Prequisite:

Nodejs

Visual studio code:

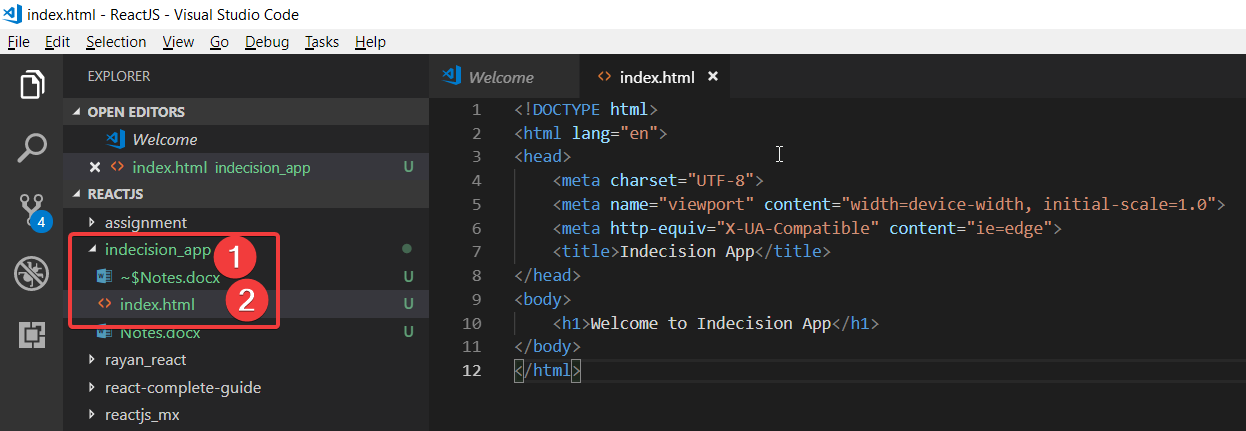
Download and install nodejs:

After Node installation:

Install Yarn Globally

npm install -g yarn

Create indecision folder and index.html file inside



Install live server globally:

Yarn add global live-server

Or

Npm install -g live-server

Go to indecision folder: D:\ReactJS\indecision\_app

Check live server version” live-server -v

Run live-server:

live-server public

import js libraries in body:

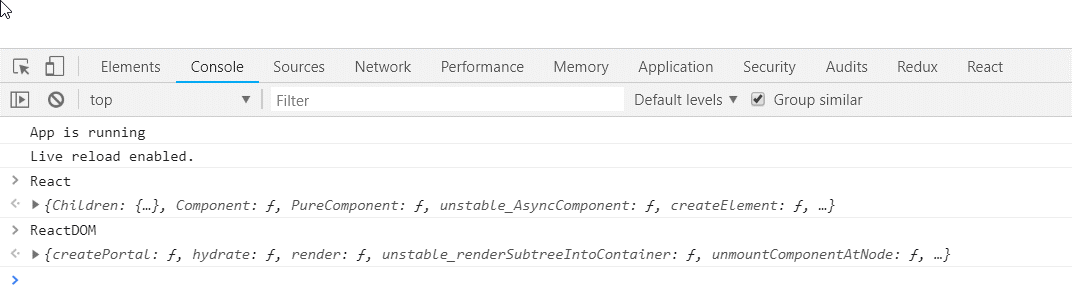
<script src="https://unpkg.com/react@16.0.0/umd/react.development.js"></script>

<script src="https://unpkg.com/react-dom@16.0.0/umd/react-dom.development.js"></script>

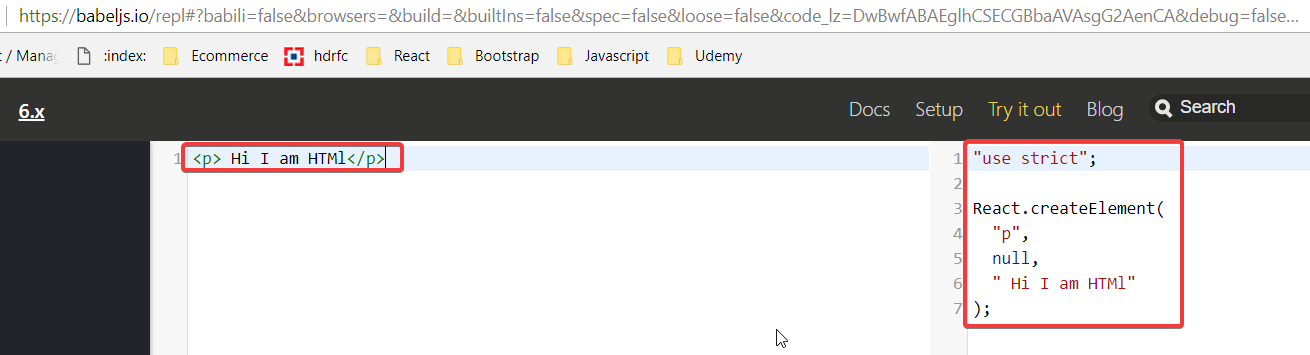
<script src="./scripts/app.js"></script>

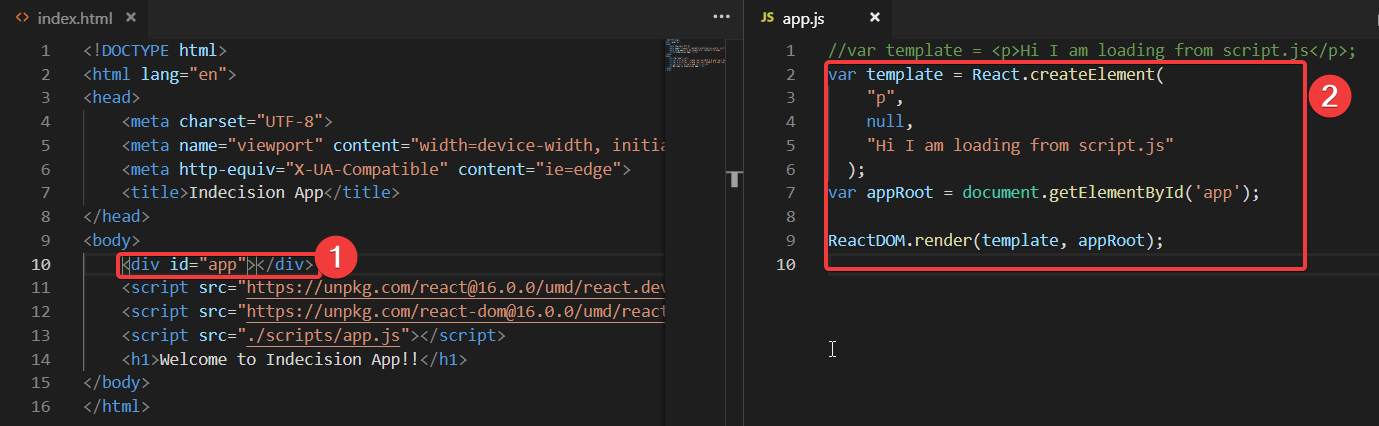
JSX: Javascript XML

Inspect react and react dom in console:



Babel is javascript compiler. Taking features from Es6 and compiling them down to ES5 code





Installing and configuring Babel:

Npm install -g [babel-cli@6.24.1](mailto:babel-cli@6.24.1)

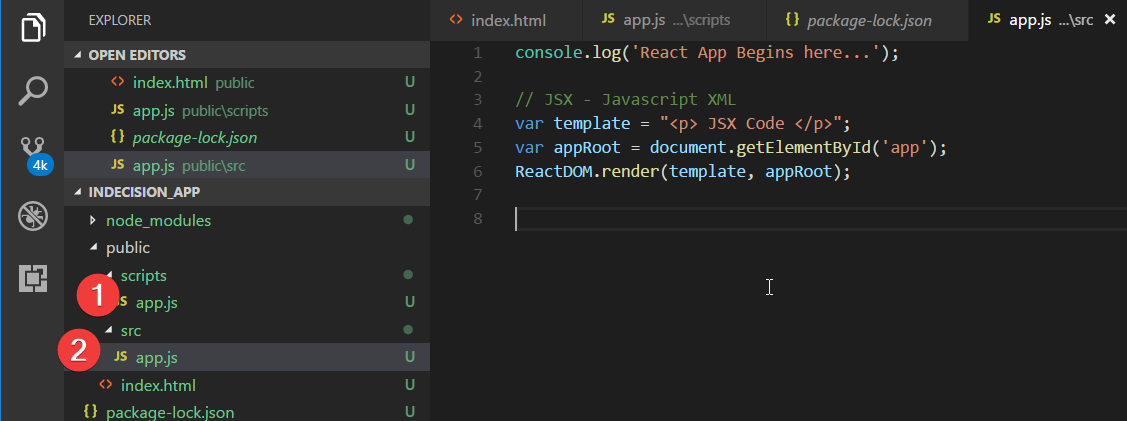
Run init command:

Npm init

Package.json: Main advantage of package.json file is to outline dependencies that our project needs in order to run

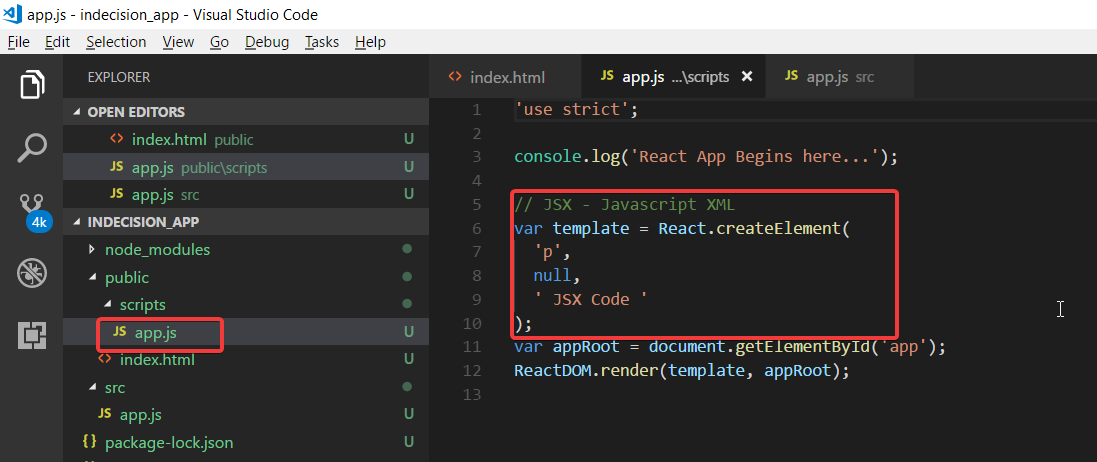
Install React presets: npm add [babel-preset-react@6.24.1](mailto:babel-preset-react@6.24.1) [babel-preset-env@1.5.2](mailto:babel-preset-env@1.5.2)

Create app.js file in src folder that will load our content jsx and that babel will compilesdinto scripts/app.js



babel src/app.js --out-file=public/scripts/app.js --presets=env,react

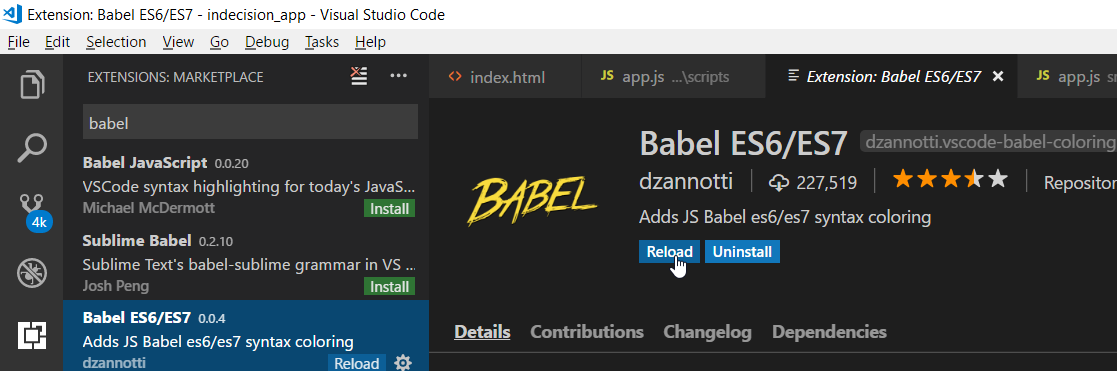
Babel will compile normal jsx code to vanilla js code:



Babel will monitor the changes in app.js file if app.js text changed it will recompile automatically:

babel src/app.js --out-file=public/scripts/app.js --presets=env,react --watch

Recommended extension for react development:



Use expressions in react:

var user = {

name: "Sunil",

age: 32,

location: 'Philadelphia'

}

var userName = "Anil";

var userAge = 28;

var userLocation = "Jammu and Kashmir";

var templateTwo = (

<div>

<h1>Name: {user.name}</h1>

<p>Age: {user.age}</p>

<p>Location: {user.location}</p>

</div>

);

Use conditioning in react:

var user = {

name: "Sunil Singh",

age: 17,

location: 'Philadelphia'

}

var userName = "Anil";

var userAge = 28;

var userLocation = "Jammu and Kashmir";

function getLocation(location){

if(location){

return <p>Location: {location}</p>

}

}

// check if user age exist then we will show content in {user.age} section

var templateTwo = (

<div>

<h1>Name: {user.name ? user.name.toUpperCase() : 'Anonymus'}</h1>

{user.age >= 18 && <p>Age: {user.age}</p> }

{getLocation(user.location)}

</div>

);

ES6:

|  |  |  |
| --- | --- | --- |
| Var | Let | Const |
| Can be redefined | Can not redefined | Can not redefined |
| Can be re-assigned | Can be re-assigned | Can’t re-assigned |
| Var has function scope | Let has block level scope / We can define value before block and then we will assign value to let variable | Const has block level scope / We can define value before block and then we will assign value to let variable |

// Var keyword

var fullName = "Anil";

var fullName = "Sunil";

console.log(fullName);

// let Keyword

let job = "Engineer";

job = "Designer"

console.log(job);

// const Keyword

const dob = 1990;

//job = "Designer"

console.log(dob);

// Var keyword Function Scope

// function getPetName(){

// var petName = "Tinku";

// return petName;

// }

// getPetName();

// console.log(petName);

const personName = "Anil Singh";

let firstName;

if(fullName){

firstName = personName.split(" ")[0];

console.log(firstName);

}

console.log(firstName);

Arrow functions :

function square(x){

return x \* x;

}

console.log("Square root of 3 is: "+square(3));

const squareArrow = (x) => {

return x \* x;

}

console.log("Square root of 5 is: "+squareArrow(5));

const squareArrow6 = (x) => x \* x;

console.log("Square root of 6 is: "+squareArrow6(6));

Arrow function 2:

// Arguments object(unable to pass more function parameters and access them using argument obeject) - No longer bound with arrow function

const add = (a, b) => {

//console.log(arguments);

return a+b;

}

console.log(add(10,5, 1000));

// this keyword no longer bound

const user = {

name : "Anil Singh",

cities : ['Philadelphia', 'New York', 'Dublin'],

printPlaceLived: function(){

// this.cities.forEach((city) => {

// console.log(`${this.name} has lived in ${city}`)

// })

return this.cities.map((city)=> `${this.name} has lived in ${city}`);

}

}

console.log(user.printPlaceLived());

JSX does not have two way data binding by default that why we should re render our app as shown in following counter app :

let count =0;

const plusOne = () =>{

count++;

renderCounterApp();

}

const minusOne = () => {

count--;

renderCounterApp();;

}

const reset = () => {

count = 0;

renderCounterApp();;

}

const appRoot = document.getElementById('app');

const renderCounterApp = () => {

const templateThree = (

<div>

<h1>Count: {count}</h1>

<button onClick = {plusOne}>+1</button>

<button onClick = {minusOne}>-1</button>

<button onClick = {reset}>Reset</button>

</div>

);

ReactDOM.render(templateThree, appRoot);

};

renderCounterApp();

forms and form inputs:

const app = {

title: 'I am the title',

subTitle: 'I am subtitle',

options: []

}

const title= "I am const title";

const subTitle = "I am const subtitle";

function getOptions(options){

return options.length > 0 ? "Here are your options" : "No Options" ;

}

const addOption = (event) => {

event.preventDefault();

let optionVal = event.target.elements.options.value;

app.options.push(optionVal);

event.target.elements.options.value = '';

renderDom();

}

const removeAll = () => {

app.options = [];

renderDom();

}

const renderDom = () => {

const template = (

<div>

<h1>{app.title}</h1>

{app.subTitle && <p>{app.subTitle}</p>}

<p>Legth of option values are: {app.options.length} <button onClick={removeAll}>Remove All Options</button></p>

<form onSubmit={addOption}>

<input name="options"></input>

<button>Add Option</button>

</form>

</div>

);

const appRoot = document.getElementById('app');

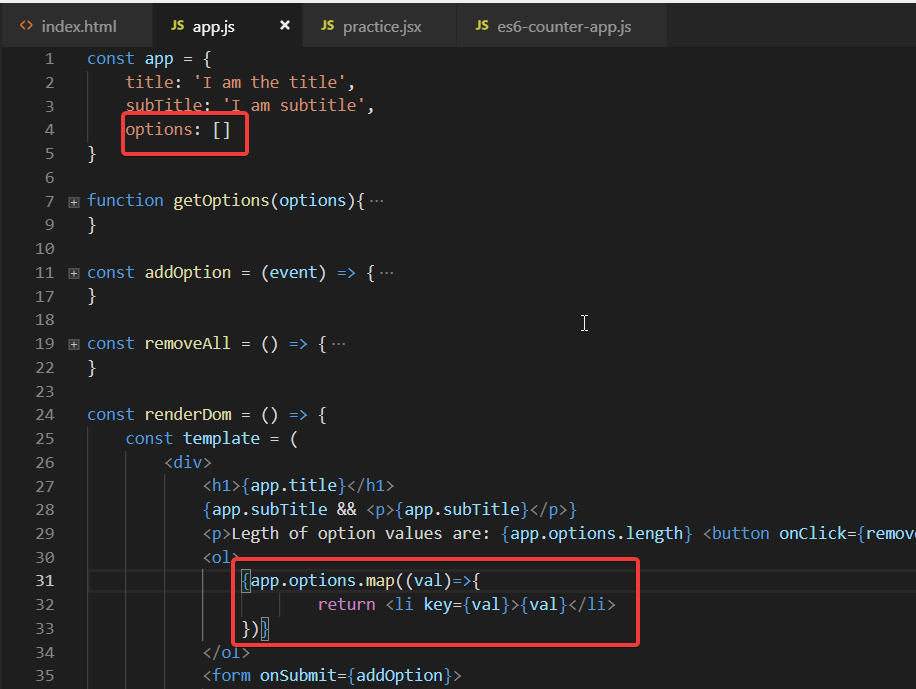
ReactDOM.render(template, appRoot);

}

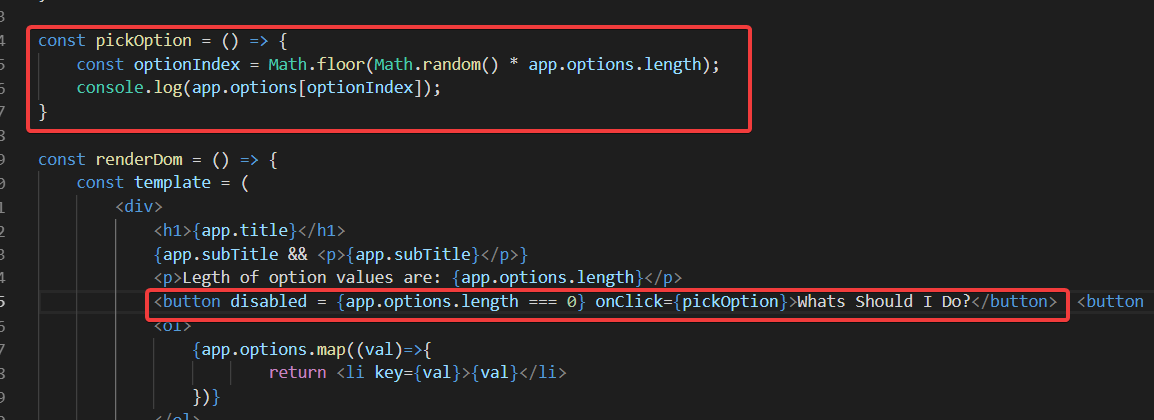
renderDom();



Render options array values inside order list(JSX inside JSX):



Pickup a option:



Classes in ES6:

class Person {

constructor(name="Anonymus", age= 19){

this.name = name,

this.age = age

}

getGreetings(){

return `Hi ${this.name} your current age is ${this.age}. `;

}

}

class Students extends Person{

constructor(name, age, major){

super(name, age); // If we extend class then we must use super keyword to use parent class constructor.

this.major = major;

}

hasMajor(){

return !!this.major;

}

getGreetings(){

let description = super.getGreetings(); // call parent class greetings function

if(this.hasMajor()){

description += ` your major is ${this.major}`;

}

return description;

}

}

class Traveler extends Person{

constructor(name, age, homeLocation){

super(name,age);

this.homeLocation = homeLocation;

}

getGreetings(){

let description = super.getGreetings();

if(this.homeLocation){

description+= `${this.name} has home location: ${this.homeLocation}. `;

}

return description;

}

}

const anil = new Students('Anil Singh', 28, 'IT');

console.log(anil.getGreetings());

const sunil = new Students('Sunil Singh', 32);

console.log(sunil.getGreetings());

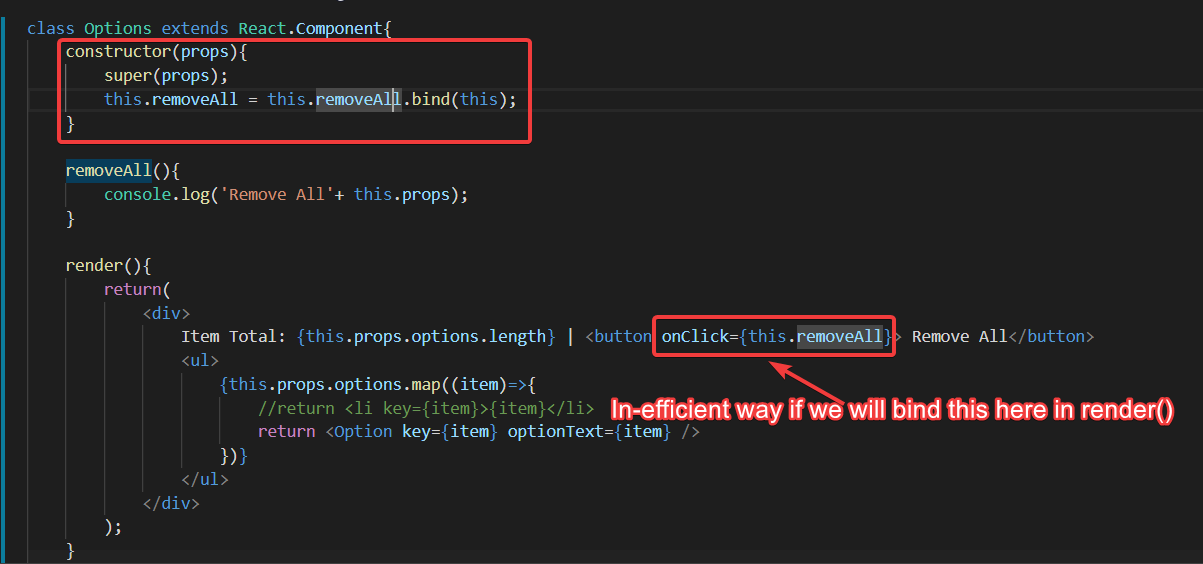
const subash = new Traveler("Subash Chandra", 31, 'Orissa');

console.log(subash.getGreetings());

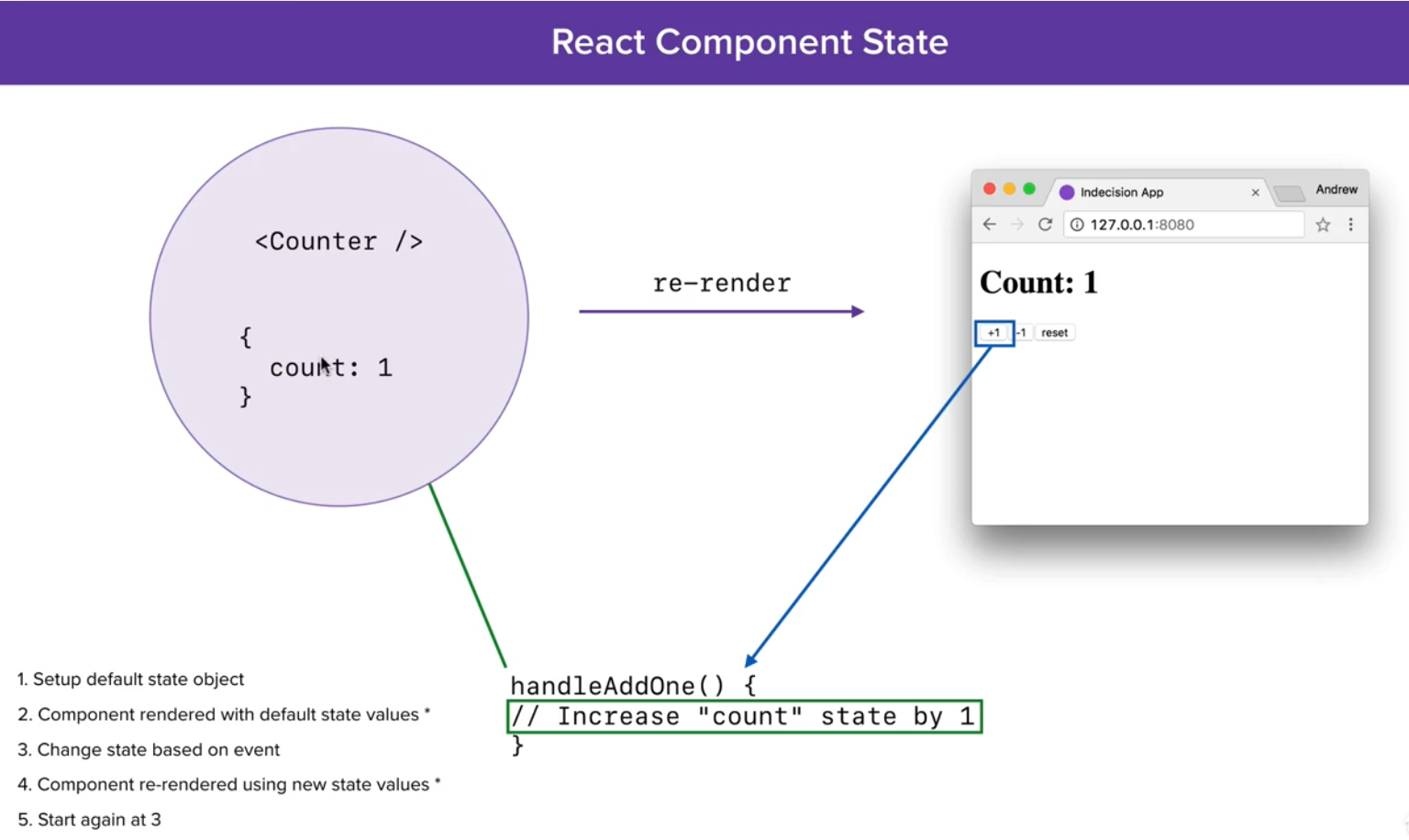
const nithin = new Traveler("Nithin EV", 30);

console.log(nithin.getGreetings());

Method binding to support this keyword in class functions. bind():



Component State:



Counter App with state:

class Counter extends React.Component{

constructor(props){

super(props);

this.plusOneHandler = this.plusOneHandler.bind(this);

this.minusOneHandler = this.minusOneHandler.bind(this);

this.resetHandler = this.resetHandler.bind(this);

this.state = {

count : 0

}

}

plusOneHandler(){

this.setState((prevState)=>{

return {

count: prevState.count + 1

}

})

}

minusOneHandler(){

this.setState((prevState) => {

return {

count: prevState.count -1

}

})

}

resetHandler(){

this.setState(() => {

return {

count : 0

}

})

}

render(){

return (

<div>

<h1>Count: {this.state.count}</h1>

<button onClick={this.plusOneHandler}>+1</button>

<button onClick={this.minusOneHandler}>-1</button>

<button onClick={this.resetHandler}>Reset</button>

</div>

);

}

}

ReactDOM.render(<Counter />, document.getElementById('app'))

Visibility example with state:

class VisibilityToggle extends React.Component{

constructor(props){

super(props);

this.toggleButton = this.toggleButton.bind(this);

this.state = {

visibility : true

};

};

toggleButton(){

console.log('toggleButton'+this.state.visibility);

this.setState((prevState)=>{

return {

visibility: !prevState.visibility

}

});

console.log('toggleButton'+this.state.visibility);

}

render(){

return(

<div>

<h1>Visibility Toggle</h1>

<button onClick={this.toggleButton}>

{this.state.visibility ? "Hide Detials" : "Show Details"}

</button>

{ this.state.visibility && <p>Here are some details you can see now</p>}

</div>

);

}

}

ReactDOM.render(<VisibilityToggle />, document.getElementById('app'));