**ReactJS Notes**

**JavaScript:** ‘this’ keyword, filter, map and reduce

**ES6:** let & const, arrow functions, template literals, default parameters, object literals, rest and spread operators and destructuring assignment.

**Creating an app**

npx create-react-app <app name>

cd <app name>

npm start

**Functional Component**

**Shortcut: rsf+Tab**

Make a components folder and create a Greet.js

import React from ‘react’

< create a function called Greet>

export default Greet;

In App.js

*import Greet from ‘./components/Greet’*

In div: *<Greet/>*

**OR**

In Greet.js

*Import React from ‘react’*

*Export <create a function called Greet>*

In App.js

*import {Greet} from ‘./components/Greet/’*

In div: *<Greet/>*

**Class Component**

**Shortcut: rcc+ Tab**

**For constructor: con+ Tab**

Welcome.js in Components folder

Import React, {Component} from ‘react’

Class Welcome extends Component{

render(){

return <h1>Class Component</h1>

}

}

Export default Welcome;

**JSX Version**

Hello.js in Components folder

Import React from ‘react’

Const Hello=()=>{

Return(

<div>

<h1>Hello nini</h1>

</div>

)

}

Export default Hello

In App.js

Import Hello from ‘./components/Hello’

In <div>: <Hello/>

**Without JSX Version**

In Hello.js

Const Hello=()=>{

Return React.createElement(‘<the HTML tag to be rendered>’,<additional properties>, <children for the tag>)

}

Eg.

Const Hello=()=>{

Return React.createElement(‘div’, {id: ‘hello’, className:’dummyClass’, React.createElement(‘h1’,null, ‘Hello Nini’))

}

**Using Props**

**In Functional Component**

In Greet.js

Const Greet= (props)=>{

console.log(props)

return <h1>Hello {props.name} is {props.adjective}</h1>

}

In App.js

<Greet name=”Nini”/ adjective=”Awesome”>

**In Class Component**

In Welcome.js

Class Welcome extends Component{

render(){

return <h1>Welcome {this.props.name} is {this.props.adjective}</h1>

}

}

**Note: Props is immutable. However state can be changed.**

**SetState function**

In Counter.js

Class Counter extends Component{

Constructor(props){

Super(props)

This.state{

Count:0

}

}

increment(){

this.setState({count:this.state.count=this.state.count+1}, <Callback Function>)

}

render(){

return (<div>

<div>Count – {this.state.count}</div>

<button onclick={()=>this.increment()}>Increment</button>

</div>

);

}

}

**Note: setState lets React know that it has to update the state that has been altered.**

**Also if you want to implement something after the this.SetState function then call it using the callback function and NOT in the line before console.log**

**Event Handler**

**Function Click:**

In FunctionClick.js

Function FunctionClick(){

Function clickHandler(){

Console.log(‘Button Clicked’)

}

Return(<div>

<button onClick={<Function name WITHOUT parenthesis>}>Click</button>

</div>);

}

**Class Click:**

<button onClick={this.<Function name WITHOUT parenthesis>}Click</button>

**Event Binding**

**Approach 1:**

In EventBind.js

Constructor(props){

Super(props);

this.state={

message: ‘Hello’

}

}

clickHandler(){

this.setState({

message:’Goodbye’

})

}

Render(){

Return(

<div>

<div>{this.state.message}</div>

<button onClick={this.clickHandler.bind(this)}></button>

</div>);

}

**Approach 2:**

<button onClick={()=>this.ClickHandler()}></button>

**Approach 3:**

In the constructor

This.clickHandler=this.clickHandler.bind(this)

In render():

<button onClick={this.clickHandler}>

**Approach 3 is better since the binding occurs only once as opposed to the other approaches.**

**Approach 4:**

Remove the approach 3 line from constructor

Instead of the previous clickHandler function, use:

clickHandler=()=>{

this.setState({message:’GoodBye’})

}