ${value}`);
    }
  };

  // Start
  return (
    // Wrap the contents of the outer container View in a SafeAreaView to ensure it doesn't overlap with the status bar
    <SafeAreaView style={styles.container}>

      {/* Results field: Create a Text element that is drawn within the ‘main’ View */}
      <Text style={styles.text}>{answerValue}</Text>

      <View style={styles.row}>
        {/* Change the “C” button to “AC” whenever relevant */}
        <TouchableOpacity style={styles.topRowButton} onPress={() => buttonPressed('C')}>
          <Text style={styles.buttonText}>{answerValue !== 0 ? 'C' : 'AC'}</Text>
```

```
147         </TouchableOpacity>
148         <TouchableOpacity style={styles.topRowButton} onPress={() => buttonPressed('+/-')}>
149           <Text style={styles.buttonText}>+/-</Text>
150         </TouchableOpacity>
151         <TouchableOpacity style={styles.topRowButton} onPress={() => buttonPressed('%')}>
152           <Text style={styles.buttonText}>%</Text>
153         </TouchableOpacity>
154         {/* Add indicators to reflect the current operator stored */}
155         <TouchableOpacity style={[styles.sideRowButton, operatorValue === '/' &&
    styles.activeOperator]} onPress={() => buttonPressed('/')}>
156           <Text style={styles.buttonText}>/</Text>
157         </TouchableOpacity>
158       </View>
159
160       <View style={styles.row}>
161         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('7')}>
162           <Text style={styles.buttonText}>7</Text>
163         </TouchableOpacity>
164         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('8')}>
165           <Text style={styles.buttonText}>8</Text>
166         </TouchableOpacity>
167         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('9')}>
168           <Text style={styles.buttonText}>9</Text>
169         </TouchableOpacity>
170         <TouchableOpacity style={[styles.sideRowButton, operatorValue === '*' &&
    styles.activeOperator]} onPress={() => buttonPressed('*')}>
171           <Text style={styles.buttonText}>*</Text>
172         </TouchableOpacity>
173       </View>
174
175       <View style={styles.row}>
176         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('4')}>
177           <Text style={styles.buttonText}>4</Text>
178         </TouchableOpacity>
179         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('5')}>
180           <Text style={styles.buttonText}>5</Text>
181         </TouchableOpacity>
182         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('6')}>
183           <Text style={styles.buttonText}>6</Text>
184         </TouchableOpacity>
185         <TouchableOpacity style={[styles.sideRowButton, operatorValue === '-' &&
    styles.activeOperator]} onPress={() => buttonPressed('-')}>
186           <Text style={styles.buttonText}>-</Text>
187         </TouchableOpacity>
188       </View>
189
190       <View style={styles.row}>
191         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('1')}>
192           <Text style={styles.buttonText}>1</Text>
193         </TouchableOpacity>
194         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('2')}>
195           <Text style={styles.buttonText}>2</Text>
196         </TouchableOpacity>
197         <TouchableOpacity style={styles.button} onPress={() => buttonPressed('3')}>
```

```
          <Text style={styles.buttonText}>3</Text>
        </TouchableOpacity>
        <TouchableOpacity style={[styles.sideRowButton, operatorValue === '+' && styles.activeOperator]} onPress={() => buttonPressed('+')}>
          <Text style={styles.buttonText}>+</Text>
        </TouchableOpacity>
      </View>

      <View style={styles.row}>
        <TouchableOpacity style={styles.longButton} onPress={() => buttonPressed('0')}>
          <Text style={styles.buttonText}>0</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.button} onPress={() => buttonPressed('.')}>
          <Text style={styles.buttonText}>.</Text>
        </TouchableOpacity>
        <TouchableOpacity style={styles.sideRowButton} onPress={() => buttonPressed('=')}>
          <Text style={styles.buttonText}>=</Text>
        </TouchableOpacity>
      </View>

      {/* Set the status bar to be 'light content' so that we can see the phone's status bar on the black background */}
      <StatusBar style="auto" barStyle="light-content"/>
    </SafeAreaView>
  );
  // End
}

// Style the application:
// Start
const styles = StyleSheet.create({
  container:
  {
    flex: 1,
    backgroundColor: 'black', // Set the background colour of the container as black
    alignItems: 'center',
    justifyContent: 'flex-end', // Change the container styling to justify contents so that all child elements align to the bottom of the container vertically
  },
  text:
  {
    color: 'white', // Change text color to white
    fontSize: 48, // Increase font size
    marginLeft:'65%'
  },
  row:
  {
    flexDirection: 'row', // Apply styling to this 'row' so that the child elements are in line with each other horizontally.
    justifyContent: 'space-between', // Adjust spacing between elements
    alignItems: 'center', // Center elements vertically
  },
  button: // Style this TouchableOpacity so that it has a light grey background and appropriate margins
```

```
  {
    backgroundColor: '#555', // Light grey background
    margin: 10, // Appropriate margins
    padding: 10, // Add padding for better touch area
    borderRadius: BUTTON_WIDTH / 8, // set the corner radius to a value that makes the ends of the buttons perfectly circular
    width: BUTTON_WIDTH / 4 - 20, // Adjust & update width for four buttons in a row
  },
  topRowButton:
  {
    backgroundColor: '#333', // Dark background for top row
    margin: 10,
    padding: 10,
    borderRadius: BUTTON_WIDTH / 8, // Make the button ends perfectly circular
    width: BUTTON_WIDTH / 4 - 20, // Adjust width for four buttons in a row
  },
  sideRowButton:
  {
    backgroundColor: 'orange', // Blue background for side row
    margin: 10,
    padding: 10,
    borderRadius: BUTTON_WIDTH / 8, // Make the button ends perfectly circular
    width: BUTTON_WIDTH / 4 - 20, // Adjust width for four buttons in a row
  },
  longButton:
  {
    backgroundColor: '#333', // Light grey background
    margin: 10,
    padding: 10,
    borderRadius: BUTTON_WIDTH / 8, // Make the button ends perfectly circular
    width: BUTTON_WIDTH / 2 - 20, // Adjust width to be twice the size of a regular button
  },
  buttonText:
  {
    color: 'white', // Set text color to white
    fontSize: 24, // Set the font size
    textAlign: 'center', // Center align the text
  },
  activeOperator:
  {
    backgroundColor: 'red', // Change to a different color to indicate active operator
  },
  // End
});
```