See the tutorials at https://facebook.github.io/react-native/docs/tutorial

React native

JSX, components, props, and state

An overview

We'll at each of these in much more detail later.

JSX

- JSX = JavaScript + XML
  - XML is a tagging system similar to HTML
  - Actually uses ES2015 (also called ES6), not JavaScript (ES5).
    - ES = ECMAScript
  - **`Import, from, class, extends`** are all ES6 features
  - See this link for ES6 features: https://babeljs.io/docs/en/learn/
- JSX allows us to embed XML in JavaScript
  - In HTML we embed JavaScript in HTML

Components

- Components are ”pieces” that fit together to make an app
  - Conceptually, components are like JavaScript functions
  - They split the UI into independent, reusable pieces
  - Components are made of “elements” or pieces of JSX code
  - Different components implement different types of UI elements like text or a button.
  - You can make your own components by `extending` built-in components (this is why we looked at objects in JS).

We’ll look more closely at components later.

Components

- There are many available components in these categories:
  - Basic Components
  - User Interface
  - List Views
  - iOS-specific
  - Android-specific
  - Others
- Basic React Native components can be found here:
  - https://facebook.github.io/react-native/docs/components-and-apis.html
- Dev's have also created components that you can include. See
  - http://www.awesome-react-native.com/#components

We'll look more closely at components later.

Hello world

Must import everything you use; import is ES6 syntax

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

export default class HelloWorldApp extends Component {
  render() {
    return (
      <View>
        <Text>Hello world!</Text>
      </View>
    );
  }
}
```

Component: often something you see on the screen

Notice the class syntax: ”class” and “extends”

render() causes the component to be displayed

JSX: XML embedded in JavaScript

This is in App.js

```
{ Component } from 'react’;
// this is destructuring syntax from ES6
// same as:
Import React from “react”;
Let Component = React.Component;
```

props

- props = properties
- Most *components* can be customized when they are created, with different parameters.
- These creation parameters are called `props`.
- Once used, `props` cannot be changed (see *state* on a later slide)

props

- `props` can be used in your own components.
- `props` make a component reusable in your app,
  - Can have different properties in each use.
  - Like an instance variable
  - refer to `this.props` in your render function to access the values passed through `props`.

Props I

```
import React, { Component } from 'react';
import { AppRegistry, Image } from 'react-native';

export default class Bananas extends Component {
  render() {
    let pic = {
      uri:
'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'
    };
    return (
      <Image source={pic} style={{width: 193, height: 110}}/>
    );
  }
}
```

Notice that {pic} is surrounded by braces inside the **`render()`** function,
This embeds the variable **`pic`** into JSX.

You can put any JavaScript expression inside braces in JSX.

render() returns the React elements to be displayed. Normally contains JSX

"**`let`**" defines a variable "pic" of type "uri"

See this link for info about the Image component: https://facebook.github.io/react-native/docs/image.html

"**`source`**" is a prop for the Image component

Code at: https://facebook.github.io/react-native/docs/props

Replace the code in App.js with the code on this slide

Props II

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native’;
class Greeting extends Component {
  render() {
    return (
      <Text>Hello {this.props.name}!</Text>
    );
  }
}
```

Must import everything you use; import is ES6 syntax

render() returns the React elements to be displayed. Normally created via JSX

Can add styles directly to a text component

Continued on next slide

Props II (cont)

```
export default class LotsOfGreetings extends Component {
 render() {
  return (
   <View style={{alignItems: 'center'}}>
    <Greeting name='Rexxar' />
    <Greeting name='Jaina' />
    <Greeting name='Valeera' />
   </View>
  );
 }
}
```

export makes this component available in the app

Notice the use of a style just like inline CSS

A **`View`** component is a container for other components, to help control style and layout.

The **`Greeting`** component returns a **`Text`** component which is embedded in the **`View`** component.

We create three *instances* of the **`Greeting`** component (previous slide) using *props* to instantiate the instance variable “name”

Replace the code in App.js with the code on this slide and previous slide

App

- To build a static app just need
  - Props
  - Text component
  - View component
  - Image component
- Dynamic apps require *state*

State vs Props

- The state is mutable while props are immutable.
  - This means that state can be updated in the future while props cannot be updated.
- *Presentational components* should get all data by passing props.
- Only *container components* should have state.

State

- initialize state in the constructor
- call setState when you want to change it.

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Blink extends Component {
  constructor(props) {
    super(props);
    this.state = {isShowingText: true};

    // Toggle the state every second
    setInterval(() => {
      this.setState(previousState => {
        return { isShowingText:
!previousState.isShowingText };
      });
    }, 1000);
  }
 render() {
    let display = this.state.isShowingText ?
this.props.text : ' ';
    return (
      <Text>{display}</Text>
    );
  }}
export default class BlinkApp extends Component {
  render() {
    return (
      <View>
        <Blink text='I love to blink' />
        <Blink text='Yes blinking is so great' />
        <Blink text='Why did they ever take this out of
HTML' />
        <Blink text='Look at me look at me look at me' />
      </View>
    );}}
```

See next slide for explaination

```
import React, { Component } from 'react';
import { AppRegistry, Text, View } from 'react-native';

class Blink extends Component {
  constructor(props) {
    super(props);
    this.state = {isShowingText: true};


    // Toggle the state every second
    setInterval(() => {
      this.setState(previousState => {
        return { isShowingText: !previousState.isShowingText };
      });
    }, 1000);
  }
 render() {
    let display = this.state.isShowingText ? this.props.text : ' ';
    return (
      <Text>{display}</Text>
    );
  }}
```

Class **`Blink`** inherits from Component so it becomes a component.

Classes have a **`Constructor`** where you initialize state.

The => syntax is a function shorthand. Here we define the function **`setInterval`** which takes no parameters. See https://babeljs.io/docs/en/learn/

`setState` takes a value and an optional callback function. Here we return a new value for **`isShowingText.`** The render() function will be called to update the component.

`display` takes a value based on the value of **`isShowingText`**. It either uses the value of the prop **`text`** or display gets the empty string.

setState()

- **`setState()`** enqueues changes to the component state and tells React that this component and its children need to be re-rendered with the updated state.
  - This is the primary method you use to update the user interface in response to event handlers and server responses.
- **`setState()`** is a *request* not an immediate command to update the component.
  - React may delay the update and then update several components in a single pass.
  - React does not guarantee that the state changes are applied immediately.

Example, explained

- probably won't be setting state with a timer in general.
- Do set state when:
    - new data arrives from the server,
    - or from user input.

- can also use a state container like Redux or Mobx to control your data flow.
    - Then would use Redux or Mobx to modify your state rather than calling setState directly.

- Example on previous slide:
    - When setState is called, BlinkApp will re-render its Component (the render method is called).
    - By calling setState within the Timer, the component will re-render every time the Timer ticks.