

# React Native

While you wait:

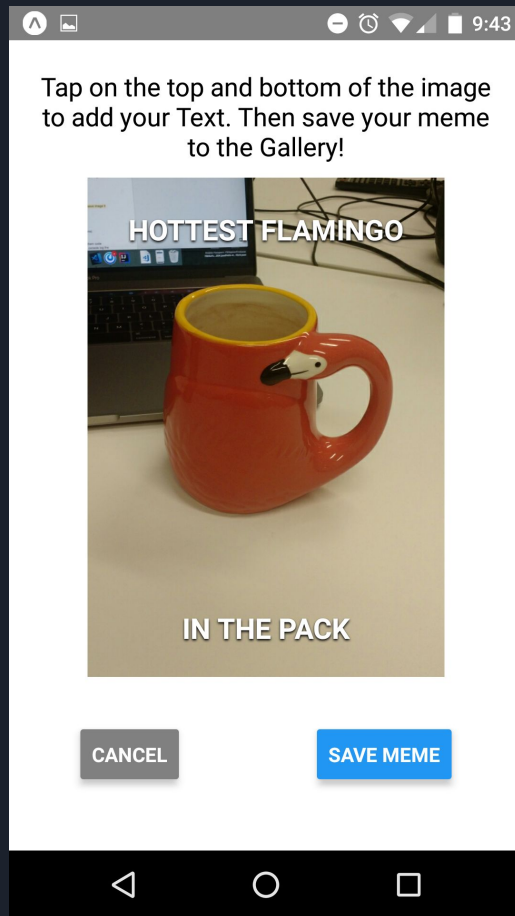
- Download The Expo App from the Play Store or App Store
- Slides - [ap.pn/2p90qqo](https://ap.pn/2p90qqo)



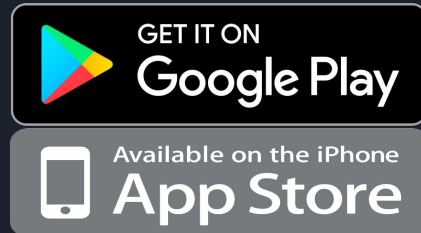
# What are we building?

## A Meme Creator App

1. Take a picture
2. Add text
3. Save your meme
4. Impress your friends with your original dank memes!



# How are we building it?



## Expo + Snack

- Online IDE created by Expo for React Native development
- You can create an app in Snack and share it easily with a QR code
- To run it on your phone you just need the corresponding Expo app from the Play or App Store

If you haven't already, open the slides: [ap.pn/2p90qqo](https://ap.pn/2p90qqo)



# Why React Native?

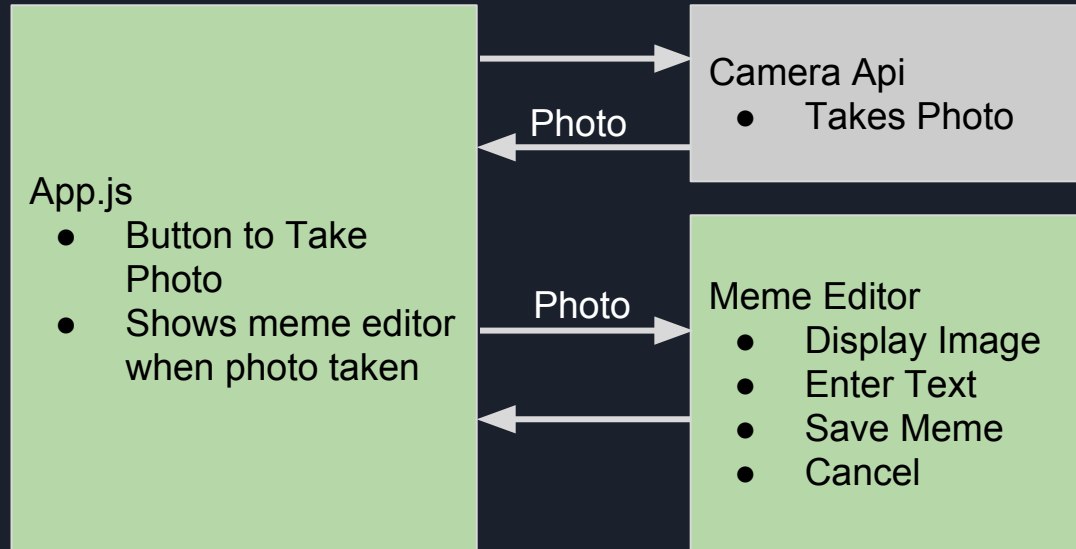
Uses React Lifecycle

Cross-Platform

Reusable Components (Fast to write)

Simple State Management

# Structure of our App





# Table of Contents

## Section 1: Let's take a picture!

- What is a Component
- Simple React Component Lifecycle
- Console Logs
- Waiting on an API
- Demo: Taking a Picture

## Section 2: Let's see our image!

- State + Props
- Intro to Component: View
- Intro to Component: Image
- Intro to Component: Text
- Demo: Showing your Image

## Section 3: Let's add text!

- Styling your Application
- Intro to Component: TextInput
- Demo: Adding Text

## Section 4: Let's save it!

- Intro to Component: CameraRoll
- Demo: Saving your Meme

## Additional Concepts

- React Component Lifecycle
- More on Building Components
- More Styles + Layout with Flex
- Sharing Code Across Files in JS
- Variables - const, let & var
- Set-up Local Development



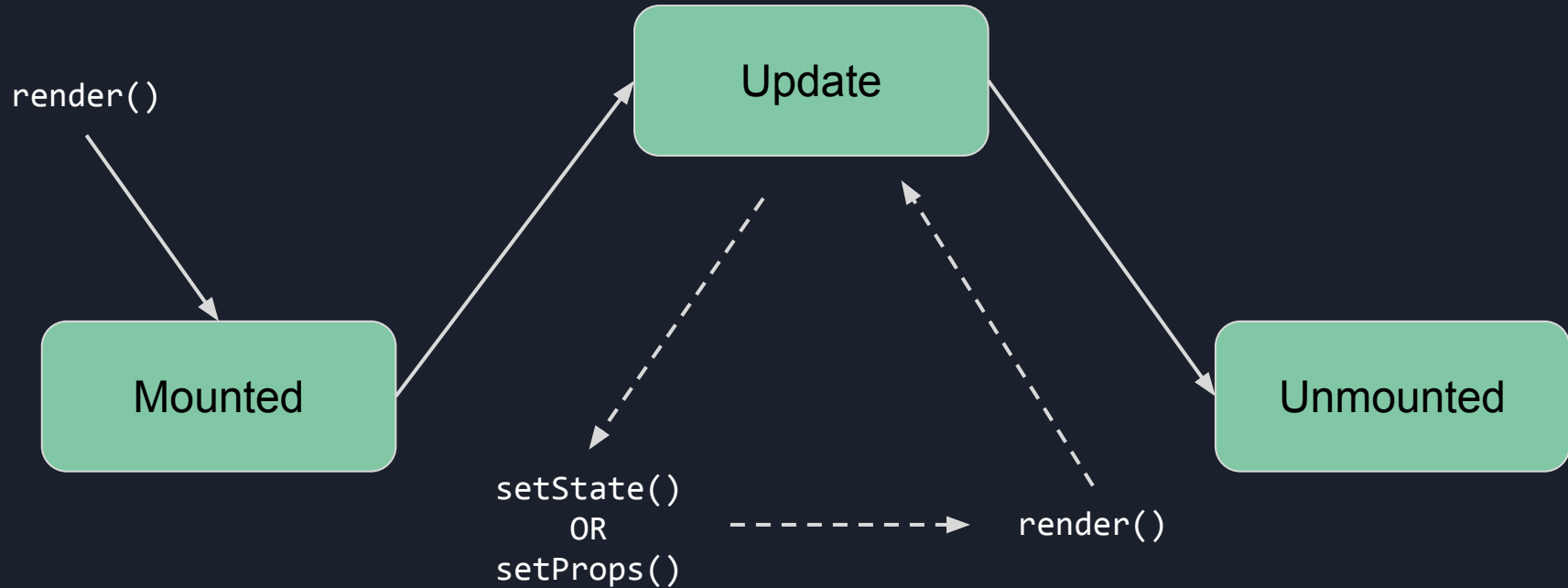
# What is a Component

```
class Greeting extends React.Component {  
  render() {  
    return <Text>Hello, {this.props.name}</Text>;  
  }  
}
```

Components let you split the UI into independent, reusable pieces, and think about each piece in isolation - React Docs

[React Component Documentation](https://reactjs.org/docs/react-component.html)

# Simplified React Lifecycle







# Javascript Functions

## Normal JS Functions

```
function myFunction() {  
  return <Text>Hello, World!</Text>;  
}
```

## Arrow Functions (aka “Fat Arrow” Functions)

```
myBoundFunction = () => {  
  return <Text>Hello, {this.props.world}</Text>;  
}
```

[Normal JS Functions](#)

[Arrow Functions](#)

# Demo Concepts



# Console Logs are our Friends

When you are debugging try using `console.log()`; to print out variables and see where your code is breaking.

Click on the bottom bar in expo to open the log/error drawer.





# Waiting on an API

Some functions may take time to return a result (networking, user interaction, slow processing). We use `async` & `await` for these occasions. (The docs for an api will tell you if it is needed).

The Camera Api is an example of this!

```
takePhoto = async () => {  
  console.log('taking photo');  
  let img = await Expo.ImagePicker.launchCameraAsync();  
  if (!img.cancelled) {  
    console.log('photo taken');  
    this.setState({ photo: img, screen: memeScreen });  
  }  
};
```

our function must have the  
“`async`” keyword to have an `await`

`await` the image to return

now we can use the image!



# Demo: Taking a Picture

## Key Concepts:

- Async and Await
- Expo Console for logs

## Steps:

1. Use the `launchCameraAsync()` function to take a photo.
2. We'll console log the result to make sure it's taken!

[Start coding on Expo from here](#)

[See the changes on Github](#)

[End Result](#)

Welcome to the MemeCreator!  
Lets build a meme.

TAKE A NEW PICTURE

(Android Only): We need external  
storage permission and Expo  
doesn't gracefully give it to us:

ALLOW EXTERNAL STORAGE ACCESS

# Adding Components: State and props

**Props:** “parameters” to a constructor.

Values that we pass into our component\*

**State:** “fields” in a class

Changing these causes the component to rerender

## Props Passed from App.js

```
export default class App extends Component {  
  ...  
  render() {  
    return <OnOffSwitch color='blue'>;  
  }  
}
```

## Props & State in OnOffSwitch

```
export default class OnOffSwitch extends Component {  
  constructor(props) {  
    super(props);  
    // Set Initial State  
    this.state = { isOn: false };  
  }  
  render = () => { // Render method that controls what shows on UI  
    const buttonText = this.state.isOn ? "ON" : "OFF";  
    const buttonColor = {backgroundColor: this.props.color};  
    return (  
      <Text onPress={this.onPress}>  
        {buttonText}  
      </Text>  
    );  
  }  
  onPress = () => {  
    this.setState({ isOn: !this.state.isOn });  
  }  
}
```

## Basic

[View](#)  
[StyleSheet](#)  
[Image](#)  
[Text](#)  
[TextInput](#)

## User Interface

[Button](#)  
Picker  
Switch

## iOS

DatePickerIOS  
AlertIOS  
NavigatorIOS

## Android

ToastAndroid  
BackHandler  
ToolbarAndroid

## Others

Animated  
Modal  
[CameraRoll](#)

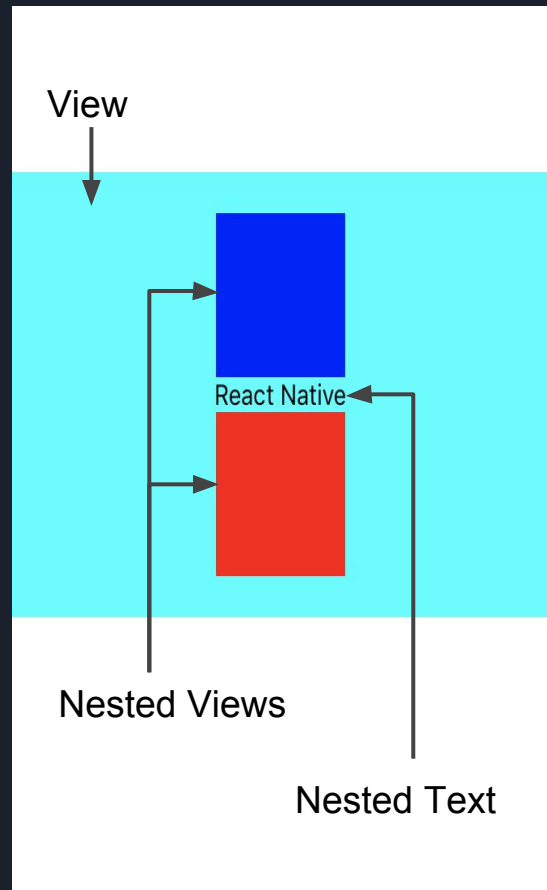


# React Native Components

# Basic Components: View

- A container for laying out your UI
- You can nest other Views or components inside
- Styling is done with the “style prop”

```
<View style={{backgroundColor: 'cyan', ... }}>  
  <View style={{backgroundColor: 'blue', ... }} />  
  <Text>React Native</Text>  
  <View style={{backgroundColor: 'red', ... }} />  
</View>
```

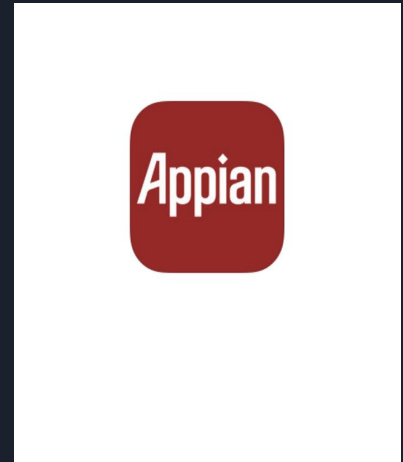




# Basic Components: Image

- Pass in a url to a local image or image on the web. And it will display it!
- Manually specify dimensions for images
- Displays most image formats\*

```
<View style={{...}}>  
  <Image  
    style={{width: 150, height: 150}}  
    source={{uri: 'https://d7um....png'}} />  
</View>
```



# Basic Components: Text

- Displays text
- Supports **nesting**, **styling** and **touch handling**

```
<View style={{...}}>  
  <Text onPress={() => Alert.alert('Incredibly awesome!')}>  
    RN Workshop  
    <Text style={{color: 'red', fontSize: 20}}>  
      {' '}is awesome?  
    </Text>  
  </Text>  
</View>
```

RN Workshop **is awesome?**

# Demo: Showing your image

## Key Concepts:

- View and Image Components
- State and Props

## Steps:

1. Update the state that determines what screen to show
2. Display the image we took
3. TODO

[Start coding on Expo from here](#)

[See the changes on Github](#)

[End Result](#)



# Styles: Basics

- Let you customize your components
- We pull our styles out of our JSX, and into stylesheets\*

```
<View style={{
  padding: 20,
  alignItems: 'center',
  marginTop: 100
}}>
  <Image
    style={{width: 150, height: 150}}
    source={{uri: '...'}} />
</View>
```

=

```
const styles =
StyleSheet.create({
  viewStyle {
    padding: 20,
    alignItems: 'center',
    marginTop: 100
  },
  imageStyle {
    width: 150,
    height: 150
  }
});
```

+

```
<View style={styles.viewStyle}>
  <Image
    style={styles.imageStyle}
    source={{uri: '...'}} />
</View>
```

\*refer to slide notes for details

# Styles: Stylesheets

## Why Stylesheets?

- Makes code readable
- Reusable
- Easier to maintain

```
const styles =
StyleSheet.create({
  viewStyle: {
    padding: 20,
    alignItems: 'center',
    marginTop: 100
  },
  imageStyle {
    width: 150,
    height: 150
  }
});
```

```
<View style={styles.viewStyle}>
  <Image
    style={styles.imageStyle}
    source={{uri: '...'}} />
</View>
```

# Basic Components: TextInput

- Input text in App via keyboard

```
<View style={{...}}>
```

```
<TextInput
```

```
  style={{...}}
```

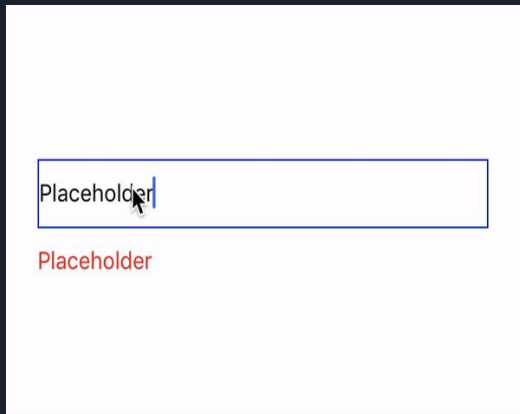
```
  onChangeText={(text) => this.setState({text})}
```

```
  value={this.state.text}
```

```
/>
```

```
<Text style={{...}}>{this.state.text}</Text>
```

```
</View>
```



# Demo: Adding Text to your Meme

## Key Concepts:

- Styling
- State and Props

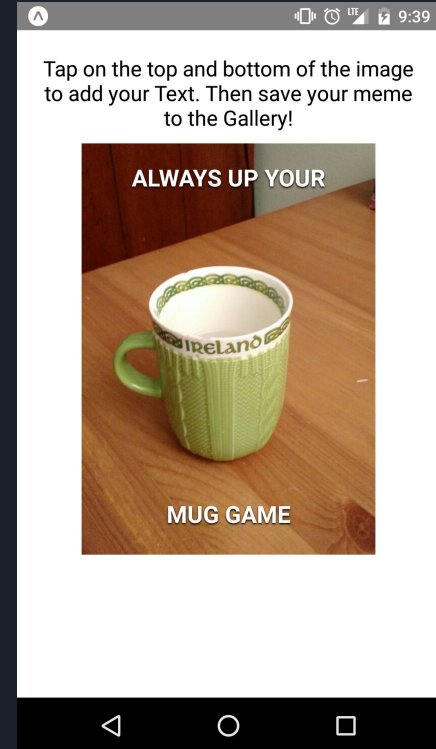
## Steps:

1. Change Image to ImageBackground
2. Add two text fields on our image + styles

[Start coding on Expo from here](#)

[See the changes on Github](#)

[End Result](#)





# CameraRoll

- Access to device's camera roll
  - `saveToCameraRoll()`
    - saves photo/video to camera roll
  - `getPhotos()`
    - Gets photo from the camera roll

```
const result = await takeSnapshotAsync (this.ref, {  
  format: 'png',  
  result: 'file',  
});  
await CameraRoll.saveToCameraRoll (result, 'photo');
```



# Demo: Saving Your Meme

## Key Concepts:

- State and Props

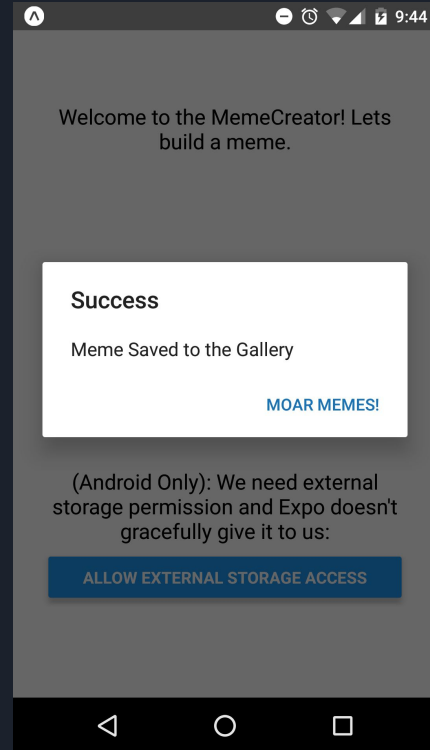
## Steps:

1. Add a save button with an onPress Listener
2. Save the image and close the editor, in our “onPress” callback

[Start coding on Expo from here](#)

[See the changes on Github](#)

[End Result](#)



# Thank you for joining us!

Please [click here](#) or go to [ap.pn/2oYN4Nx](https://ap.pn/2oYN4Nx) to share feedback

*Next Step: Add a cancel button that just closes the editor*

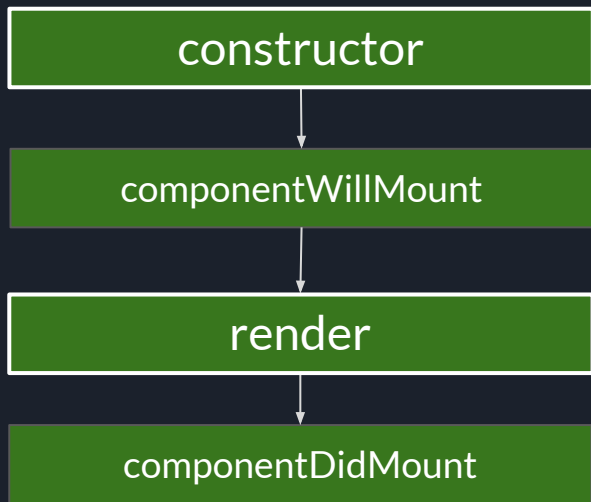
[Start coding on Expo from here](#)

# Additional Concepts

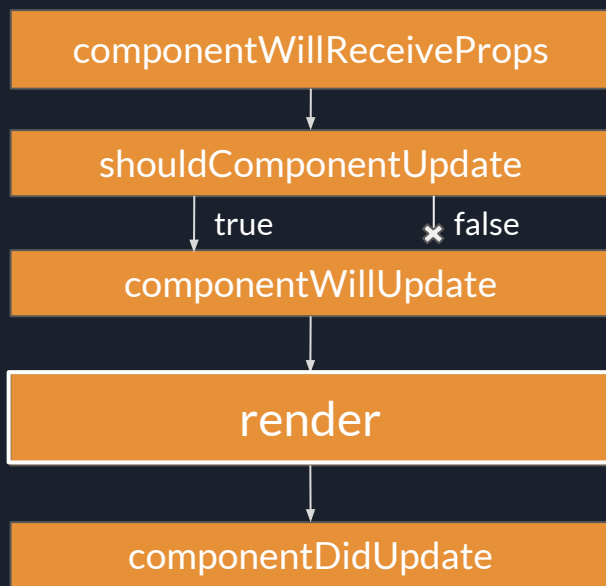


# Detailed React Component Lifecycle

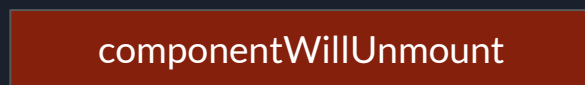
## Mounting (Creating)



## Updating



## Unmounting (Destroying)



# Adding Components

Let's build an "on-off" button

We will put together our requirements, and find ways to achieve them.

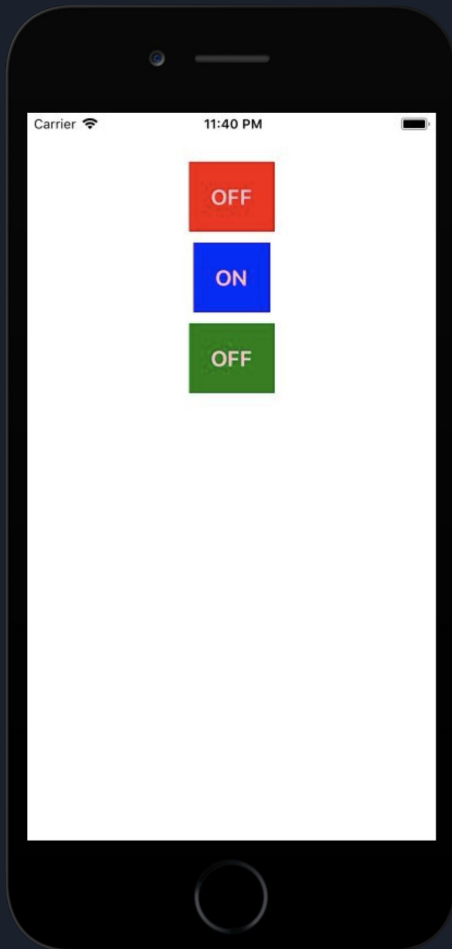
## Requirements

- We want a reusable button
- Button color can be configured
- You can turn the button on and off
- Text switches when you press

## Technical Approach

- Make a component
- Pass color in as a prop
- Have a state variable
- An onPress listener is needed

Try to think about your components this way!



# Adding Components: Building onOffSwitch

```
import React, { Component } from 'react';
import { Text, TouchableOpacity, StyleSheet } from 'react-native';
```

```
export default class OnOffSwitch extends Component {
```

```
  constructor(props) {
    super(props);
    this.state = { isOn: false };
  }
```

```
  toggle = () => {
    this.setState({ isOn: !this.state.isOn });
  }
```

```
  render = () => {
    const buttonText = this.state.isOn ? "ON" : "OFF";
    const buttonColor = { backgroundColor: this.props.color };
    return (
      <TouchableOpacity style={[styles.button, buttonColor]}
        onPress={this.toggle} >
        <Text style={styles.buttonText}>{buttonText}</Text>
      </TouchableOpacity>
    );
  }
}
```

Imports

Class Creation + export (making it visible to other files)

Initial state

Callbacks + Changing State

Props

Render + other lifecycle methods

# Styling: Basics

## Create Your Styles

```
const styles = StyleSheet.create({  
  button: {  
    padding: 20,  
    Margin: 5,  
    backgroundColor: 'blue'  
  },  
  buttonText: {  
    fontSize: 20,  
    fontWeight: 'bold',  
    color: 'pink'  
  }  
});
```

## Component with style applied

```
<TouchableOpacity style={styles.button}>  
  <Text style={styles.buttonText}>  
    { 'On' }  
  </Text>  
</TouchableOpacity>
```

## Rendered view



2 approaches to finding styles:

- Look through the style docs for [Text](#) and [View](#) (do this at least once)
- Google “how to do <this particular style> in React Native”

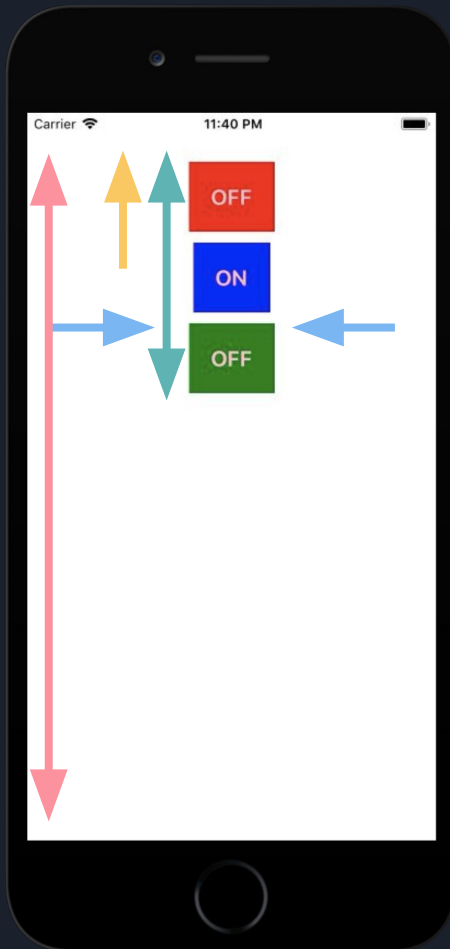
# Styling: Laying out views

Flex is a set of styles for laying out views in React Native

- **flexDirection**: row vs column
- **justifyContent**: alignment in flexDirection
- **alignItems**: alignment in cross-Direction
- **flex**: how big the view should be (relative to its neighbors)
  - Here, the container has no neighbors  
So it takes up the full height

```
container: {  
  flexDirection: 'column',  
  justifyContent: 'start',  
  alignItems: 'center',  
  flex: 1  
}
```

These are the basics, if you want to know more [check this flex game out](#).





# Sharing Code across files in JS

We split our js code into different files for cleanliness, these are called modules. To share code with other files we need to “**export**” it. Then, other files can “**import**” what they need.

## ModuleOne.js

```
export const number1 = 1;
export const number2 = 2;
default export function add(x, y) {
  return x + y;
}
```

## ModuleTwo.js

```
import add, {number1, number2} from
  './ModuleOne.js';

function solve() {
  return add(number1, number2);
}
```

You can export multiple things from a file. But only one labeled “**default**”. The non “default” exports are listed in curly brackets. The “default” one is not.



## Variable Declarations: Const, Let, Var

<b>const</b>	Variable is instantiated once and cannot be changed
<b>let</b>	Variable can be changed and is scoped to the nearest enclosing block
<b>var</b>	Variable can be changed and scoped to the nearest function block

# Setup Local Development

1. Install [node.js + yarn](#)
2. Install [Expo XDE](#) (App) or the [Expo Command Line Tool](#)
3. “Export” your project from snack
4. Unzip the download
5. Open the folder in a code editor
6. In the Command Line, go to the root of the project and run “yarn install”
7. If using the command line tool:
  - a. Run “exp start” and scan the barcode shown
8. If using the Expo XDE:
  - a. Open the project in Expo XDE and “Share” it to your device.
  - b. Scan the displayed barcode with the expo app on your phone.

