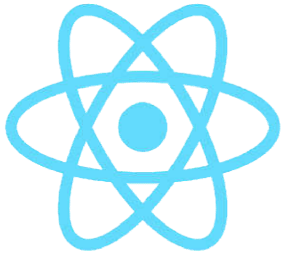


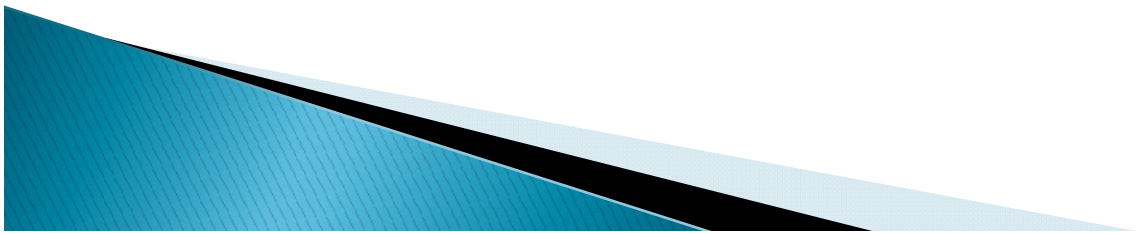
React Native

พัฒนา Mobile Apps ด้วย React Native

โค้ชเอก

Codingthailand.com

Understanding React

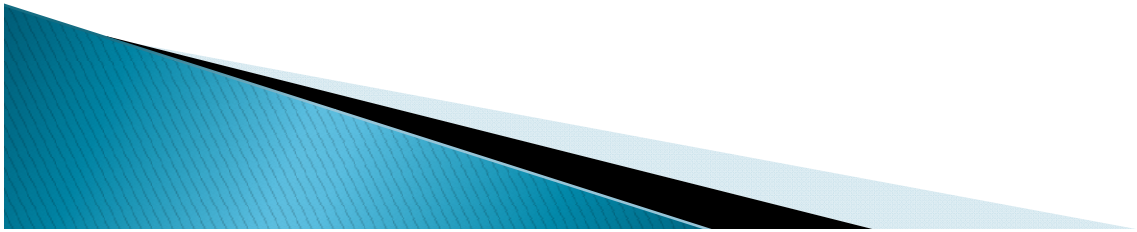


React คืออะไร

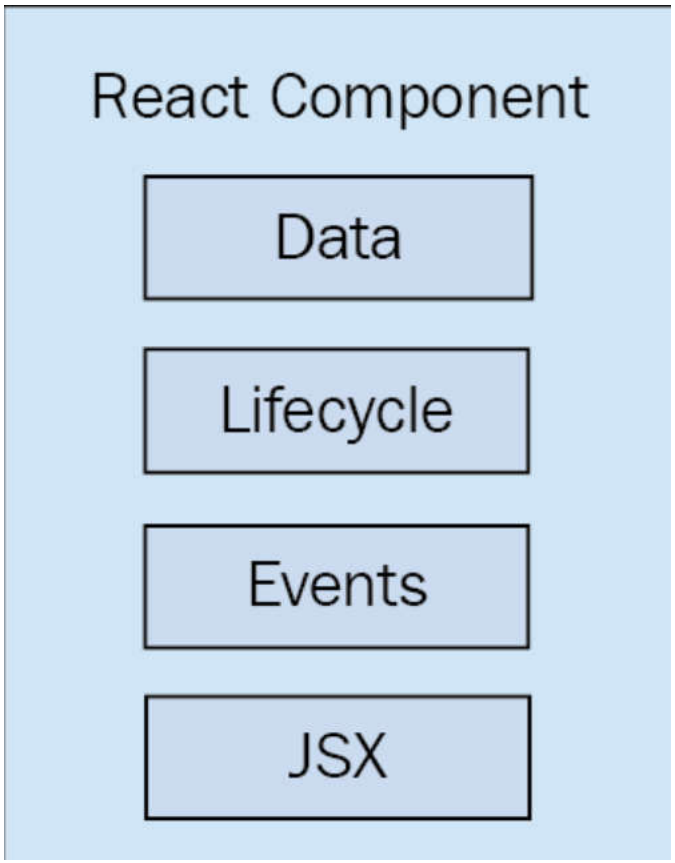
- **React** is a JavaScript library open sourced by and used within Facebook, and was originally used to **build user interfaces** for web applications.

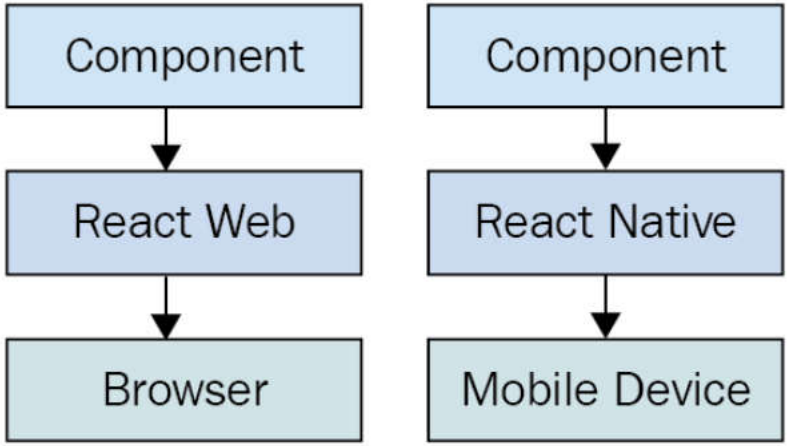
- **React** is a JavaScript library for building user interfaces across a variety of platforms.

- เว็บไซต์หลัก: https://reactjs.org/



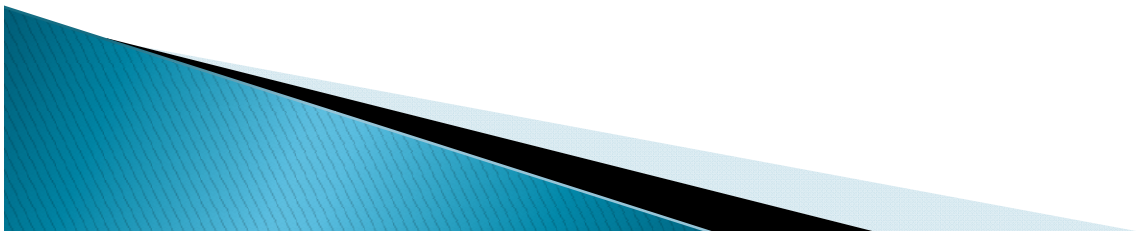
Components


React Component
Data
Lifecycle
Events
JSX


Component
Component
React Web
React Native
Browser
Mobile Device

React Concept

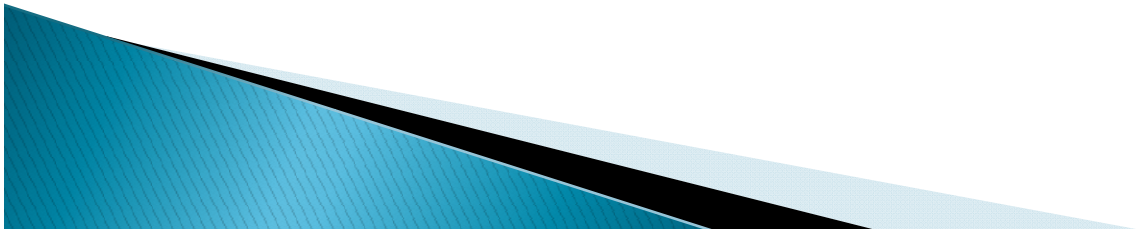
- JSX
- Components
- Props
- State
- Handling Events



JSX

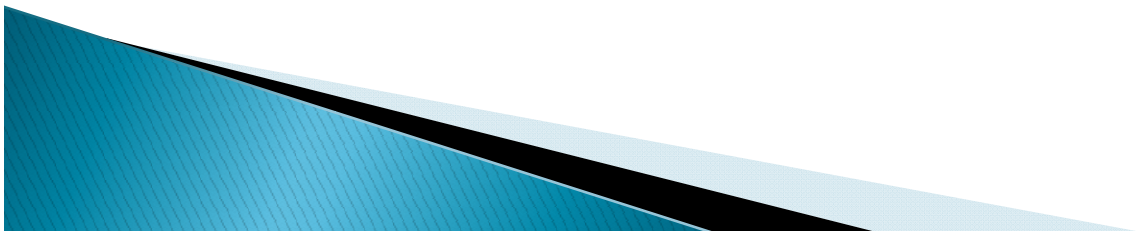
- https://reactjs.org/docs/introducing-jsx.html

- https://reactjs.org/docs/jsx-in-depth.html

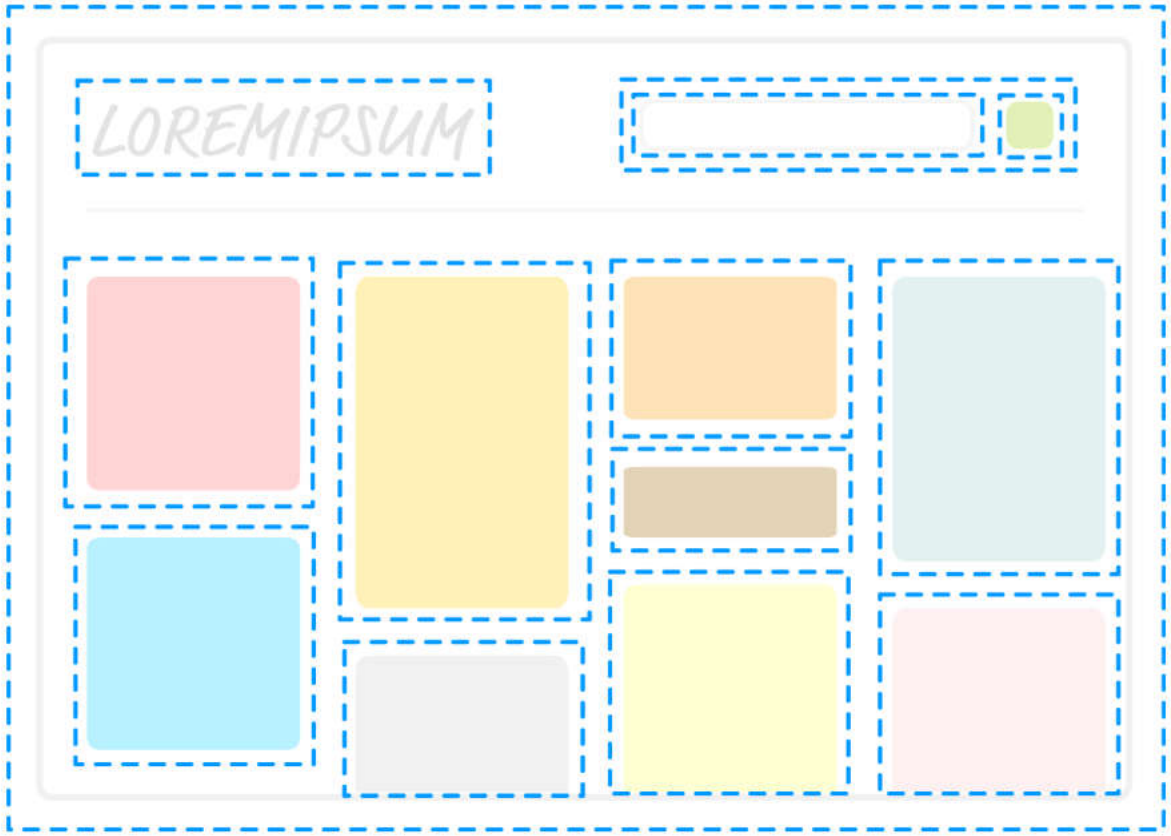


Components

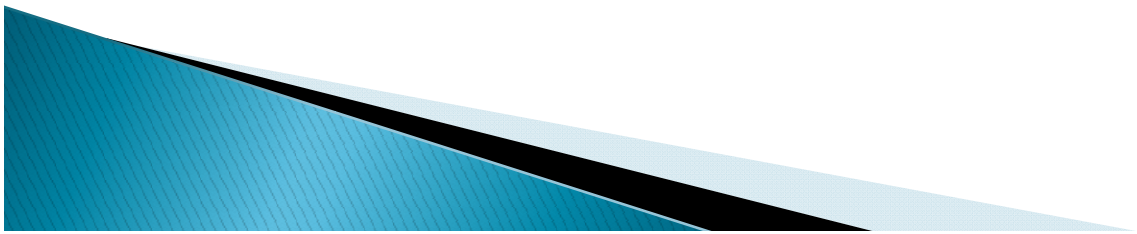
- **Data**: This is data that comes from somewhere (the component doesn't care where), and is rendered by the component.
- **Lifecycle**: These are methods that we implement that respond to changes in the lifecycle of the component. For example, the component is about to be rendered.
- **Events**: This is code that we write for responding to user interactions.
- **JSX**: This is the syntax of React components used to describe UI structures.



Components

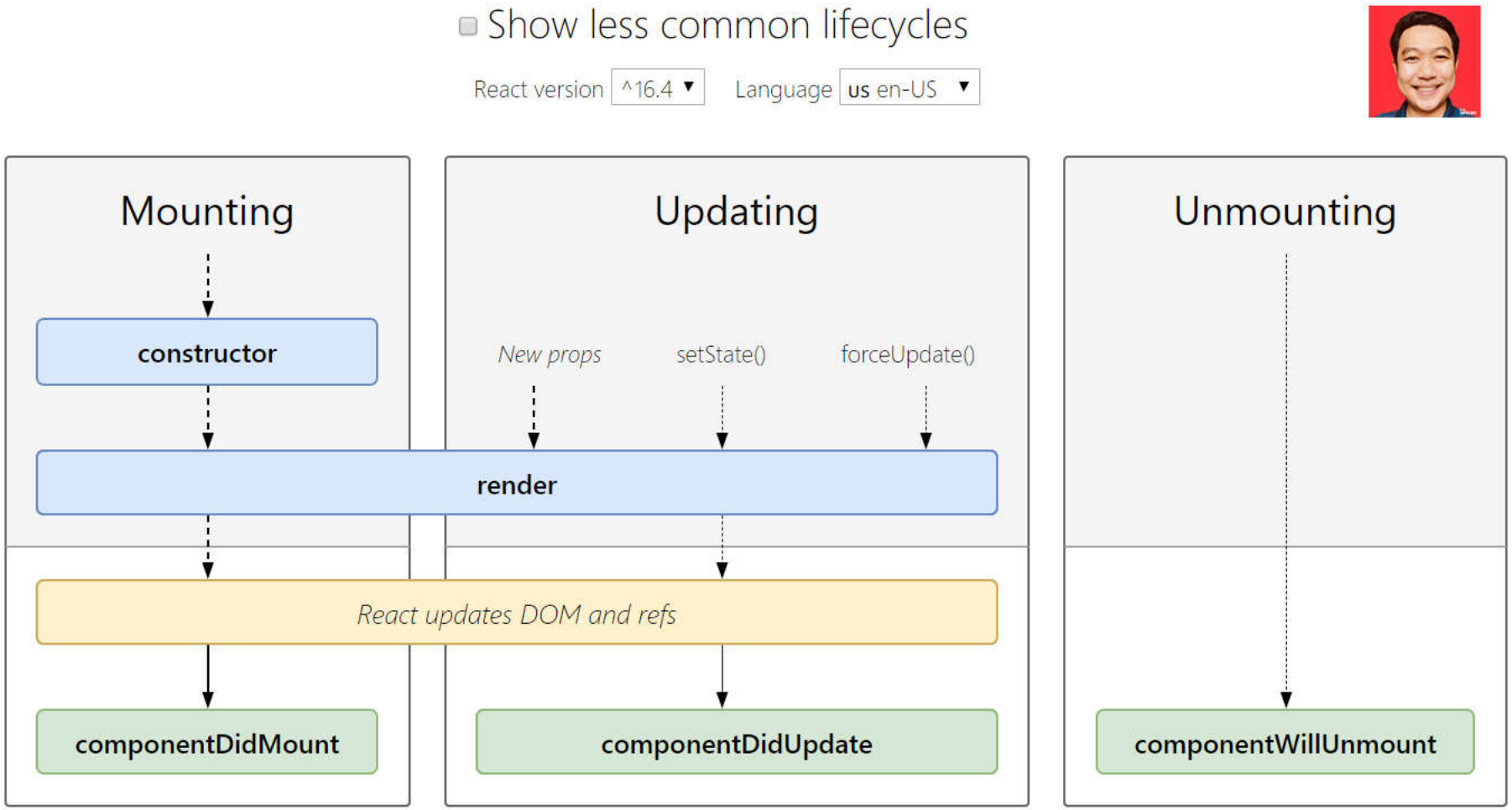


React Lifecycle Methods



The Component Lifecycle

http://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/



The componentDidMount lifecycle method

- **componentDidMount** is called **exactly once, just after the component has been loaded.** This method is a good place **to fetch data with ajax calls**

```
class MainComponent extends Component {
  constructor() {
    super()
    this.state = { loading: true, data: {} }
  }
  componentDidMount() {
    // simulate ajax call
    setTimeout(() => {
      this.setState({
        loading: false,
        data: {name: 'Nader Dabit', age: 35}
      })
    }, 2000)
  }
  render() {
    if(this.state.loading) {
      return <Text>Loading</Text>
    }
    const { name, age } = this.state.data
    return (
      <View>
        <Text>Name: {name}</Text>
        <Text>Age: {age}</Text>
      </View>
    )
  }
}
```

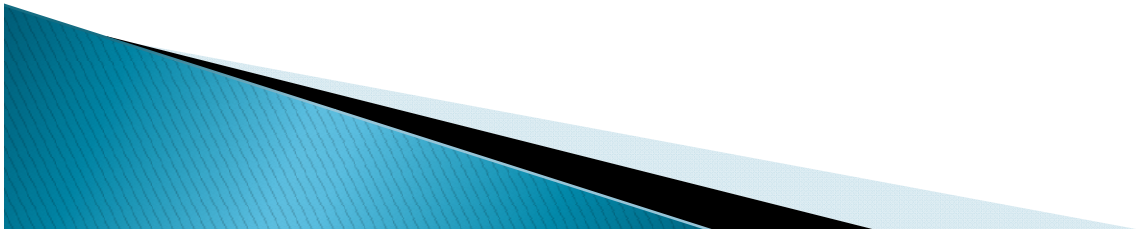
The componentWillUnmount lifecycle method

- **componentWillUnmount is called before the component is removed** from the application. Here, you can perform any necessary **cleanup, remove listeners, or remove timers** that were set up in componentDidMount.

```
class MainComponent extends Component {

   handleClick() {
      this._timeout = setTimeout(() => {
        this.openWidget();
      }, 2000);
    }
    componentWillUnmount() {
      clearTimeout(this._timeout);
    }
    render() {
      return <SomeComponent
                handleClick={() => this.handleClick()} />
    }
}
```

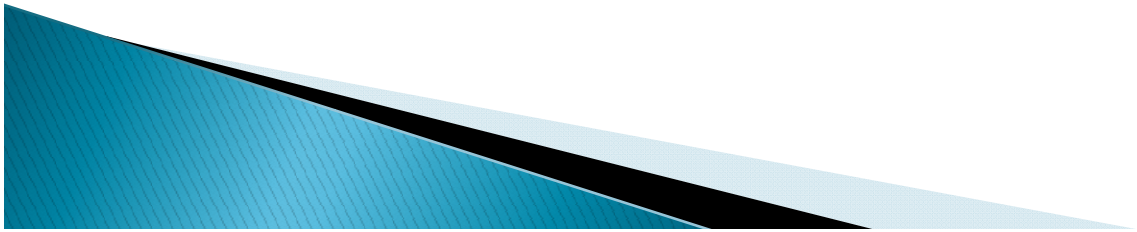
ทำความรู้จักกับ React Native

- **React Native** is a framework for building native mobile apps in JavaScript using the React JavaScript library and compiles to real native components.



ใครบ้างที่ใช้ React Native

- Facebook Ads Manager
- Airbnb
- Bloomberg
- Tesla
- Instagram
- Soundcloud
- Uber
- Walmart
- Amazon and Microsoft



Thinking in components

A Basic React Native Class

- There are two main types of React Native components, stateful and stateless.

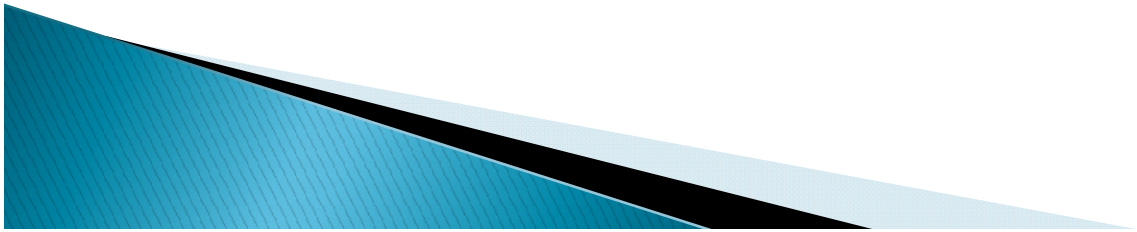
- Stateful component using ES6 class

```
class HelloWorld extends React.Component {
  constructor() {
    super();
    this.state = { name: 'Chris' }
  }

  render () {
    return (
      <SomeComponent />
    )
  }
}
```

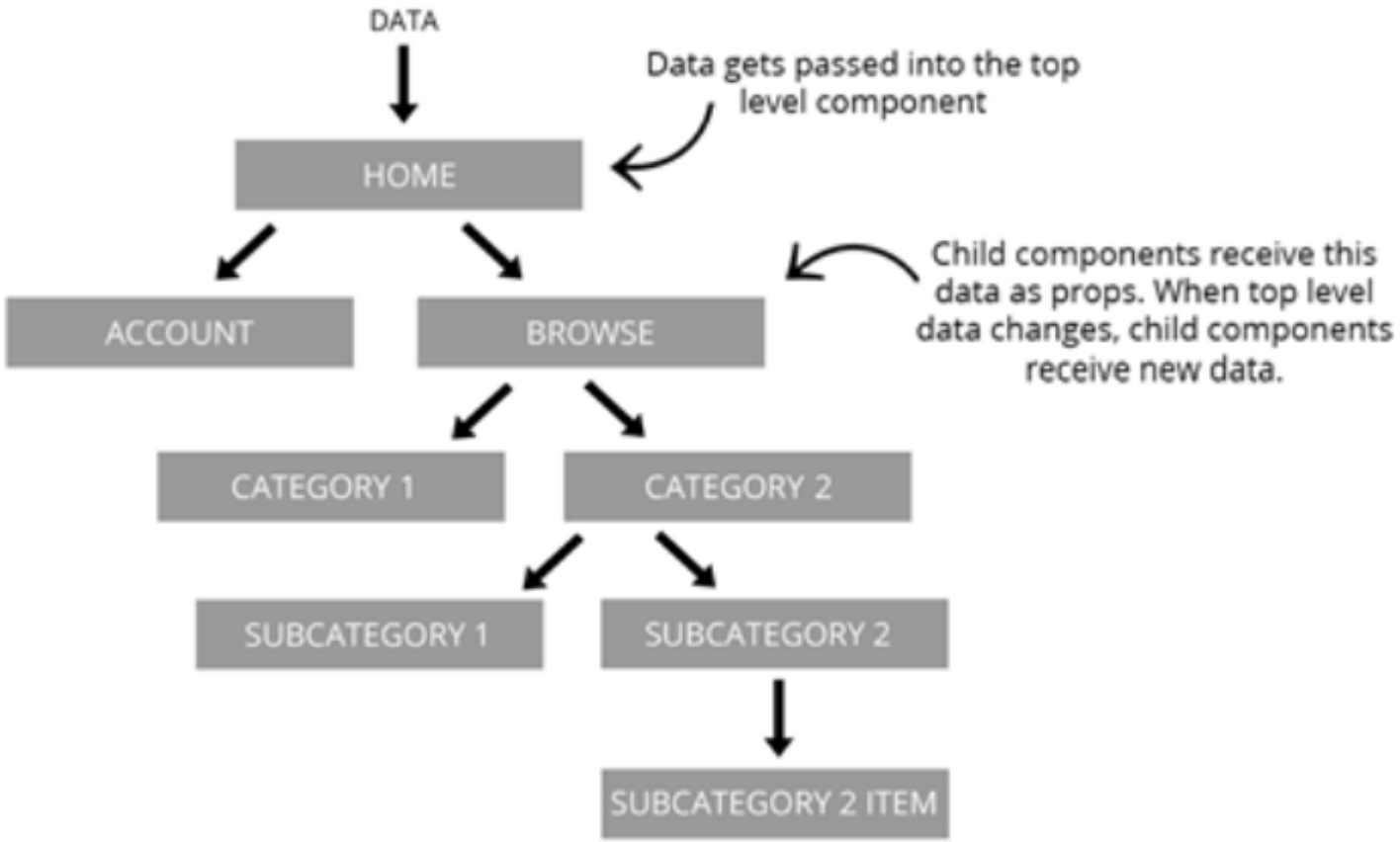
A Basic React Native Class

- Stateless component

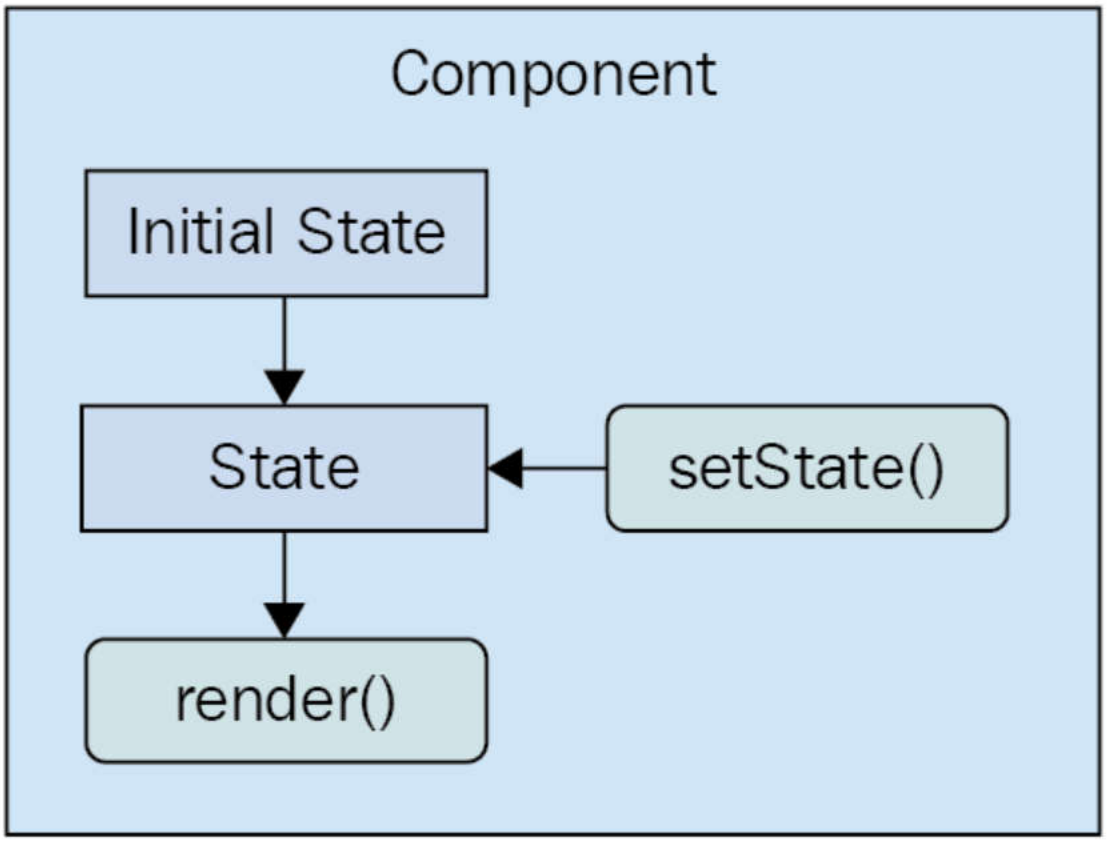
```
const HelloWorld = () => (
  <SomeComponent />
)
```



How one-way data flow works



Component State

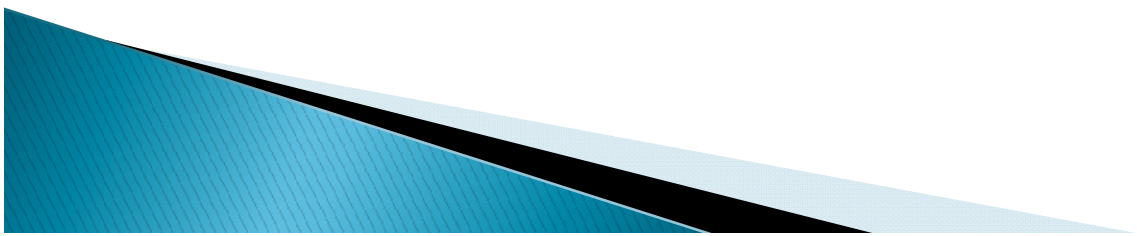


Managing component data using state

- state is declared when the component is created, and **its structure is a plain JavaScript object.**

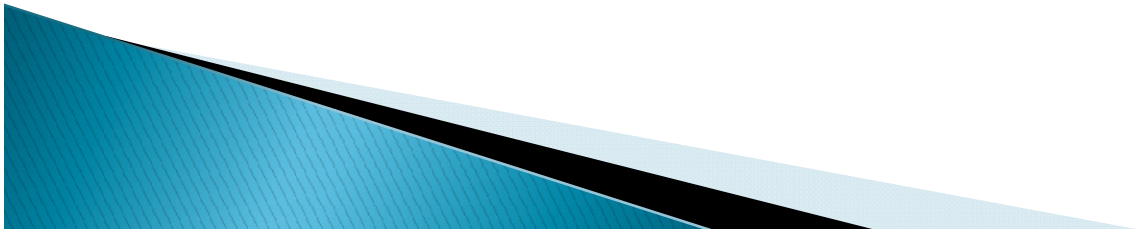
- State can be updated within the component using a function called **setState**

- When the state of a component changes using the setState function, **React rerenders the component.** If any child components are inheriting this state as props, then **all of the child components get rerendered as well.**



Setting *Initial State*

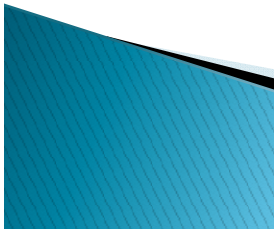
- State is initialized when the component is created in either the constructor or with a property initializer. Once the state is initialized, it is then available in the component **as this.state.**



Setting *Initial State*

- Setting state with a property initializer

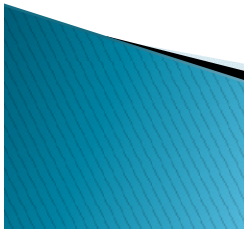
```
import React from 'react'

class MyComponent extends React.Component

  state = {
    year: 2016,
    name: 'Nader Dabit',
    colors: ['blue']
  }

  render() {
    return (
      <View>
        <Text>My name is: { this.state.name }</Text>
        <Text>The year is: { this.state.year }</Text>
        <Text>My colors are { this.state.colors[0] }</Text>
      </View>
    )
  }
}
```



Setting *Initial State*

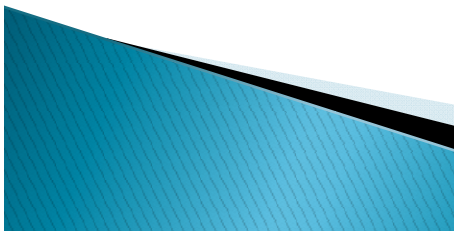
- Setting state with a constructor (class syntax)

```
class MyComponent extends Component {
  constructor(){
    super()
    this.state = {
      year: 2016,
      name: 'Nader Dabit',
      colors: ['blue']
    }
  }
  render() {
    return (
      <View>
        <Text>My name is: { this.state.name }</Text>
        <Text>The year is: { this.state.year }</Text>
        <Text>My colors are { this.state.colors[0] }</Text>
      </View>
    )
  }
}
```



Updating state

```
class MyComponent extends Component {
    constructor(){
      super()
      this.state = {
        year: 2016,
      }
    }
    updateYear() {
      this.setState({
        year: 2017
      })
    }
    render() {
      return (
        <View>
          <Text
            onPress={() => this.updateYear()}>
            The year is: { this.state.year }
          </Text>
        </View>
      )
    }
}
```

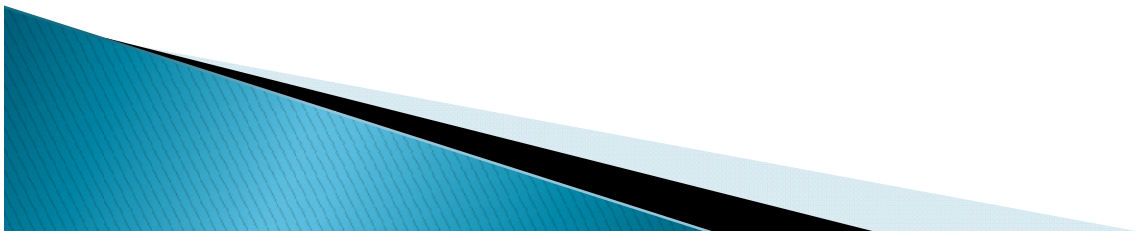


The flow of setState

```
state = {
  year: 2016
}
```

```
this.setState({
  year: 2017
})
```

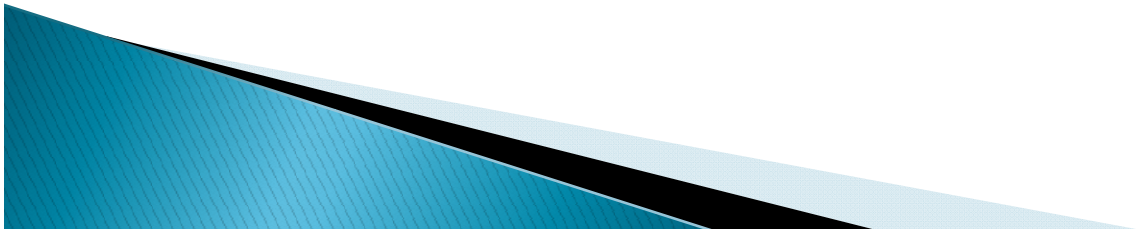
```
state = {
  year: 2017
}
```



The flow of setState

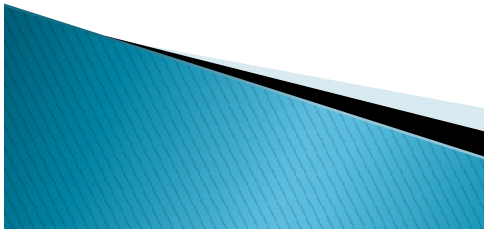
- **Every time setState is called, React will rerender the component (calling the render method again),** and any child components.

- Calling this.setState is the way to change a state variable and trigger the render method again, **as changing the state variable directly will not trigger a rerender of the component** and therefore no changes will be visible in the UI.

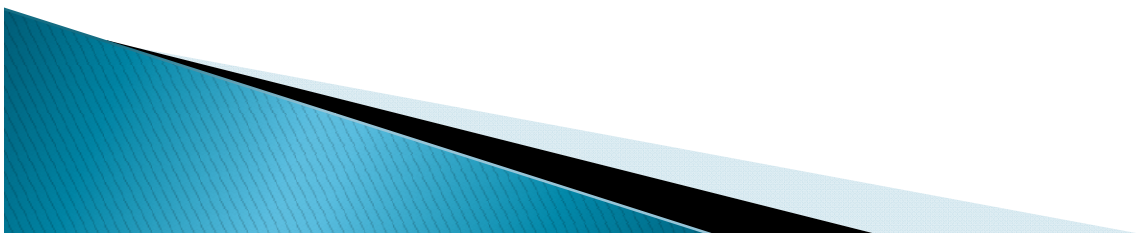


State with other data types

```
class MyComponent extends Component {
    constructor(){
      super()
      this.state = {
        year: 2016,
        leapYear: true,
        topics: ['React', 'React Native', 'JavaScript'],
        info: {
          paperback: true,
          length: '335 pages',
          type: 'programming'
        }
      }
    }
    render() {
      let leapyear = <Text>This is not a leapyear!</Text>
      if (this.state.leapYear) {
        leapyear = <Text>This is a leapyear!</Text>
      }
      return (
        <View>
          <Text>{ this.state.year }</Text>
          <Text>Length: { this.state.info.length }</Text>
          <Text>Type: { this.state.info.type }</Text>
          { leapyear }
        </View>
      )
    }
}
```



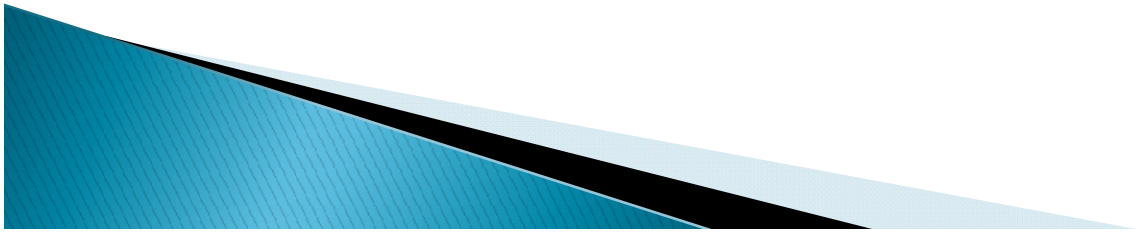
Managing component data using props



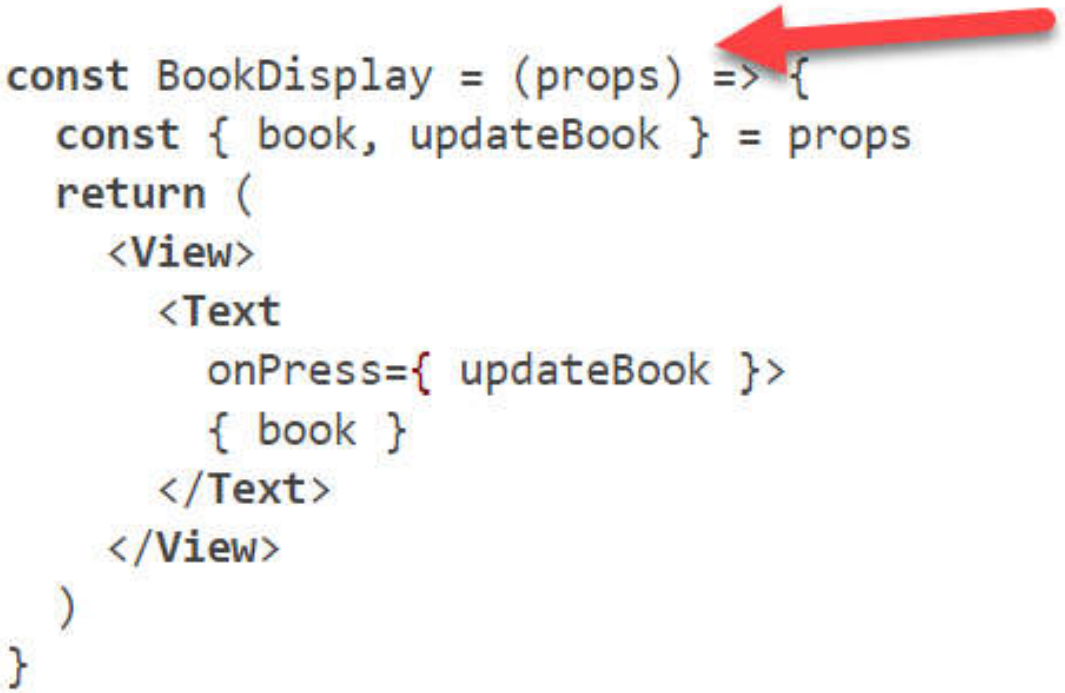
Managing component data using props

- **props** (short for properties) are **a component's inherited values or properties** that have been **passed down from a parent component.**

- "a way of passing data from parent to child."



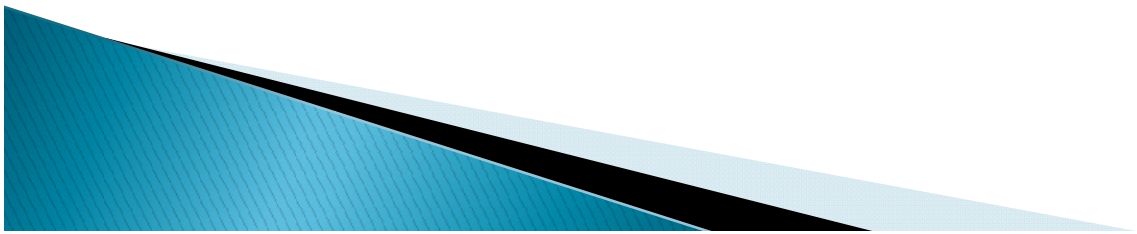
Props with stateless components



```
const BookDisplay = (props) => {
  const { book, updateBook } = props
  return (
    <View>
      <Text
        onPress={ updateBook }>
        { book }
      </Text>
    </View>
  )
}
```

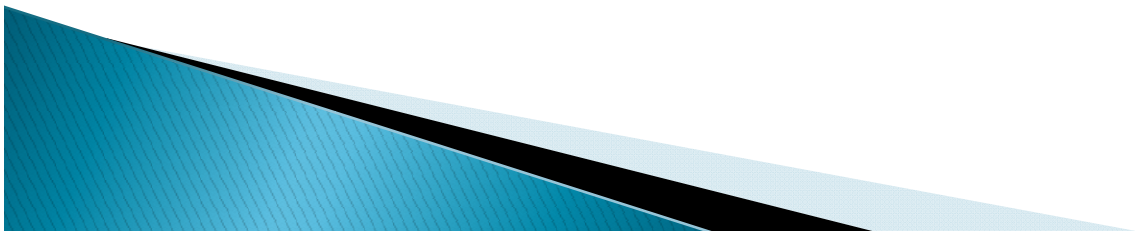
Destructuring props in a stateless component

```
const BookDisplay = ({ updateBook, book }) => {
  return (
    <View>
      <Text
        onPress={ updateBook }>
        { book }
      </Text>
    </View>
  )
}
```



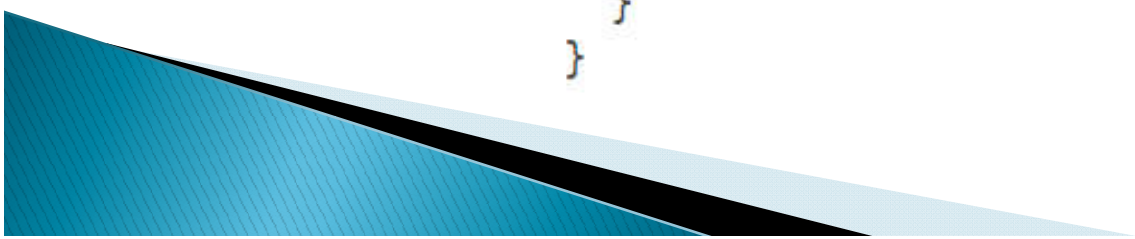
Static props

```
class MyComponent extends Component {
  render() {
    return (
      <BookDisplay book="React Native in Action" />
    )
  }
}
class BookDisplay extends Component {
  render() {
    return (
      <View>
        <Text>{ this.props.book }</Text>
      </View>
    )
  }
}
```

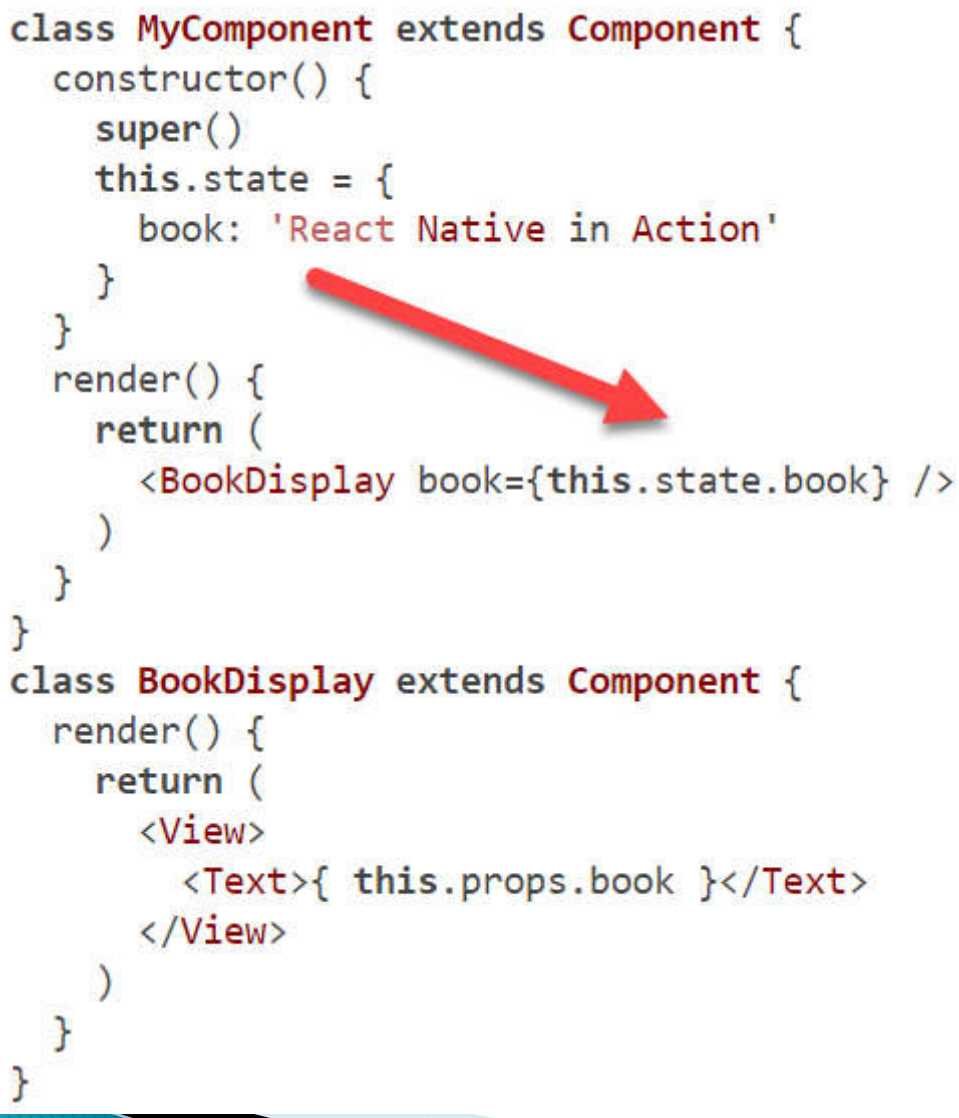


Dynamic props

```
class MyComponent extends Component {
    render() {
      let book = 'React Native in Action'
      return (
        <BookDisplay book={ book } />
      )
    }
  }

  class BookDisplay extends Component {
    render() {
      return (
        <View>
          <Text>{ this.props.book }</Text>
        </View>
      )
    }
  }
```



Dynamic props using state



```
class MyComponent extends Component {
  constructor() {
    super()
    this.state = {
      book: 'React Native in Action'
    }
  }
  render() {
    return (
      <BookDisplay book={this.state.book} />
    )
  }
}
class BookDisplay extends Component {
  render() {
    return (
      <View>
        <Text>{ this.props.book }</Text>
      </View>
    )
  }
}
```

Updating dynamic props

```
class MyComponent extends Component {
    constructor(){
      super()
      this.state = {
        book: 'React Native in Action'
      }
    }
    updateBook() {
      this.setState({
         book: 'Express in Action'
      })
    }
    render() {
      return (
        <BookDisplay
         updateBook={ () => this.updateBook() }
         book={ this.state.book } />
      )
    }
  }
```

```
class BookDisplay extends Component {
  render() {
    return (
      <View>
        <Text
         onPress={ this.props.updateBook }>
          { this.props.book }
        </Text>
      </View>
    )
  }
}
```

PASSING ARRAYS AND OBJECTS AS PROPS

```
class Header extends Component {
    constructor(){
      super()
      this.state = {
        leapYear: true,
        info: {
          type: 'programming'
        }
      }
    }
    render() {
      return (
        <BookDisplay
          leapYear={ this.state.leapYear }
          info={ this.state.info }
          topics={['React', 'React Native', 'JavaScript']}
        />
      )
    }
}
```

```
const BookDisplay = (props) => {
  let leapyear
  let { topics } = props
  const { info } = props
  topics = topics.map((topic, i) => {
    return <Text>{ topic }</Text>
  })
  if (props.leapYear) {
    leapyear = <Text>This is a leapyear!</Text>
  }
  return (
    <View>
        { leapyear }
        <Text>Book type: { info.type }</Text>
        { topics }
    </View>
  )
}
```

Props vs. State

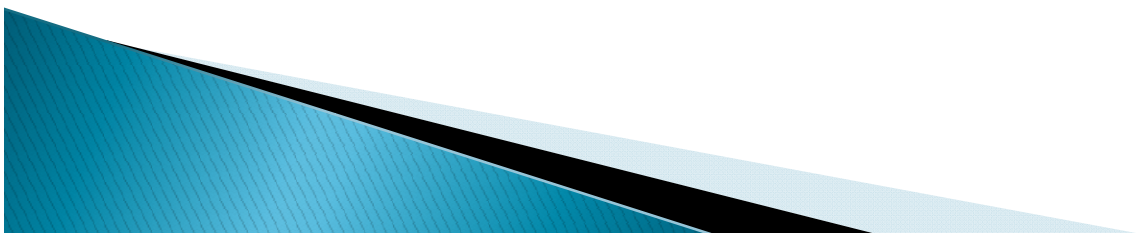
Props	State
external data	internal data
immutable	mutable
inherited from parent	created in the component
can be changed by parent component	can only be updated in the component
can be passed down as props	can be passed down as props
cannot change inside component	can change inside component

Using the render method to create your UI

- Render method

```
render() {
  return (
    <View>
      <Text>Hello</Text>
    </View>
  )
}
```

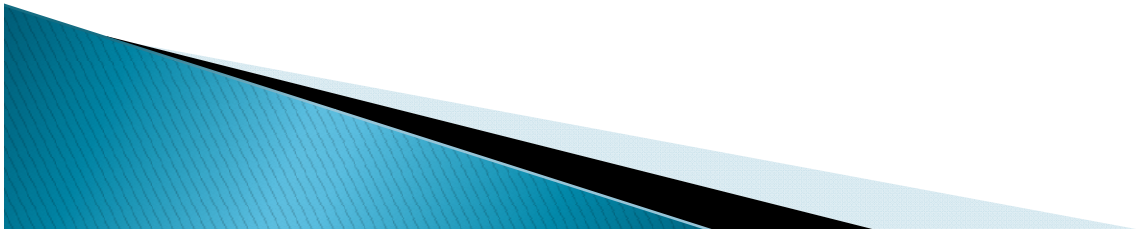
```
render() {
   if(something === true) {
     return <SomeComponent />
   } else return <SomeOtherComponent />
 }
```



React Native Components

- Components are a collection of data and UI elements that make up your views and ultimately your application.

- 1. Text component
- 2. View component
- 3. Touchable highlight



JSX components vs HTML elements

1. Text component

HTML

```
<span>Hello World</span>
```

React Native JSX

```
<Text>Hello World</Text>
```

2. View component

HTML

```
<div>
  <span>Hello World 2</span>
</div>
```

React Native JSX

```
<View>
  <Text>Hello World 2</Text>
</View>
```

3. Touchable highlight

HTML

```
<button>
  <span>Hello World 2</span>
</button >
```

React Native JSX

```
<TouchableHighlight>
  <Text>Hello World 2</Text>
</TouchableHighlight>
```

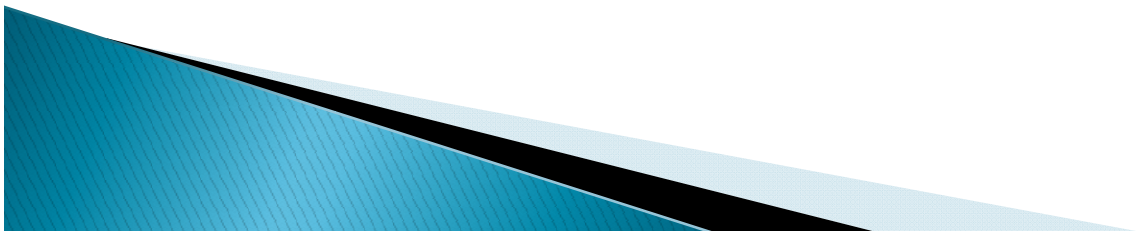
Native components

- The framework offers native components out of the box, such as View, Text, and Image, among others.

```
const Button = () => (
  <TouchableHighlight>
    <Text>Hello World</Text>
  </TouchableHighlight>
)
export default Button
```

```
Import React from 'react'
Import { Text, View } from 'react-native'
import Button from './pathtobutton'
const Home = () => (
  <View>
    <Text>Welcome to the Hello World Button!</Text>
    <Button />
  </View>
)
```

Introduction to styling

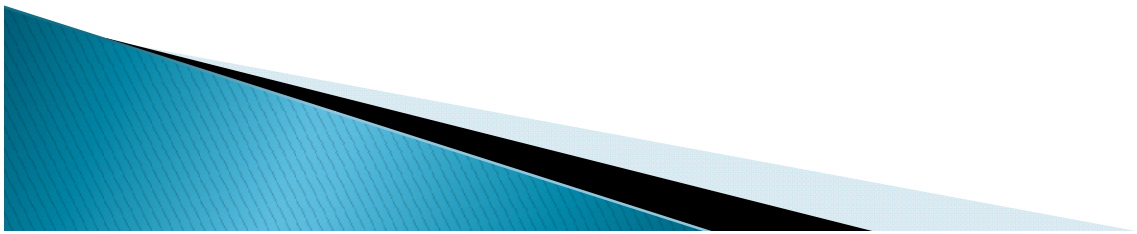


Styling overview

- Each component supports a specific set of styles.

- Those styles **may or may not be applicable to other types** of components.

- For example, the Textcomponent supports the fontWeight property (fontWeight refers to the thickness of the font), **but the View component does not.** Conversely, **the View component supports the flex property** (flex refers to the layout of components within a View), **but the Text component does not.**



Applying styles

- Using inline styles

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

class App extends Component {
  render () {
    return (
      <View style={{marginLeft: 20}}>     // A
        <Text style={{fontSize: 18,color: 'red'}}>Some Text</Text> // B
      </View>
    )
  }
}
A An inline style applied to a React Native component.
B Multiple inline styles applied at once
```

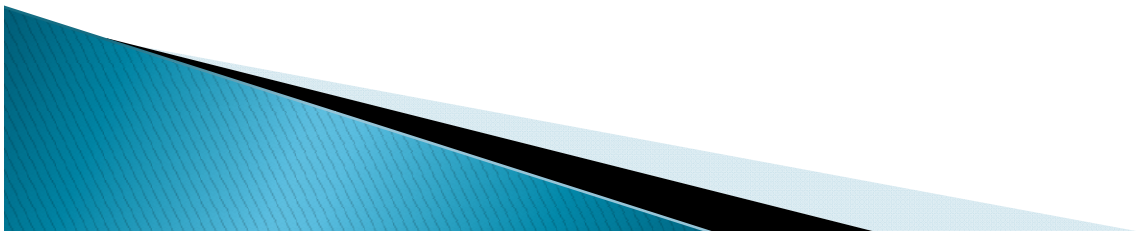
Referencing styles defined within a StyleSheet

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

class App extends Component {
  render () {
    return (
      <View style={styles.container}>     // A
        <Text style={[styles.message,styles.warning]}>Some Text</Text> //
      </View>
    )
  }
}
const styles = StyleSheet.create({
  container: {          // C
    marginLeft: 20
  },
  message: {            // C
    fontSize: 18
  },
  warning: {
    color: 'red'        // C
  }
})
A Referencing the container style defined within the styles stylesheet.
B Using an array to reference both the message and warning styles from th
C Defining the styles using StyleSheet.create
```

b

Organizing styles

- declaring stylesheets within the same file as the component.
- declaring stylesheets in a separate file, outside of the component.



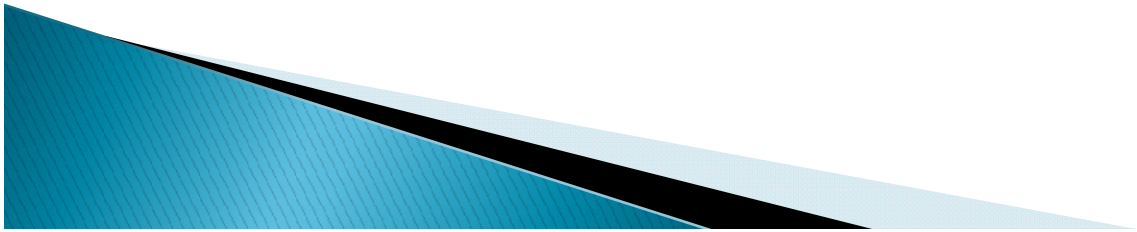
Externalizing a component's stylesheets (styles.js)

```
import { StyleSheet } from 'react-native'

const styles = StyleSheet.create({   // A
  container: {    // B
    marginTop: 150,
    backgroundColor: '#ededed',
    flexWrap: 'wrap'
  }
})

const buttons = StyleSheet.create({  // C
  primary: {     // D
    flex: 1,
    height: 70,
    backgroundColor: 'red',
    justifyContent: 'center',
    alignItems: 'center',
    marginLeft: 20,
    marginRight: 20
  }
})

export { styles, buttons }    // E
```

Externalizing a component's stylesheets (styles.js)

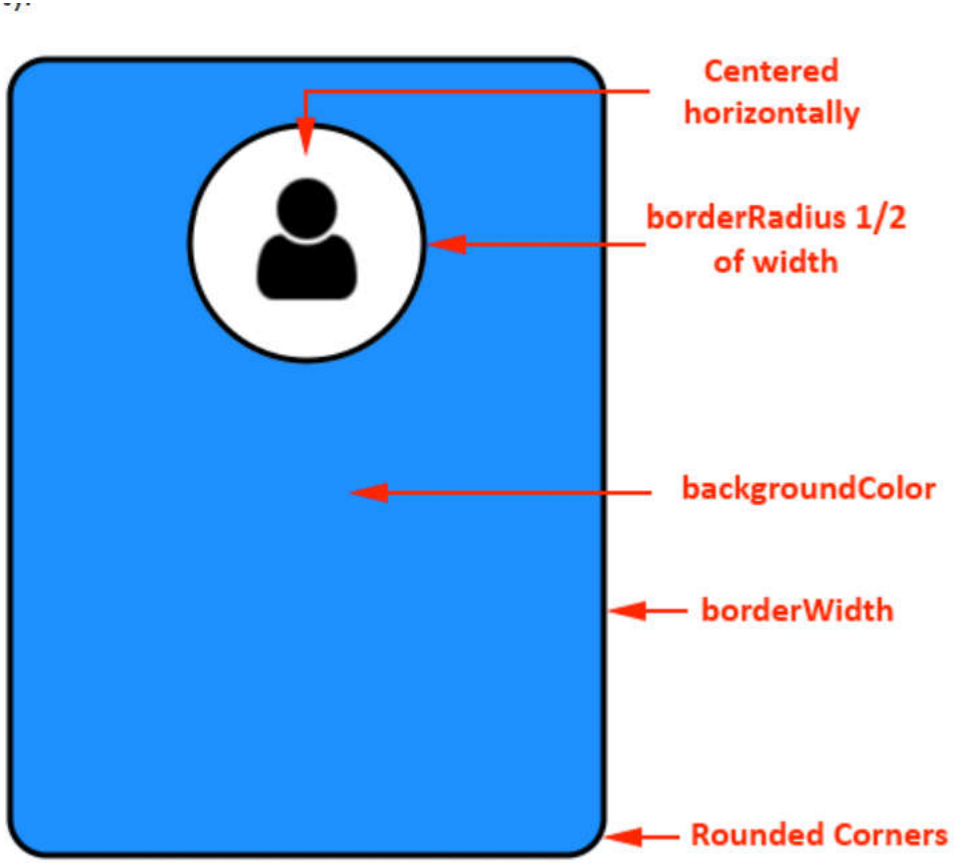
- Importing external stylesheets

```
import { styles, buttons } from './component/styles'    // A

<View style={styles.container}>    // B
  <TouchableHighlight style={buttons.primary} />    // C
    ...
  </TouchableHighlight>
</View>
A Importing multiple stylesheets exported from styles.js
B A reference to the styles.container style created in styles.js
C A reference to the buttons.primary style created in styles.js
```



Styling view components

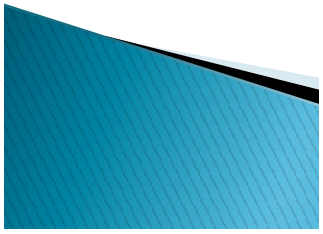
- The View is the main building block of your UI and is one of the most important components to understand if you want to get your styling right.



Styling view components: backgroundColor

- The backgroundColor property sets the background color of an element.

Supported Color Format	Example
#rgb	'#06f'
#rgba	'#06fc'
#rrggbb	'#0066ff'
#rrggbbaa	'#ff00ff00'
rgb(number, number, number)	'rgb(0, 102, 255)'
rgb(number, number, number, alpha)	'rgba(0, 102, 255, .5)'
hsl(hue, saturation, lightness)	'hsl(216, 100%, 50%)'
hsla(hue, saturation, lightness, alpha)	'hsla(216, 100%, 50%, .5)'
transparent background	'transparent'
any CSS3 specified named color (black, red, blue, etc...)	'dodgerblue'

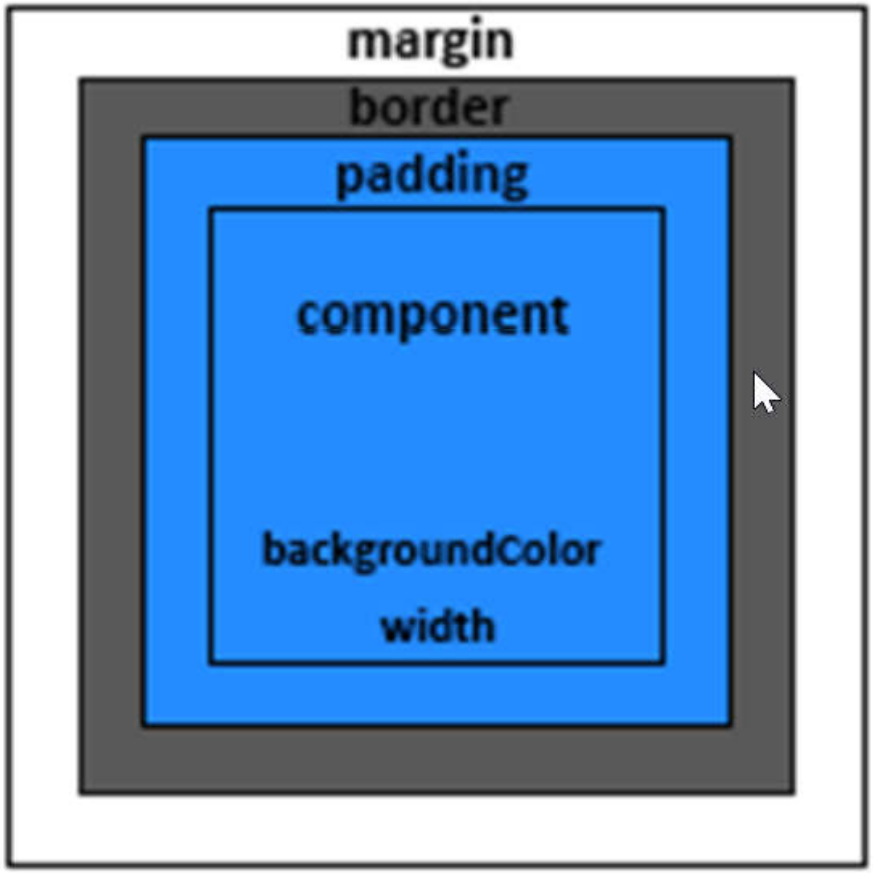


Styling view components: Border properties

- borderColor
- borderRadius
- borderStyle
- borderWidth

- borderTopColor
- borderRightColor
- borderBottomColor
- borderLeftColor
- borderTopWidth
- borderRightWidth
- borderBottomWidth
- borderLeftWidth
- borderTopRightRadius
- borderBottomRightRadius
- borderBottomLeftRadius
- borderTopLeftRadius

Styling view components: Margin and Padding



Styling text components

- fontFamily
- fontSize
- fontStyle
- fontWeight

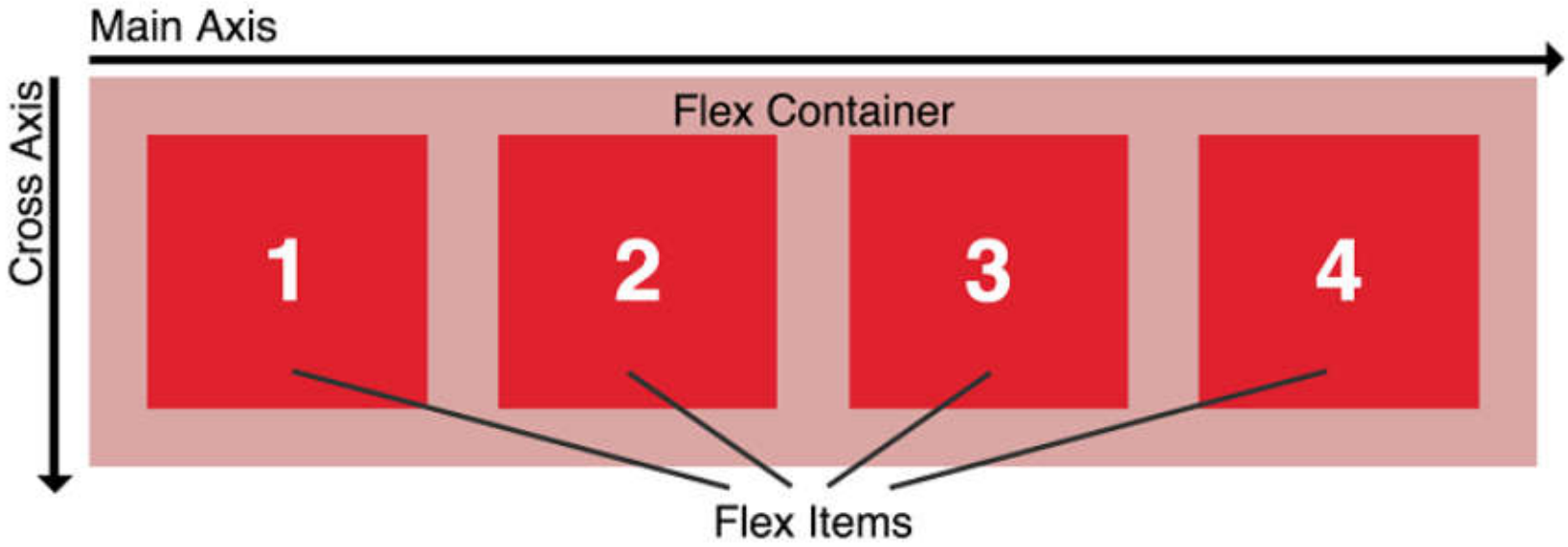
- lineHeight
- textAlign
- textDecorationLine
- textShadowColor
- textShadowOffset
- textShadowRadius

```
cardName: {
    color: 'white',
    fontWeight: 'bold',
    fontSize: 24,
    marginTop: 30,
    textShadowColor: 'black',   // A
    textShadowOffset: {         // B
        height: 2,
        width: 2
    },
    textShadowRadius: 3         // C
},
```

Flexbox principles (for web)

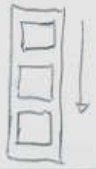
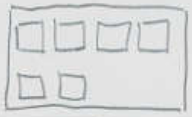
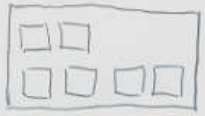
- **Flexbox** begins with the familiar display property.
- Applying **display: flex** to an element turns it into **a *flex container***, and its direct children turn into ***flex items*.**
- By default, flex items will align side by side**, left to right, all in one row.**
- **The flex container will fill available width like a block element**, but the flex items **will not necessarily fill the width** of their flex container. The flex items will all be equal heights, determined naturally by their contents.

A flexbox container and its elements



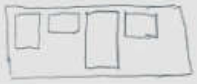
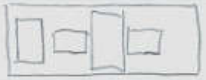
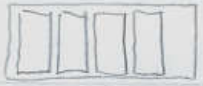
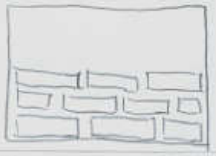
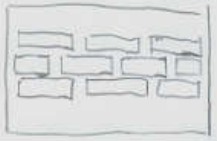
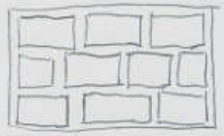
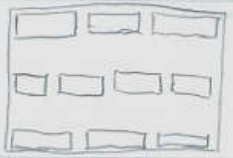
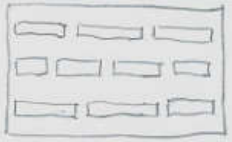
flexbox

FLEX CONTAINER PROPERTIES

PROPERTY	DESCRIPTION	VALUES (initial values in bold)
flex-direction	Specifies direction of the main axis. The cross axis will be perpendicular to the main axis	**row** 
		row-reverse 
		column 
		column-reverse 
flex-wrap	Specifies whether flex items may wrap onto a new row inside the flex container (or a new column, if flex-direction is column or column-reverse)	**nowrap** 
		wrap 
		wrap-reverse 
flex-flow	Shorthand property for flex-direction and flex-wrap	<flex-direction> || <flex-wrap>
justify-content	Controls how items are positioned along the main axis.	**flex-start** 
		flex-end 
		center 
		space-between 
		space-around 

flexbox

FLEX CONTAINER PROPERTIES (CONTINUED)

PROPERTY	DESCRIPTION	VALUES (initial value in bold)	
align-items	Controls how items are positioned along the cross axis.	flex-start	
		flex-end	
		center	
		stretch	
		baseline	
align-content	If flex-wrap is enabled, this controls the alignment of the flex rows along the cross axis. Ignored if items do not wrap.	flex-start	
		flex-end	
		center	
		stretch	
		space-between	
		space-around	

Flexbox for React Native

- Flexbox is a layout implementation that React Native uses to provide an efficient way for users to create UIs and control positioning in React Native.

- Flexbox layout is only available for use on **View components.**

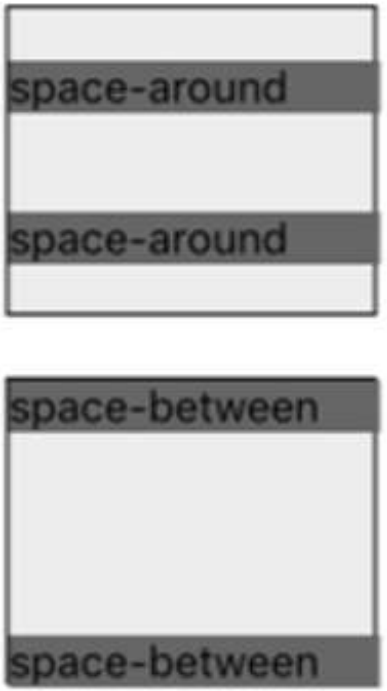
Flexbox

- 1) A = {flex: 1} and B = {flex: 1} resulting in each taking up 50% of the space
- 2) C = {flex: 1} and D = {flex: 2} resulting in C taking up 33% of the space and D taking up 66%
- 3) E = {flex: 1} and F = {flex: 3}

Flexbox: flexDirection

Flexbox: justifyContent

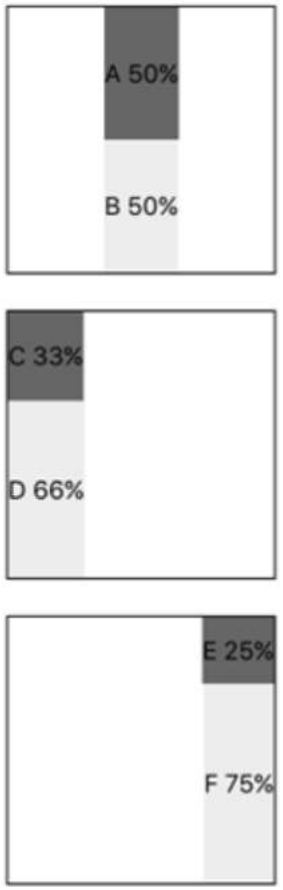
- center
- flex-start
- flex-end
- space-around
- space-between



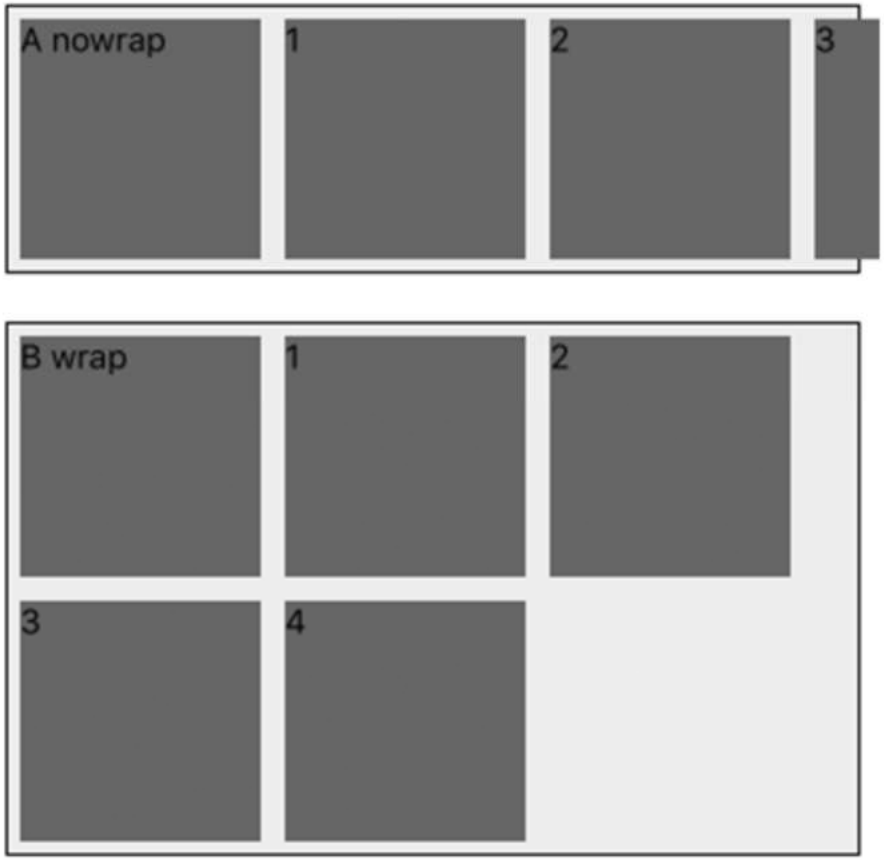
Flexbox: alignItems

- alignItems defines how to align children along the secondary axis of its container.

```
stretch
center
flex-start
flex-end
```



Flexbox: flexWrap



The end