**Java Script:**

* Java Script is a scripting language.
* It is an Object based scripting language.
* It is a client-side language.
* It is cross platform language.

**React Native:**

* React Native is a JavaScript framework for building native mobile apps.

**Components:**

* In react native, we can create components using function or class.
* In react native we add all the running code and other components in our base (class/function) component.

**Function Component:**

* Doesn't maintain state.
* We can keep the data only does not change.
* They cannot manage their state or use lifecycle methods on their own.
* We can change the data using "use State" or another method.

* we first import React from the react library.

import React from 'react';

* Import React Native’s Text Core Component.

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

* Whatever a function component returns, is rendered as a react element. So, we don't need to use the render () method inside it.

export default function App () {

  return (

    <View>

      <Text>Open up App.js to start working on your app! </Text>

    </View>

  );

}

**Class Component:**

* Maintain State.
* We can change the data and keep the data.
* They can manage state and have life cycle methods like constructor (), render (), componentDidMount (), etc.
* We first imported React and Component from the react library.

import React, {Component} from 'react';

* we import a Text component that's a core component of react-native.

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

* A class must have a render method and whatever is returned inside it is rendered as a React element.

class Venkatesh extends Component {

    render () {

      return <Text>Hello, I am your Venkatesh! </Text>;

    }

  }

* And as with function components, you can export your class component:

class Venkatesh extends Component {

    render () {

      return <Text>Hello, I am your Venkatesh! </Text>;

    }

    export default Venkatesh;

**JSX:**

* JSX is an advanced version of Java Script.
* JSX stands for JavaScript XML.
* JSX allows us to write HTML in React.
* JSX makes it easier to write and add HTML in React.
* JSX converts HTML tags into react elements.

**Expressions in JSX:**

* With JSX you can write expressions inside curly braces {}.

const myelement = <h1>React is {5 + 5} times better with JSX</h1>;

* JSX will execute the expression and return the result.

# React is 10 times better with JSX

**One Top Level Element:**

* The HTML code must be wrapped in ONE top level element.
* So, if you like to write two headers, you must put them inside a parent element, like a div element.

const myelement = (

    <div>

      <h1>I am a Header. </h1>

      <h1>I am a Header too. </h1>

    </div>

  );

**Elements Must be Closed:**

* JSX follows XML rules, and therefore HTML elements must be properly closed.
* JSX will throw an error if the HTML is not properly closed.

 const element = <input type="text" />;

**Props:**

* React Native components have some Props which are helpful to customize the component.
* For example, placeholder, value, style etc are props. Parent view sets React Native Props and they are fixed throughout the lifetime of a component. Presentational components should get all data by passing Props.
* In React Native, most of the components can be customized at the time of their creation with different parameters. These parameters are known as props. They are immutable, and they cannot be changed.
* Most of react native core components can be customized with props too. For, when using Image, you pass it a prop named source to define what image it shows.

import React from 'react';

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

export const Login = () => {

    return (

       <View>

           <Image source={require('../Images/logo1.jpg')} style= {{width:150, height:150,}}/>

            <Text style= {{fontSize:35, alignItems:'center'}}>Login</Text>

       </View>

    );

};

The double curly braces {{}} surrounding style’s width and height in JSX, JS values are referenced with {}. This is handy if you are passing something other than a String as props, like an array number.

style= {{width:150, height:150}}

**State:**

* Props are set by the parent and they are fixed throughout the lifetime of a component. For data that is going to change, we have to use State.
* React provides so many different hooks which is very useful.
* To manage State React provides useState hook which will help you to get/set state.

**@ When would I use a Hook?**

* If you write a function component and realize you need to add some state to it, previously you had to convert it to a class. Now you can use a Hook inside the existing function component. We’re going to do that right now!
* It does not work inside classes.

**@Rules of Hooks:**

* Hooks should always be used at the top level of the React functions.
* Only Call Hooks from React Functions.

import React, {useState} from "react";

import {Button, Text, View} from "react-native";

const Cat = (props) => {

  const [isHungry, setIsHungry] = useState(true);

  return (

    <View>

      <Text>

        I am {props.name}, and I am {isHungry ? “hungry” : "full"}!

      </Text>

      <Button

        onPress= {() => {

          setIsHungry(false);

        }}

        disabled= {! isHungry}

        title= {isHungry ? "Pour me some milk, please!" : "Thank you!"}

      />

    </View>

  );

}

const Cafe = () => {

  return (

    <>

      <Cat name="Munkustrap" />

      <Cat name="Spot" />

    </>

  );

}

export default Cafe;

**@UseState Hook**

* you can simply import it from react using

import React, {useState} from 'react';

* Once you Import the useState hook you need to define state variable and setState function like this

const [isHungry, setIsHungry] = useState(true);

* It creates a “state variable” with an initial value—in this case the state variable is isHungry and its initial value is true
* It creates a function to set that state variable’s value—setIsHungry
* Next you add the Button Core Component and give it an onpress prop:

<Button

        onPress= {() => {

          setIsHungry(false);

        }}

      />

* Now, when you press the button, calling the setIsHungry(false). This sets the state variable isHungry to false. When isHungry is false, the Button’s disabled prop is set to true and its title also changes:

<Button

        onPress= {() => {

          setIsHungry(false);

        }}

        disabled={!isHungry}

        title= {isHungry ? "Pour me some milk, please!" : "Thank you!"}

      />

**Arrow Function:**

* Arrow functions were introduced in ES6.
* Symbol of Arrow Function **=>**
* To write shorter syntax for the function we use Arrow.
* The react Native arrow function come with the combination of equal and greater than sign, it can be show below like syntax:

const submitHandler= () => {

* Arrow functions make your code look cleaner and more presentable but on top of that there are more reasons to use them in React.

**@Use arrow function to avoid binding ‘this’ to methods:**

* Usually when you want to access this inside a class method you would need to bind it to your method like so:

class Button extends Component {

  constructor(props) {

    super(props);

    this.state = {clicked: false};

    this.handleClick = this.handleClick.bind(this);

  }

  handleClick () {

    this.props.setState ({clicked: true});

  }

  render () {

    return <button onClick={this.handleClick}>Click Me! </button>;

  }

}

**Without an arrow function**

Binding this to handleClick in the constructor allows us to use this. setState from Component inside handleClick. Without this binding, this is re-scoped for handleClick and therefore cannot be used with the setState method.

* **But this is completely unnecessary, extra code!**
* **You can clean up this ugliness by using some new ES6+ features. Here is the same component, rewritten using class properties and arrow functions to avoid binding this to handleClick:**

class Button extends Component {

  state = {clicked: false};

  handleClick = () => this.setState({clicked: true});

  render () {

    return <button onClick={this.handleClick}>Click Me! </button>;

  }

}

**With an arrow function**

* As you can see using arrow functions in class components can end up saving you lines of code and time.
* Here you can see 3 lines of code have been cut out just from changing that one function.

**@Why Does This Work?**

* This works because of two reasons:
* Arrow functions, by their nature, so we don’t need to bind this in the class constructor.
* JavaScript has first-class functions, meaning functions are treated the same as data. So, arrow functions can be assigned to variables.