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

React native

State

An overview

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

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

setInterval()

- Used in JavaScript to set an alarm clock.
  - Part of the Window object
  - Give it an interval and the name of a function
  - The browser counts down the time. When reach 0, the function is called.
  - See https://www.w3schools.com/jsref/met_win_setinterval.asp

- **`setInterval(function, milliseconds[, param1, param2, ...])`**
  - *function*. Required. The function that will be called
  - *milliseconds*. Required. The intervals (in milliseconds) on how often to call the function. If the value is less than 10, the value 10 is used
  - [, *param1, param2*, …]. Optional. Additional parameters to pass to the *function*

`setInterval` example

```
var myVar;

function myFunction() {
    myVar = setInterval(alertFunc, 3000);
}

function alertFunc() {
    alert("Hello!");
}
```

Can use **`myVar`** to turn off the timer.

Arrow syntax

The => syntax is a function shorthand. See https://babeljs.io/docs/en/learn/

```
(param1, param2, …, paramN) => { statements }

(param1, param2, …, paramN) => expression   // equivalent to: => { return expression; }

 // Parentheses are optional when there's only one parameter name:

          (singleParam) => { statements }

          singleParam => { statements }

// The parameter list for a function with no parameters should be written with a pair of parentheses.

          () => { statements }
```

Arrow syntax example

```
var elements = [ 'Hydrogen', 'Helium', 'Lithium', 'Beryllium' ];

 elements.map(function(element ) {
                  return element.length;
}); // [8, 6, 7, 9]

elements.map(element => {
      return element.length;
}); // [8, 6, 7, 9]

elements.map(element => element.length); // [8, 6, 7, 9]
```

map is a built-in function of arrays. It applies the passed function to each element of the array and returns an array of the results.

This the exact same thing as the previous line of code.

This the exact same thing as the previous line of code. Note that we don't need the "return", the result is automatically returned.

Arrow syntax example

```
elements.map(({ length }) => length); // [8, 6, 7, 9]
```

This does the exact same thing as the previous examples. Just note how weird the syntax can get. It relies on destructuring and the fact that **`length`** is actually a function.

```
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 can initialize state.

`setInterval()` takes 2 arguments (and some optional ones).

1st argument: a function that takes no parameters.

2nd argument: a time until call the function (in milliseconds; 1000ms=1sec). Repeat forever.

setInterval takes 2 paramters: function and a value

```
// Toggle the state every second
setInterval(
      () => {
  this.setState(previousState => {
    return { isShowingText: !previousState.isShowingText };
  });
}, 1000);
}
```

Anonymous function that takes 0 parameters

`setState` takes a function that returns a new object. Function can have 1 or 2 parameters (see next slide)

Second argument to **`setInterval`**.

setState(updater[, callback])

- **`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.

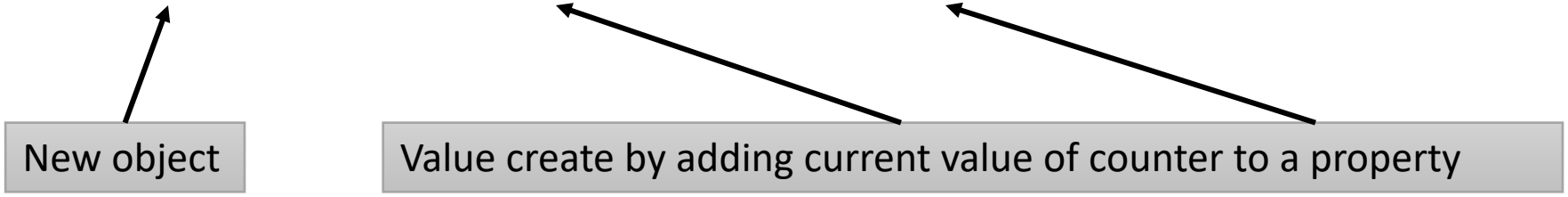
setState(updater[, callback])

- **`updater`** first (required) argument is a function with the signature:
  - (state, props) => stateChange
  - state is a reference to the component state at the time the change is being applied.
  - State should not be directly mutated.
  - Instead, changes should be represented by building a *new* object based on the input from **`state`** and **`props`**.
  - If a variable is created in the constructor but not mentioned in the updater, it remains in the state with its value unchanged.

setState(updater[, callback])

- For instance, suppose we wanted to increment a value in state by **`props.step`**:

```
this.setState((state, props) => {
        return {counter: state.counter + props.step};
});
```



`props` contains all the properties that you created in this component

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.

Example 2

Based on tutorialspoint, but their example is wrong!!

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

class Home extends Component {
  constructor(props){
    super(props);
    this.state = { myState: 'The origninal state', isOrig: true};
  }
  updateState = () => {
      if (this.state.isOrig){
            this.setState({ myState: 'The state is updated', isOrig:
false })
      }
      else{
            this.setState({ myState: 'The original state ', isOrig:
true })
      };
  };

  render() {
    return (
          <Text onPress = {this.updateState}>
            {this.state.myState}
          </Text>
    );
  }
}

export default class homeApp extends Component {
  render() {
    return (
      <View style={{alignItems: 'center', marginTop: 100}}>
        <Home />
      </View>
    );
  }
}
```

Note that component/class names must start with a capital letter!

Home class

```
class Home extends Component {
 constructor(props){
   super(props);
   this.state = { myState: 'The origninal state', isOrig: true};
 }
 updateState = () => {
     if (this.state.isOrig){
         this.setState({ myState: 'The state is updated', isOrig: false })
     }
     else{
         this.setState({ myState: 'The original state ', isOrig: true })
     }; {/* end of else */}
 }; {/* end of updateState function*/}
```

1

2

3

4

```
render() {
  return (
      <Text onPress = {this.updateState}>
        {this.state.myState}
      </Text>
    );
  }
}
```

Text responds to different events

Same syntax, different meanings!

Can use the arrow syntax for named functions also

If a variable is not mentioned in setState, its value is not changed but it remains in the state

Export class

```
export default class BlinkApp extends Component {
  render() {
    return (
      <View style={{alignItems: 'center', marginTop: 100}}>
        <Home />
      </View>
    );
  }
}
```

Just creates an instance of the Home class

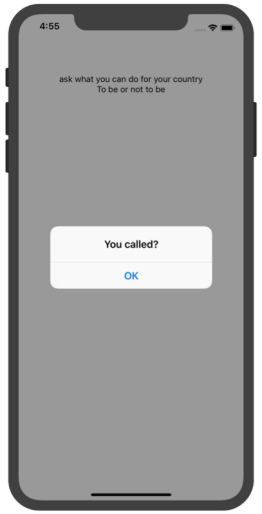
Example 3 (on 3 slides)

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

class Home extends Component {
  constructor(props){
    super(props);
   {/* State has 3 variables */}
    this.state = { myState: this.props.origText, myState2: this.props.origText2, isOrig: true};
  }
```

This is how to create comments. Note that you can put comments inside functions but not between functions

This app has two lines of text. Clicking on each changes only that line of text.



Example 3 (slide 2)

```
updateState = () => {
      Alert.alert('You called?');
      if (this.state.isOrig){
            this.setState({ myState: 'ask what you can do for your country', isOrig: false })
      }
      else{
            this.setState({ myState: this.props.origText, isOrig: true })
      };
  };
```

```
updateState2 = () => {
      Alert.alert('You called?');
      if (this.state.isOrig){
            this.setState({ myState2: 'that is the question', isOrig: false })
      }
      else{
            this.setState({ myState2: this.props.origText2, isOrig: true })
      };
  };
```

Note that each function updates state in a different way and leaves a variable in the state unchanged.

Example 3 (slide 3)

```
    render() {
     return (
       <View style={{alignItems: 'center', marginTop: 100}}>
         <Text onPress = {this.updateState}>
           {this.state.myState}
         </Text>
         <Text onPress = {this.updateState2}>
           {this.state.myState2}
         </Text>
       </View>
     );
   }
}
```

```
export default class homeApp extends Component {
  render() {
    return (
       <Home origText='Ask not what your country can
do for you' origText2='To be or not to be' />
    );
  }
}
```

There seems to be no way to pass a parameter to an event handler. So I created two different event handlers here.

State vs instance variables

- State is passed back to React Native with the JSX when the render() function returns.
- Instance variables are *not* passed back.
  - But they’re known by the instance of the class
  - They can be used locally
  - Then cannot be used in JSX code that is returned

```
import React, { Component } from 'react';
import { AppRegistry, Text, View, Alert, Button } from 'react-native’;
class Home extends Component {
 constructor(props){
   super(props);
   // Declaration of a local variable
   this.x = 0;
    this.y = “Something”;
  // creation of state
   this.state = { myState: this.props.origText, isOrig: true};
 }
 updateState = () => {
     if (this.state.isOrig){
          this.setState({ myState: 'ask what you can do for your country', isOrig: false })
     }
     else{
          this.setState({ myState: this.props.origText, isOrig: true })
     };
 };
```

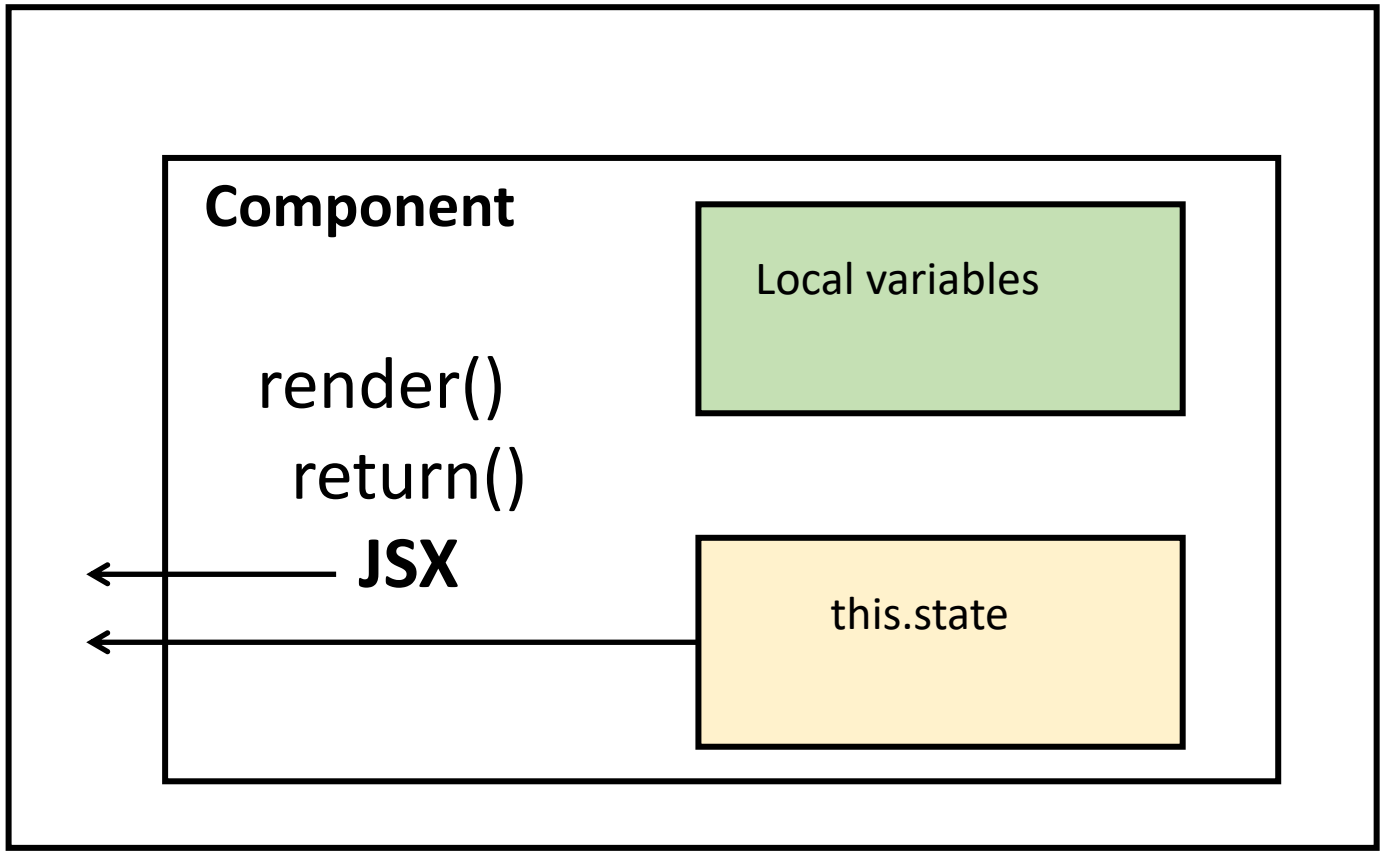
Instance variable known by this class

State passed back with the JSX

```
testFunction = () => {
    Alert.alert("x = " + this.x + " and this.y = " + this.y);
  };
```

State vs instance variables

React Native



`JSX` and **`this.state`** known to React Native environment

Local variables are not known

```
  render() {
    // can update local variables here
    this.x = this.x + 1;
    this.testFunction();
   // updateState();
     return (
     // Next line won't work.  Cannot reference an instance
     // variable in JSX code.  That's why we have state
     //{this.x = this.x + 1;}
     <View>
        <Text onPress = {this.updateState}>
          {this.state.myState}
        </Text>
        <Button
           title="update"
           onPress={this.updateState}
        />
     </View>
    );
  }
}
```

Can use Instance variables in any code executed by the class. The **`testFunction`**() will be called everytime **`render`**() is called.

Cannot call updateState() in render(); would be called everytime render() is called!

Cannot use instance variables in the JSX passed back to React

```
export default class StateApp extends Component {
 render() {
   return (
     <View style={{alignItems: 'center', marginTop: 100}}>
       <Home origText='Ask not what your country can do for you'/>
     </View>
   );
 }
}
```