Lab 02 – React Basics

By Eng. Rania El-Sayed
Mobile Computing Lab [HU]
4th Computer - Spring 2025

Contents

React Native Basic Components

- View
- Text
- TextInput
- Button
- Image

React Native Styling

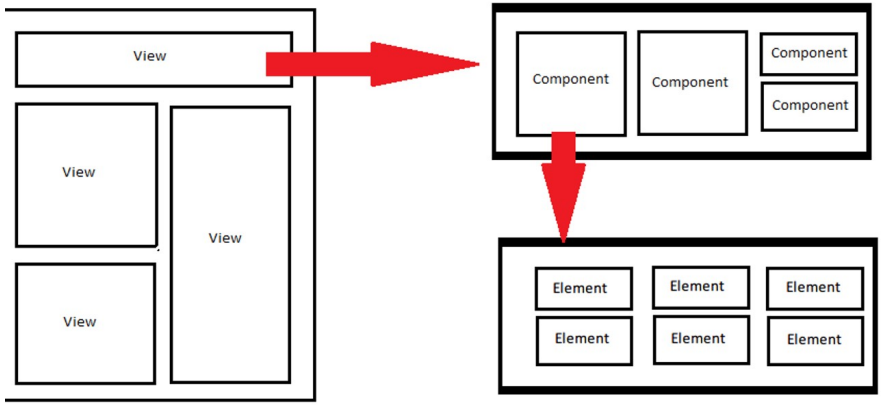
- Width and Height
- Layout with FlexBox
- Color Reference

React Native Components

React Components

SafeAreaView
StatusBar
Button
Switch
TouchableHighlight
TouchableOpacity
Modal
RefreshControl
Pressable
VirtualizedList
ActivityIndicator
InputAccessoryView

View
Text
TextInput
ScrollView
FlatList
SectionList
Image
ImageBackground
KeyboardAvoidingView
TouchableWithoutFeedback
DrawerLayoutAndroid
TouchableNativeFeedback



React Components - Example

```
export default function App() {
  return (
    <View style={styles.container}>
      <Text>Functional Component</Text>
      <TextInput placeholder="Enter your name!"/>
      <Button title="Click Me!" />
      <Image source={require("./assets/images.jpg")} />
    </View>
  );
}
```

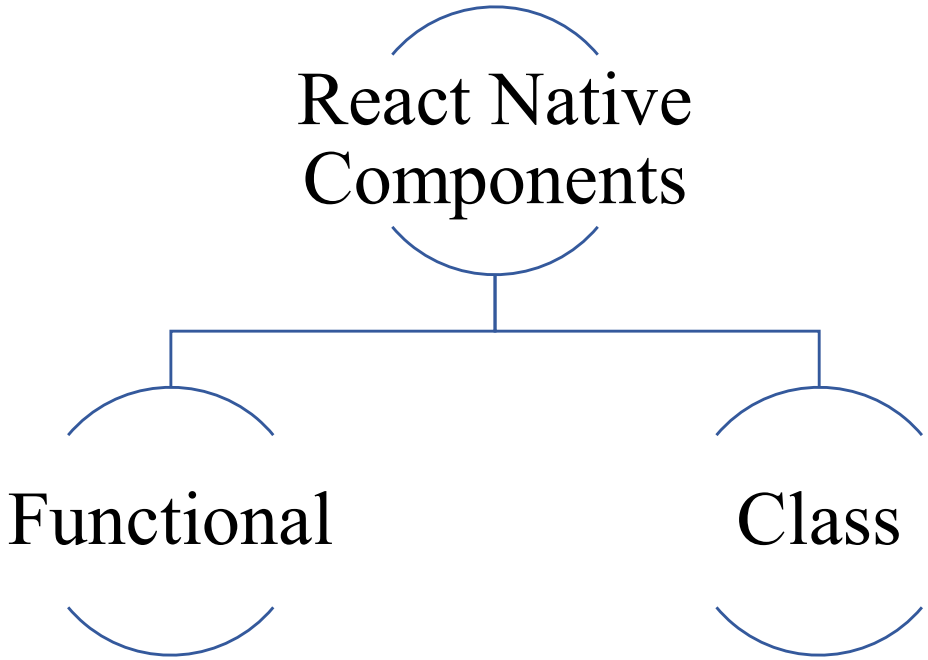
Image Component - require()

- A built-in function to include external modules that exist in separate files.
- It basically reads a JavaScript file, executes it, and then proceeds to return the export object.
- It allows to add built-in core NodeJS modules and community-based and local modules.
- In React Native, you cannot directly use a string URL for local images, so everything in react is called using require function

React Native Components

Components:
Independent and reusable pieces of code. They serve the same purpose as JavaScript functions but work in isolation and return JSX via a render() function.

Props:
Stands for properties and is used for establishing communication between different components through passing data from one component to another like arguments passed to functions.



React Native - Functional Components

- Functional Components are simple JavaScript functions that can be arrow functions or written using function keyword.
- They are ***stateless*** components as they cannot manage their state or use lifecycle methods on their own such as render() method.
- Pure Functional components focus on rendering the UI rather than behavior. They simply accept props and return valid JSX element.

```
export default function App() {
  return (
    <View style={styles.container}>
      <Text>Functional Component</Text>
    </View>
  );
}
```

React Native - Class Components

- Class components are JavaScript ES6 classes which are extended from a base class called React.Component.
- They are ***stateful*** components as they can manage state and have life cycle methods like constructor(), render(), componentDidMount(), etc, as well as state/props functionality from the parent.
- Class components act as a container which can wrap child components into it.

```
import React from "react";
```

```
export default class App extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>Class Component</Text>
      </View>
    );
  }
}
```

render() method returns JSX code that is responsible for UI elements representation.

React Native Components (Cont.)

- The render method returns the JSX (a syntax extension of JavaScript) that describes what the UI should look like. This method is required in every class component.

- Exporting a component (whether it is class based or functional component) is using it in some other Component/ Components by importing them.

- Use props for static data: Use props to pass data from parent components to child components. Props should be used for data that doesn’t change during the component’s lifecycle.

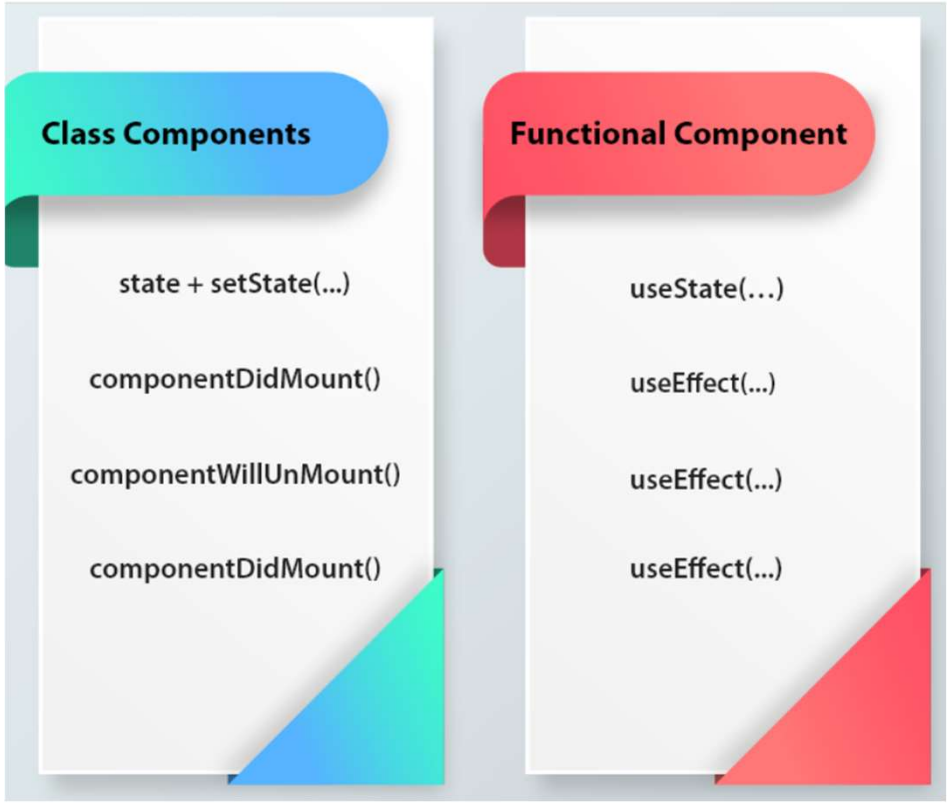
React Native Components (Cont.)

- Use state for dynamic data: Use state to manage data that changes during the component's lifecycle. This could include data that's generated from user interactions or fetched from an API.

- Keep state separate from UI logic: Avoid mixing state management with UI logic in your components. Instead, use a separate state management library like Redux.

```jsx
export default class App extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>Class Component</Text>
      </View>
    );
  }
}
```

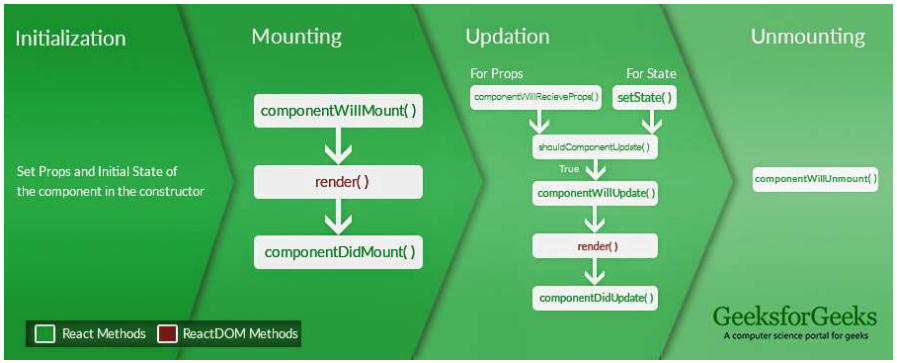
```jsx
export default function App() {
  return (
    <View style={styles.container}>
      <Text>Functional Component</Text>
    </View>
  );
}
```

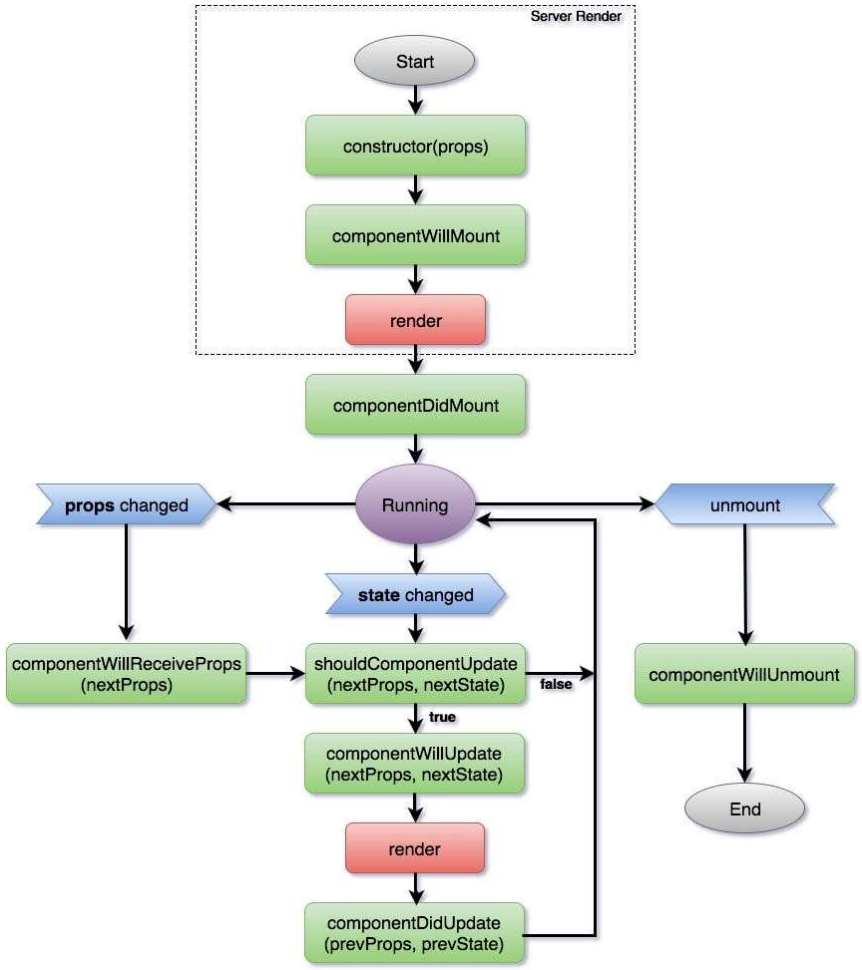


Functional VS Class Components

Aspect	Functional Components	Class Components
Definition	A simple JavaScript pure function that takes props and returns a React element (JSX).	Must extend from `React.Component` and define a `render()` method that returns a React element.
Render Method	No `render` method is used. JSX is returned directly from the function.	A `render()` method is required to return JSX, similar to HTML in syntax.
Execution	Executes from top to bottom. Once the function is returned, it's done with its execution.	Instantiates a class instance with lifecycle methods that can be invoked at different component phases.
State Management	Initially known as stateless, but can now manage state using hooks like `useState`.	Known as stateful components and manage their own state traditionally through `this.state`.
Lifecycle Methods	Cannot use lifecycle methods but can mimic them using hooks like `useEffect`.	Can use lifecycle methods such as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount`.
Hooks Usage	Implements hooks easily within the function body to handle state and effects.	Does not use hooks. State and lifecycle logic are handled differently through class methods.
Constructor	Does not require a constructor. Hooks handle state initialization and effects without it.	Requires a constructor for initializing state and binding event handlers.
Efficiency	More efficient due to less boilerplate and direct usage of hooks for state and effects.	Slightly less efficient. Can have more boilerplate code due to lifecycle methods and state management.
Code Complexity	Requires fewer lines of code, leading to simpler, more readable components.	Typically requires more code, which can lead to more complexity and verbosity.

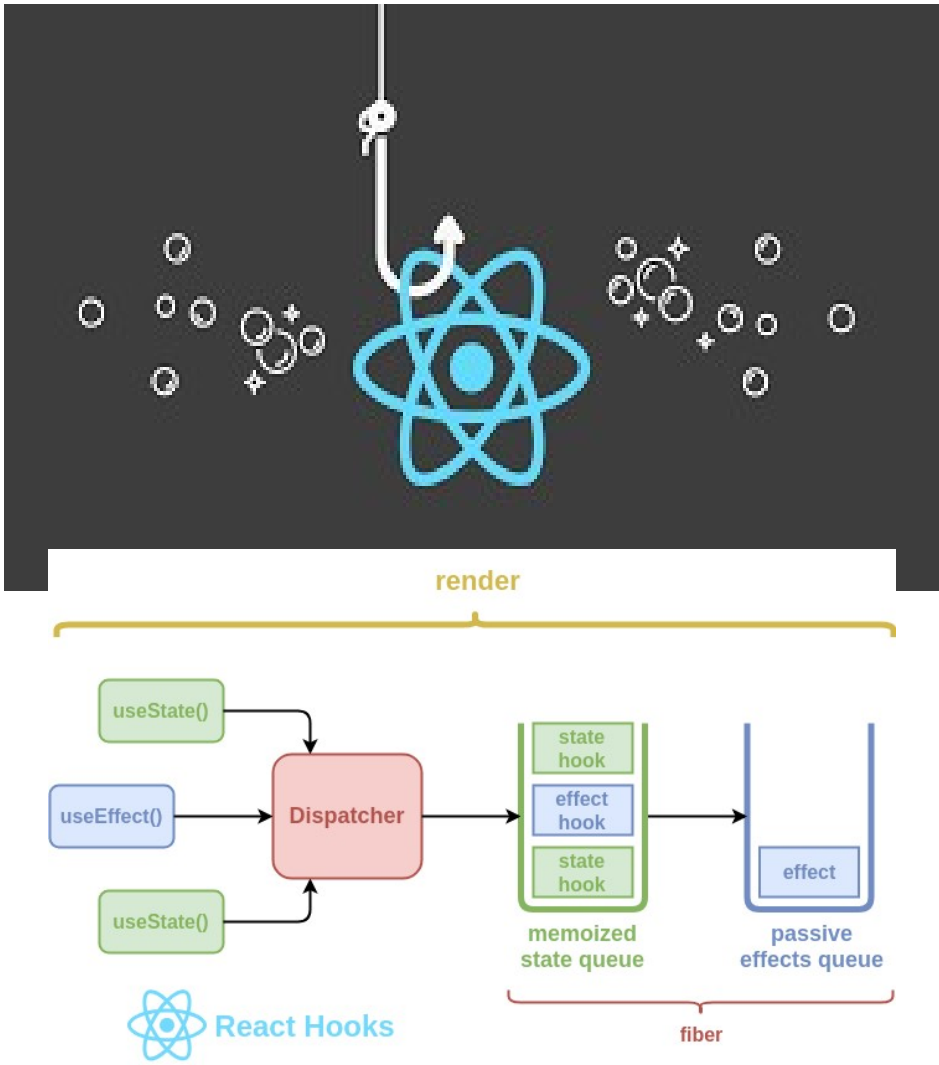
React Native Life Cycle





React Native - Hooks

- Hooks are functions that allow you to hook into React State and lifecycle features from functional component. Hooks don't work with Class component.
- They allow us to use state and React features like lifecycle methods without modifying our existing functional component.
- use state and lifecycle methods inside functional components



Map of Hooks

State Management

- useState
- useReducer
- useSyncExternalStore

Context Hooks

- useContext

Transition Hooks

- useTransition
- useDeferredValue

Ref Hooks

- useRef
- useImperativeHandle

Random Hooks

- useDebugValue
- useId

Performance Hooks

- useMemo
- useCallback

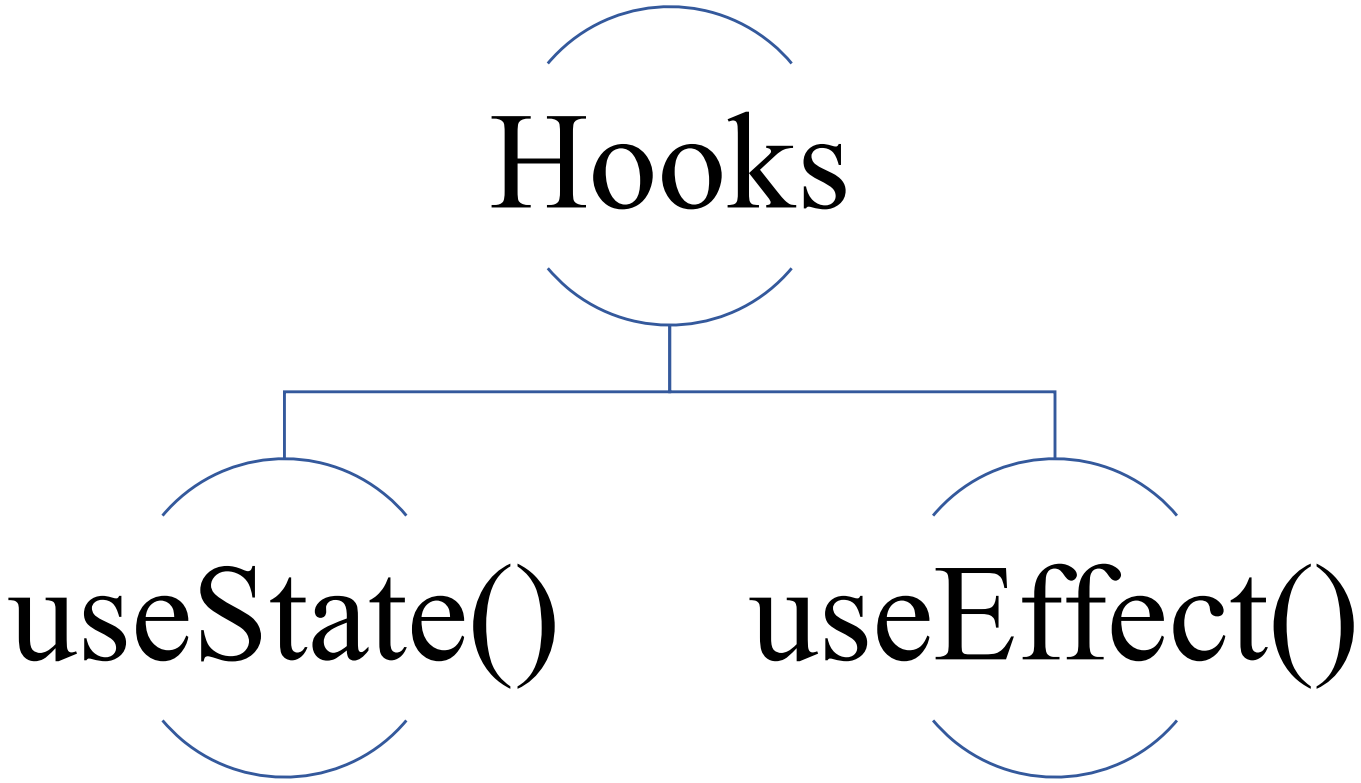
Effect Hooks

- useEffect
- useLayoutEffect
- useInsertionEffect

React 19 Hooks

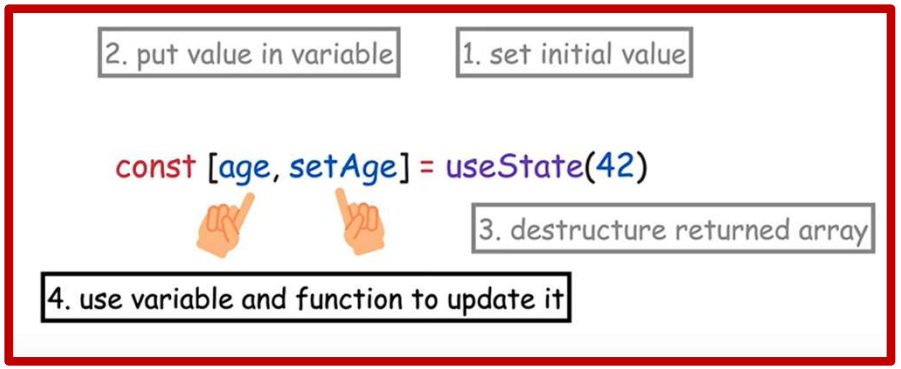
- useFormStatus
- useFormState
- useOptimistic
- use

Hooks - we will focus on



useState() Hook

- A big reason why the react exist is to help us manage state and re-render components when state changes.
- Use state is best for client components that need their own simple specific state.
- It is great for capturing user input and form fields like: inputs, textAreas and selects.
- Can be used to show or hide components like modals, tooltips or dropdowns when you give it a Boolean state value.
- We can also use a Boolean state variable to conditionally apply classes and styles.




```
<button
  className={isActive ? 'active' : 'inactive'}
  // Click to toggle classes
  onClick={() => setIsActive(!isActive)}
>
  Click Me
</button>
```

useEffect() Hook

- A **built-in React Hook** that allows you to perform **side effects** in functional components like fetching data from an API, or setting up event listeners, or managing timers or intervals.
- It replaces lifecycle methods like: **componentDidMount**, **componentDidUpdate**, and **componentWillUnmount** that are used in class components.
- **useEffect() accepts two arguments:**
  - A callback function (where the side effect is performed).
  - A dependency array (which determines when the effect runs).
- Dependencies includes state or props, and all the variables and functions declared directly inside your component body.

```
import { useEffect } from "react";

useEffect(() => {
  // Side effect code here (e.g., API call

  return () => {
    // Cleanup code (optional)
  };
}, [dependencies]); // Dependency array
```

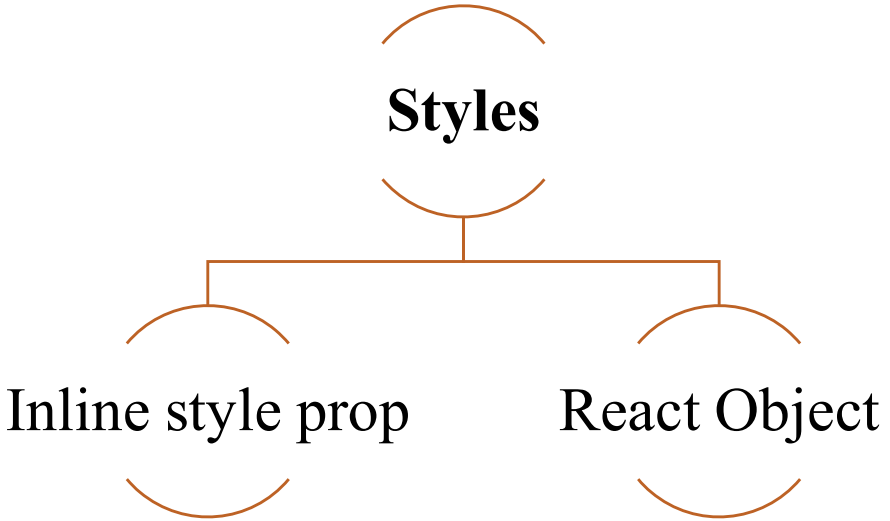
useEffect() Hook (Cont.)

- **Cases:**
  - No dependency array => useEffect() runs every render.
  - Empty dependency array => useEffect() runs only once when the component mounts. [componentDidMount]
  - Dependencies provided => useEffect() runs whenever any dependency changes. [componentDidUpdate]

- For more follow docs => https://react.dev/reference/react/useEffect

Styling

Styles in React Native



Inline + how to use style object

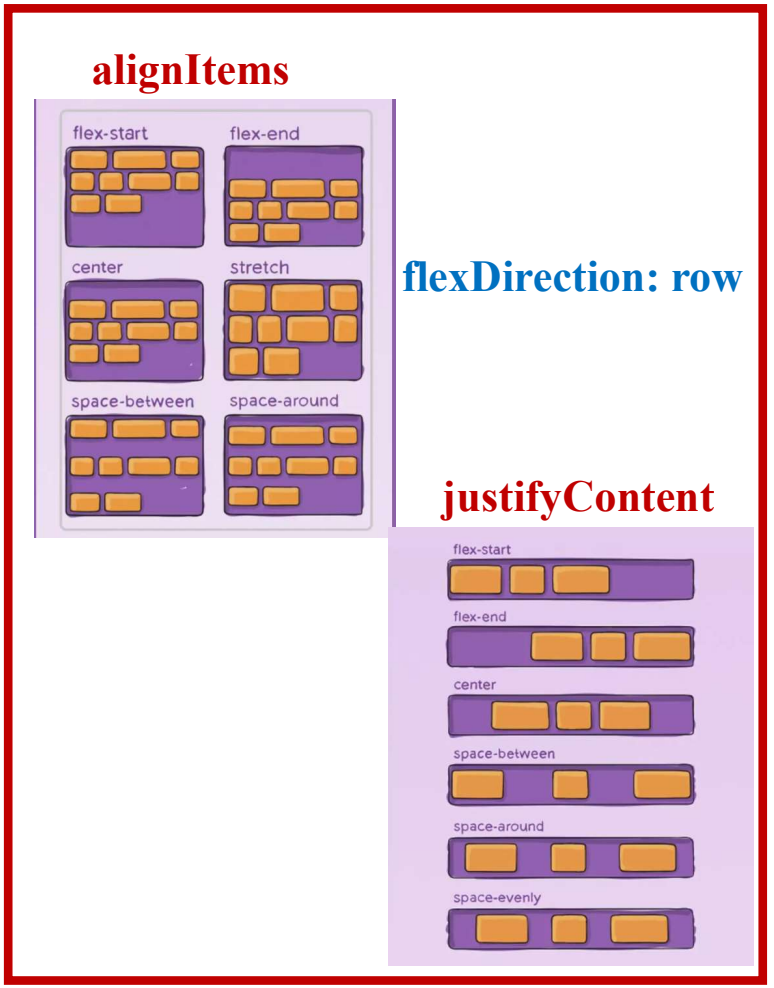
```
export default function App() {
  return (
    <View style={styles.container}>
      <View
      style={{ backgroundColor: 'red',
      width: 100,
      height: 100,
      justifyContent: 'center',
      alignItems: 'center'
      }}>
        <Text>1</Text>
      </View>
```

Style Object

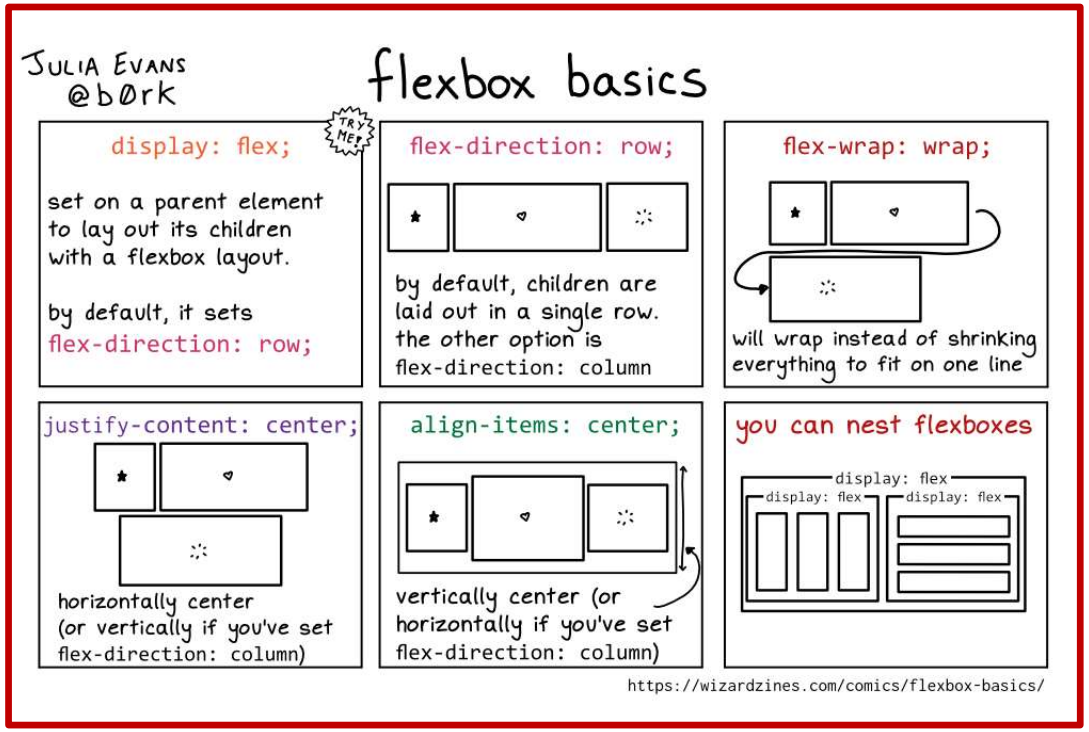
```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

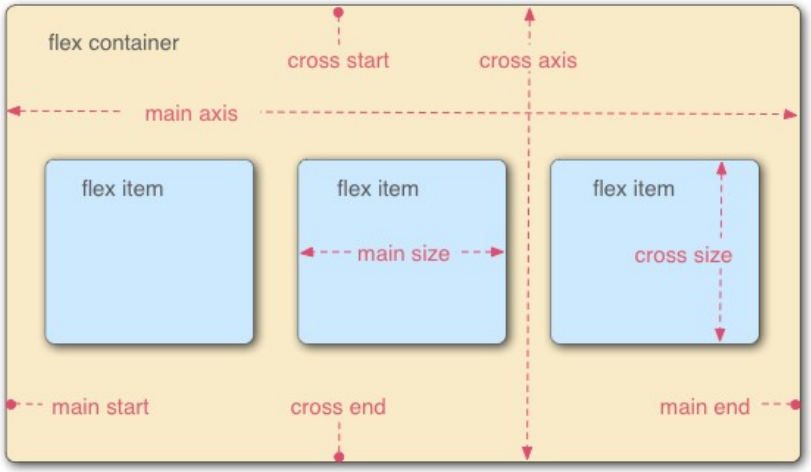
Flex Layout

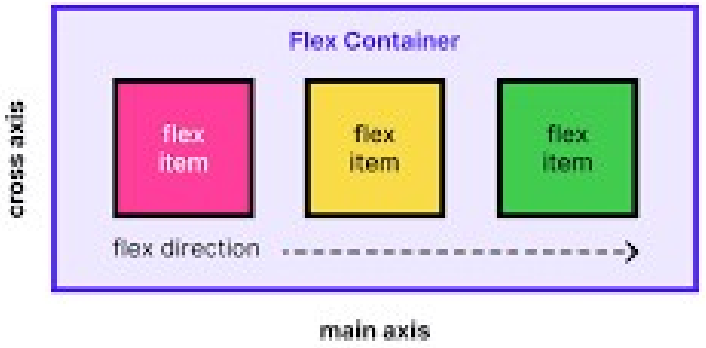
- It controls how elements are positioned into containers and how much space certain elements take up.
- Positioning is controlled via style settings applied to the element container.
- **justifyContent** organizes elements along the main axis *(i.e., flexDirection either row or column)*.
- **alignItems** organizes the elements along the cross axis *(i.e., orthogonal to flexDirection)*.
- N.B. Flex direction is column by default.
- N.B. Make sure to set width and height for each item and for the upper container.



Flex Layout (Cont.)







Props names are a little bit changed for react native for example:
React: justify-content
React Native: justifyContent

Flexbox - Example

JSX Components

Upper Container

Nested Container 01

Nested Container 02

Nested Container 03

```
return (
  <View style={styles.container}>
    <View>
      <Text>1</Text>
    </View>

    <View>
      <Text>2</Text>
    </View>

    <View>
      <Text>3</Text>
    </View>

  </View>
);
```

Item 01

Item 02

Item 03

Output before styling

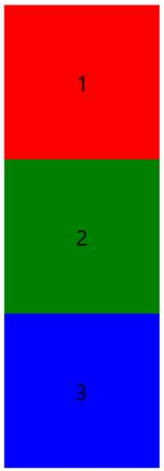
1
2
3

Flexbox - Example

1 **Output before styling parent container**

Inline styling for each item (nested view components)

```
style={
  {
    width: 100,
    height: 100,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: 'red'
  }}
```



JSX Components

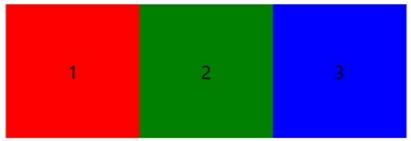
```
return (
  <View style={styles.container}>
    <View>
      <Text>1</Text>
    </View>

    <View>
      <Text>2</Text>
    </View>

    <View>
      <Text>3</Text>
    </View>

  </View>
);
```

2 **Output after styling parent container**



Parent Container Styling

```
const styles = StyleSheet.create({
  container: {
    padding: 50,
    width: '80%',
    height: 300,
    flexDirection: 'row',
    backgroundColor: '#fff',
    justifyContent: 'center',
    alignItems: 'center',
  },
});
```

Changing flexDirection to row in upper container

Flexbox - Example

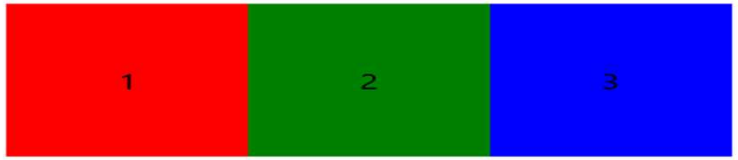
- If we set the flex of each box as follows:
  - Red box: 2
  - Green box: 1
  - Blue box: 1

- It is just like we have a total of 4 segments, red box will take 2 of them.

- By default, views take as much space as their child requires but with flex: 1, it takes as much space along the main axis (row => width), however, for the cross axis, we need to change the alignItems from the parent container itself.

Flexbox - Example

Changing justifyContent of upper container to '*space-between*' and width is 20%
Width and Height of nested container are 100



If we change justifyContent to ***'space-between'***:

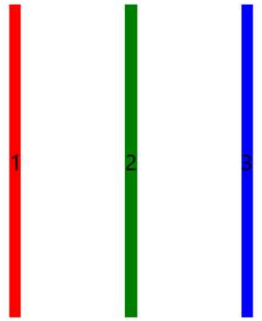
There will be spaces between all items.

If space-between has no effect this means that the items cover all the available width of the upper container

Changing justifyContent of upper container to '*space-between*'
Remove width and height of nested container

```
justifyContent: 'space-between',
alignItems: 'stretch',
```

- If we further change alignItems to ***'stretch'***:
  - The items will stretch to the full height of the upper container which is set to 300 in this example.



State Handling Example
Handling Button Event

- Create 3 buttons where the handler is: function, function with props, and function with state

```
<Button
  title="Click Me!"
  onPress={btnHandler}
/>
```

```
<Button
  title="Click Me!"
  onPress={btnHandlerCount.bind(null, ++c)}
/>
```

```
<Button
  title="Click Me!"
  onPress={btnHandlerStateCount}
/>
```

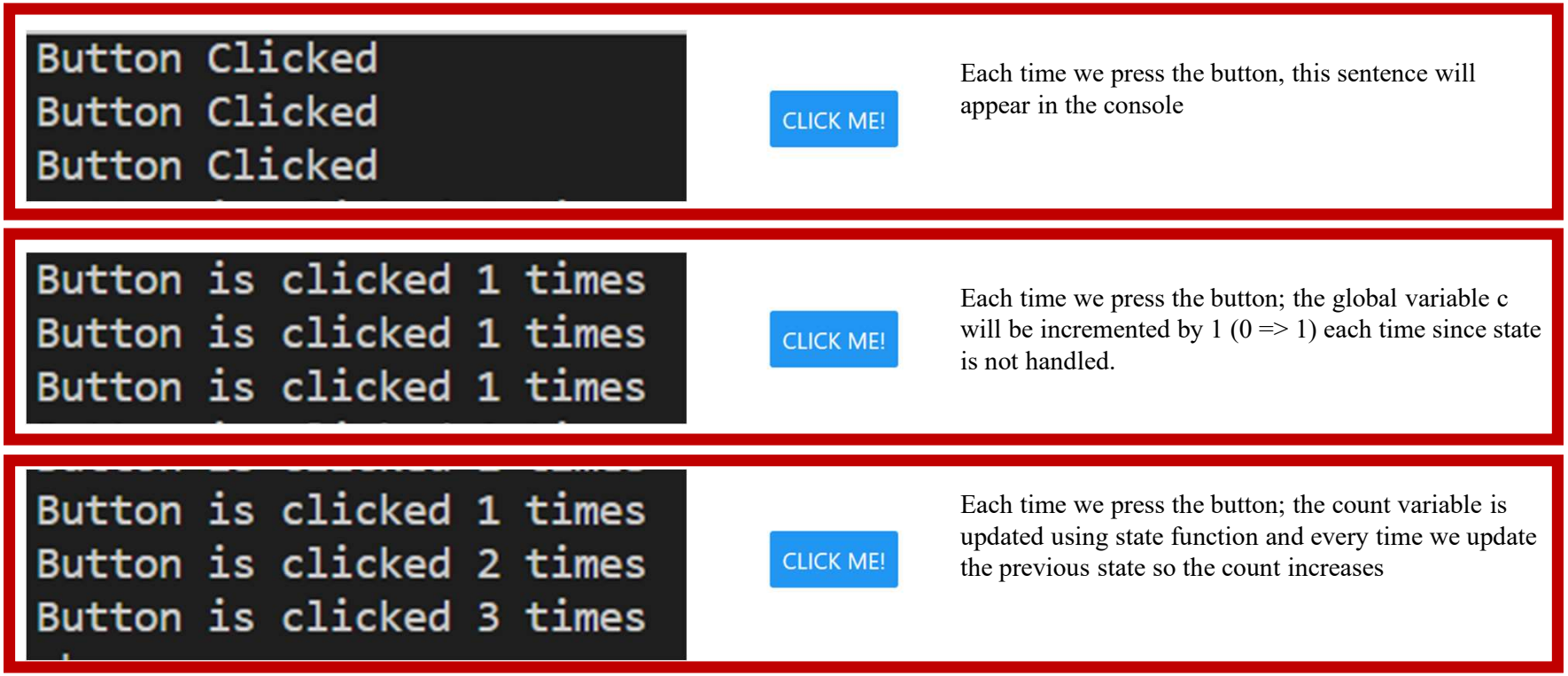
```
function btnHandler(){
  console.log('Button Clicked');
}
```

```
let c = 0;
function btnHandlerCount(c){
  console.log(`Button is clicked ${c} times`);
}
```

```
const [count, setCount] = useState(1);
function btnHandlerStateCount(){
  setCount(count + 1)
  console.log(`Button is clicked ${count} times`);
}
```

Button Component – handling event

- Click each button 3 times:



Assignment 01

- **Logic**
  - **Create a simple To-Do list that consists of these components:**
    - **Text** for adding the title.
    - **textInput** for input goals.
    - **Button** for adding an item to the list of goals.
    - **FlatList** for creating the list of goals and handling scrolling view.
  - **You will need two states to handle state change of:**
    - Adding text in the **textInput**.
    - Clicking the button for adding items to list.
  - **You will need one function for handling button click event.**

Assignment 01

- **Design**
  - Feel free to use your own colors but make sure to use consistent colors.
  - I provide you with a classic design to follow but feel free to make a more elegant design with excessive features.
  - Add custom fonts to your app => follow this link:
    - https://docs.expo.dev/develop/user-interface/fonts/
  - Make a report that includes:
    - Screenshots of your design including different actions and their results.
    - Colors and Fonts used in your project.
    - Provide code used for creating components, styling, and logic.
  - Make a small video (2min) showing your project in runtime.

Assignment 01