*Understanding React*

About

I am uploading my react notes and all source code. This is based on the course *React-A complete guide* available on Udemy. I have tried to be as detailed as possible while writing all the explanations. The whole course follows a workflow where a simple project is made and all the basic features are integrated into it step by step. I have pasted the code everywhere for better understanding. Hope this helps.

Contents

[JavaScript 4](#_Toc62168138)

[Base features and Syntax 9](#_Toc62168139)

[Styling react Components 22](#_Toc62168140)

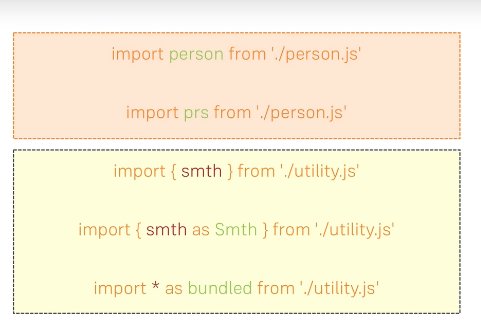
[Debugging React Apps 29](#_Toc62168141)

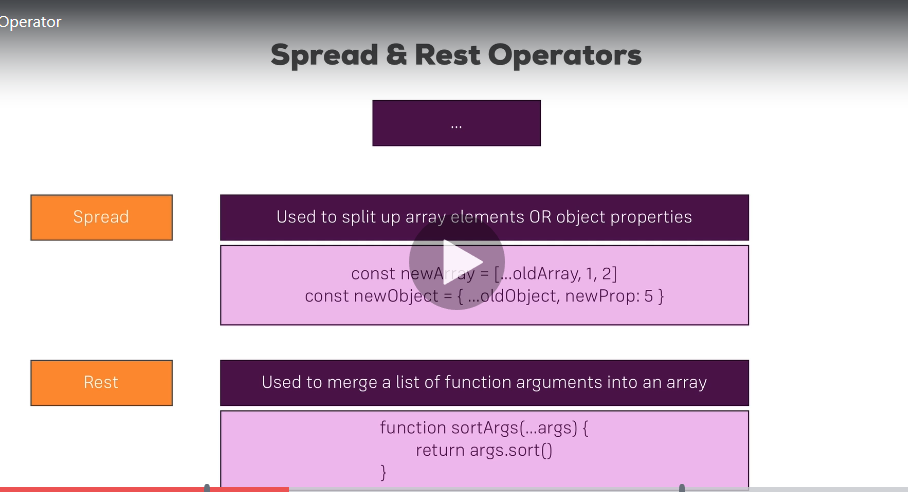
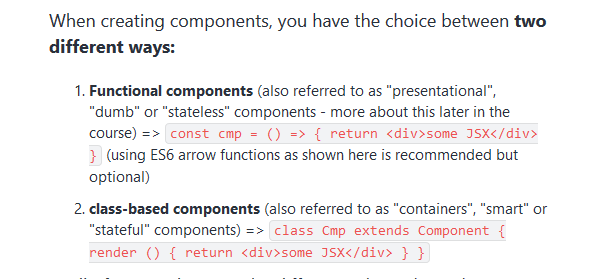
[Deeper into React Components 30](#_Toc62168142)

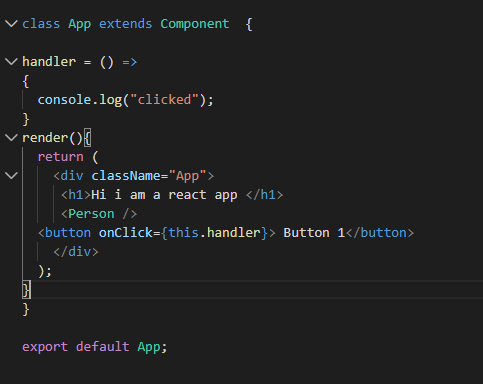
# JavaScript

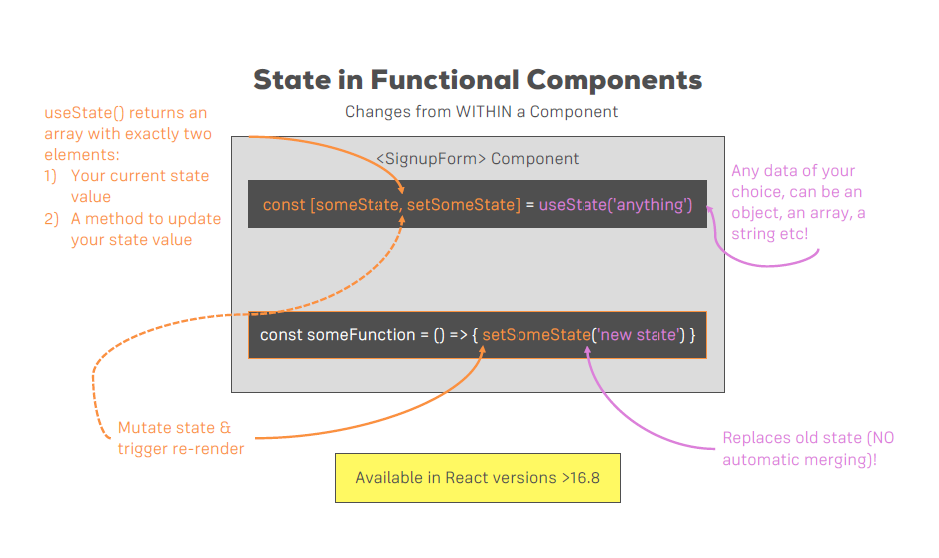
This module is a breif introduction to next gen features of JavaScript.

1. Arrow function =()=>{ .. .} we can skip the brackats if we have only one paramter.
2. If exporting by default then we can name it any thing we want while importing otherwise we need to explicitly mention which function or variable we are importing using { … }.



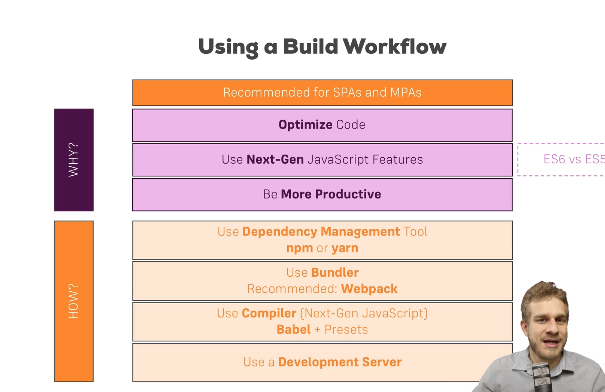
1. While extending class if the constructor in derived class is defined the we need to call use the word super(); which calls the constructor of parent class.
2. 
3. Primitive data types like var,let just copy the data meanwhile array,striing are reference type so if we are reassigning them then we are bascially making changes at two different locations. So use spread while re assigning them to avoid unexpected results.
4. JS array functions developer.mozilla
5. React lets us write jsx with simple html.
6. Classname must be used because class keyword is already reserved in javascript.
7. With jsx in return function where we write the actual javascript we can only return one div.
8. Component is a funnction returning some jsx.
9. Components which we are importing should bgein with a capital letter as all small letter keywords are revsered for react.
10. 
11. To make react understand we are usng js in between html just use { } .
12. Sending props to components-> functional components can use props.name and props.age etc. while class components will only use props as this.props.name and this.props.age.
13. Props.childern refer to any component that comes in between the opening ans closing tags of our component.
14. State is a property of component class. If we change the state it will lead to rerender of my component. Now functional components can’t have state only class components extended by component. state  simply is a property of the component class, you have to call it state  though - the name is not optional. You can then access it via this.state  in your class JSX code (which you return in the required render()  method).
15. The event handler functions that we are making will be running a function of that class described before render must be written without () as () will make react render the function.



1. Event handlers in react : <https://reactjs.org/docs/events.html#supported-events>
2. Setstate function overwrite only the state variable we are changing.
3. Only props and state can compel react to re render.
4. As we know state varaiable can only be used with the react class components and to use the state variables with the functional component we need to use **react hooks**.
5. **UseState** needs to be imported from react which is a function to which we can pass our initial state and what it does it returns two things and two things always 1. The initial state and when it is updated the updated state 2. The function which will lets us update the state.
6. We can have a function inside a function in react or javascript in general.
7. Important point about the function which lets us change the state in functional component:
8. The setState overwrites whichever state component we want to change.
9. But this new function replaces everything.
10. 
11. We can pass methods as props to change thee state or anything.

# Base features and Syntax

1. Why we need to build a work enviornment. And what actually create-react-app provide:



2.  // return (

  //   <div className="App">

  //     hello

  //   </div>

  // );

The above code gets converted to: (helping us write htmlish code)

return React.createElement('div',{className:'App'},React.createElement('h1',null,'Hello guysz'));

**The above code is easy to write but it is important to know the inside working of it.**

Inside the above statement get translated to this. That is why when making class components in react we need to import react so that we can use the create element fucntion.

However it is not necessary to import react while making a functional component.

1. We just saw how jsx helps us write html code which is converted to javascript. But there are some restrictions of jsx:

* We can’t use certain keywords which are reserved in react**. Class** for example is react reserved word. Hence in order to refer to css class we use **className**.
* Everything should be enclosed in a div as a best practice.

1. Functional component:In most basic form it is just a javascript function which return some jsx. Remember to import your functional component by giving it an upper case name. As react reserve most of the lower case keywords. For ex.: Div can be your custom component which can be used along side div

. import React from 'react';

const Person = ()=>{

   return <p>This is a person!!</p>;

};

export default Person;

1. Given above is a functional component. *This is a person,*it is a static html code.In order to make function calls and use javascript functions we need to tell the react by using curly braces {}.

Like this

Like this: import React from 'react';

const Person = ()=>{

   return <p>This is a person!! And he is {Math.floor(Math.random()\*10)} years old</p>;

};

export default Person;

1. **How to use pass parameters to functional components:**

We can do so using props. This way we can send values to our components. And the same component just like a function can be used to render different things based on different props passed.

Parameter can be passed using props(or you can call it anything)

<Person name ='Akash' age='21'></Person>

This is normal prop. This is children prop.

      <Person name ='Sky' age ='22'>This is an alter ego</Person>

Above we see two ways by which we are passing parameters. This is known as passing value from outside of component.

Normal props can be used to send simple parameters while the children props can be used to send even react elements to complex html codes to anything.

**How they are used inside components**:

return (

            <div>

                <p>This is a {props.name}!! And he is {props.age} years old</p>

                <p>{props.children}</p>

            </div>

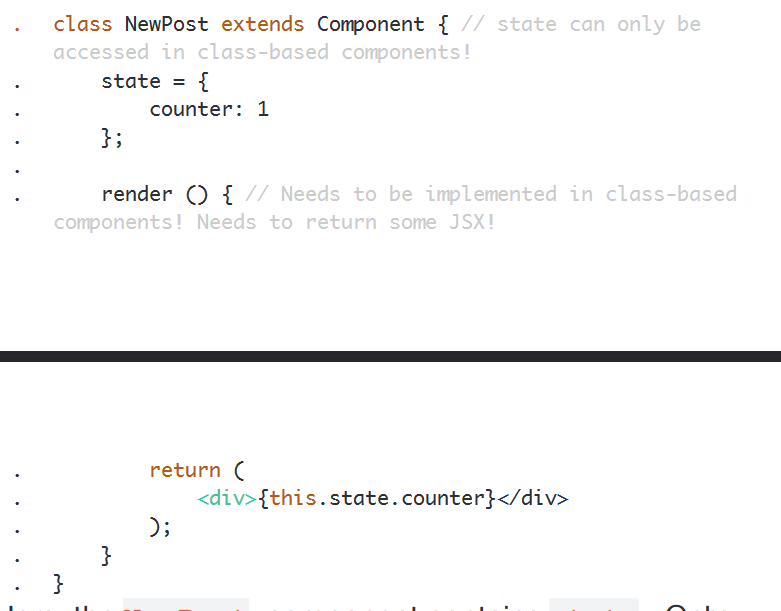
        )

This now opens a variety of possiblities. Same component although will have the same structure but can be customized to the extent limited to only the imagination of the developer.

**State:**

It is a property of the class inherited from components Whilst props allow you to pass data down the component tree (and hence trigger an UIupdate), state is used to change the component, well, state from within. Changes to state also trigger an UI update

**Example:**



. return (. <div>{this.state.counter}</div>. );. }.}

Here, theNewPost component containsstate. Only class-based components can define and usestate. You can of course pass thestate down to functional components, but these then can't directly edit it.

State simply is a property of the component class, you have to call itstate though - the name is not optional. You can then access it viathis.state in your class JSX code (which you return in the requiredrender()method).

When ever state changes (taught over the next lectures), the component will re-render and reflect the new state. The difference toprops is, that this happens within one and the same component - you don't receive new data (props) from outside!

**Event Handlers:**

We can use event handlers in react to change the react state.Remember to always pass a reference to eventHandler function and not the function. As react will render it if done otherwise.

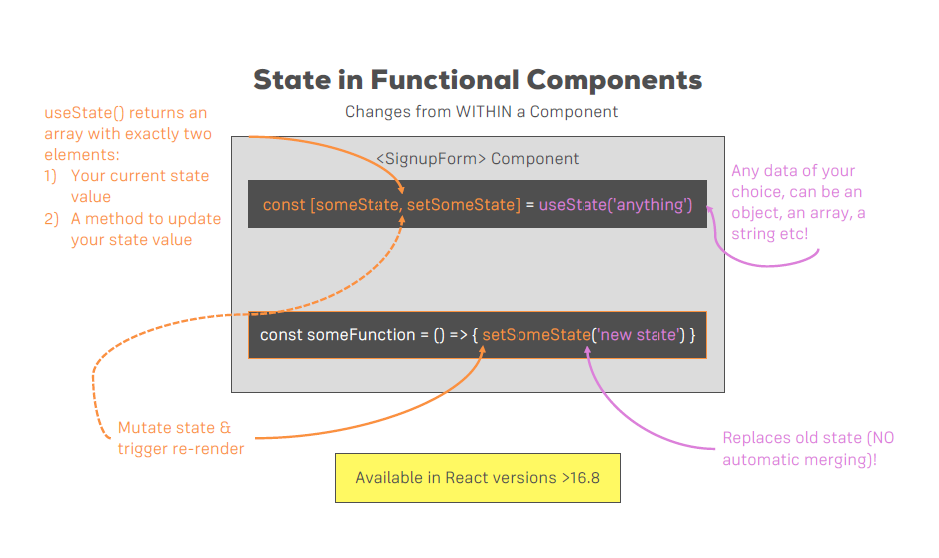
**Correct way**: onClick= { this.eventHandler}

**Incorrect way**: onClick= { this.eventHandler() } //This will execute it instantly.

What all events you can listen to:

<https://reactjs.org/docs/events.html>

**Manipulating states:**

****

Set can be changed(or mutated) only by using setState method. This method only overwrite the state we pass and nothing else. For example if there are name,age,address inside state variables and we overwrite age only then name and address will remain same. Remember it only overwirte.

Example: state = {

    persons: [

      {name:'Akash',age:22},

      {name:'Sky', age:40}

    ],

    otherstateVariable:"somevalue"

  }

  personswaitchHandler = ()=> {

    this.setState ({

      persons: [

        {name:'Kali',age:22},

        {name:'Sky', age:69}

      ]

    }

    )

  }

Other state variable will remain unchanged. Also note down the syntax of setState(). It is a method.

**Handling states of functional Components:**

This can be done in react 16.8 and higher versions using a new feature known as react hooks. They are a number of them.One of them is:

**UseState** needs to be imported from react which is a function to which we can pass our initial state and what it does it returns two things and two things always 1. The initial state and when it is updated the updated state 2. The function which will lets us update the state.

const [PersonState,ChangePersonState]= useState({

    persons: [

      {name:'Akash',age:22},

      {name:'Sky', age:40}

    ],

    otherstateVariable:"somevalue"

  });

  const personswaitchHandler = ()=> {

    ChangePersonState ({

      persons: [

        {name:'Kali',age:22},

        {name:'Sky', age:69}

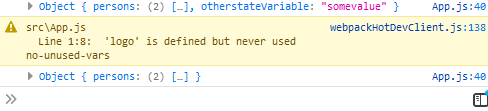
      ]

    }

    )

  }

If we do console.log(PersonState) we get:



Notice how otherstateVariable is missing. This enfoce the fact that UseState returns a function which doesn’t merge the new state but other but replaces it.So in order to manage the other state as well we will use useState again and extract that otherstatevariable.

It is a good practice to use as much stateless components as possible in our project.

**Passing methods as refernce to components:**

In our code,personswitchHandler is a function defined in App.js. Can we send it to some other component ? Think about it. Sending a function defined in main file to some other component.This can be done easily by passing a refernce of that function.

<Person

      name ={this.state.persons[0].name}

      age={this.state.persons[0].age}

      click={this.personswaitchHandler} //Here i am sending a method's refernce to my react component.

      >

      </Person>

And how to use it:

<p onClick={props.click}>This is a {props.name}!! And he is {props.age} years old</p>

**Passing parameters to eventhandler functions:**

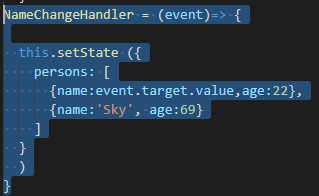
Now what if we want to send some paramter to PersonSwitchHandler ? This can be achieved using bind.

{this.personswaitchHandler.bind(this,'Kali')}

And this is the method to do so. This is used to refer anything in the same class. Bind takes two parameters this and “Parameter value”.

**Two way binding:**

So far we have hard coded the parameter which are passed to the eventhandler function. We want to take input from the user and then change the state using it. So for that we have to listen to an event and after getting that event we need to pass that to our eventHandler function.

In person component we are listening to onchange event. And passing it to nameChangeHandler.

Value=props.name is used to show the initial state. That shows two way binding as we are sending data to component and also reciving data from it.

A good read:

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

Working with Lists and Conditionals

MAKING OUR COMPONETS MORE DYNAMIC.

How to render lists and conditional rendering.

1. Conditional statements can be used to render properly the things you want and things you don’t.We can’t use if else block here for reasons still don’t known to me but you can employ tieranary logic to show some divs conditionally.
2. {this.state.showPerson ?  //Note how we have enclose it in {} because it

Not simple html rather javascript.

1. <div>
2. <Person
3. name ={this.state.persons[1].name}
4. age = {this.state.persons[1].age}>
5. This is an alter ego
6. </Person>
7. <Person
8. name ={this.state.persons[2].name}
9. age = {this.state.persons[2].age}
10. >
11. </Person>
12. </div>: null} //Else display null.
13. As you can see this can be cumbersome. How can we improve it ? How can we use if else block. The simple fact that react render anything that we return and we can write javascript code outside return statement comes to our use.



Notice how morePersons is given a value only when state turns out to be true. More persons will always be displayed but its value depends on the state now.

**Outputting lists:**

1. So far we have been hardcoding each person in component using array like indices.
2. <Person
3. name ={this.state.persons[1].name}
4. age = {this.state.persons[1].age}>
5. This is an alter ego
6. </Person>

There can be a number of entries in the state and it is not feasible to hard code all of them. Instead we will take help of the map function in javascript.

.map function works on an array elements and map them to something.That something is given as a parameter to it.

So it takes an array,process each value, and then returns an array.

**Example:** var numbers = [4, 9, 16, 25];  
var x = numbers.map(Math.sqrt)

Here a square root of each number is taken. And a new array with new values is generated.

In our case we want to map persons array to persons component. That is for each entry in person array we want a component for it.

And the way to do it:

{this.state.persons.map(Ele =>{ //We can name it anything instead of Ele.

            return <Person

              name={Ele.name} //For each array element that we take what we

//return is a component.

              age={Ele.age}

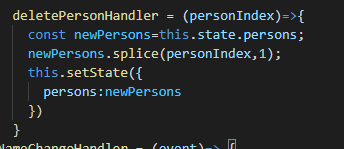
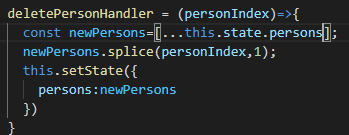
            />

          })

          }

**Lists and state:**

**4.** Arrays and objects in javascript are of refernce type. That means while manipulating the state always copy it instead of manipulating it directly. Look at the code below:

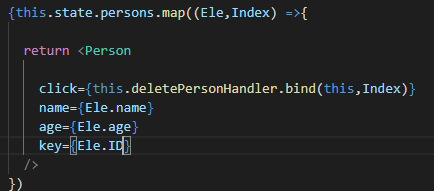
Here newPersons is a pointer pointing to a state ||| Here newPersons is pointing to new memory

Variable. which stores persons array using spread operator

**List and Keys:**

5. While mapping lists in React it expects a key property to be returned. This is some unique key by which react can identify data.

Example:



**Flexible Lists:**

1. Sometimes you would like to listen to events from the componets and pass them to your event handler functions.

This is the way: //NameChange= {(event)=>this.NameChangeHandler(event,Index)}

Using an inline anonymous function.

The **inline** anonymous **arrow function**

is used to **pause** the **immediate** invocation

So now,  a **new** problem is introduced ...

The **inline** anonymous **arrow function** breaks the

default **passing in** of the event **object**.

The event **object** is **not** (in scope) available (as an **argument**).

To make event **object** (in scope) available,  it is first **passed** **into**

the **inline** anonymous **arrow function** as a **parameter**.

event => this.nameChangedHandler(event, person.id)

(above) Meaning the **first** (left) use of event **object**.

Now,  event object **is** (in scope) available to be passed in

as nameChangedHandler's **argument**. (event on the right).

Remember previously we were just passing the refernce of the event handler method because we don’t explicitly need to send event. Other approach here is to use bind obviously like:

NameChange= {this.NameChangeHandler.bind(this,Index)}

And then in event Handler the second parameter by defualt would have been **event** object

NameChangeHandler = (Index,event)=> {

# Styling react Components

Styling react components:

1. Inline style properties can be altered using state variables. For example we can change some property inside if block.

**Dynamic ClassName**

We can attach css classes dynamically to code by again using the fact that eveything is javascript. let classes=[];

    if(this.state.persons.length<=2)

    {

      classes.push('red');

    }

    if(this.state.persons.length<=1)

    {

      classes.push('bold');

    }

Red and bold are two css classes.

<p className={classes.join(' ')}

As className expect a string that’s why we use join method.

**What are Pseudo-classes?**

A pseudo-class is used to define a special state of an element.

For example, it can be used to:

* Style an element when a user mouses over it
* Style visited and unvisited links differently

**Style an element when it gets focus**

1. This pseudo classes can not be used in the in line styles. We can use them in the global css files but then it will make that property get applied to every instance of that component.
2. In order to solve this problem,we can dowload an external package:radium. This lets us use css psedo classes by just wrapping the classes and functional components that we are exporting.

**Example:**

export default Radium(App);

To use the propertise:

  ':hover':{

        backgroundColor:'lightgreen',

        color:'black'

      }

We need to give the property as a string.

**CSS3 Introduced Media Queries**

Media queries in CSS3 extended the CSS2 media types idea: Instead of looking for a type of device, they look at the capability of the device.

Media queries can be used to check many things, such as:

* width and height of the viewport
* width and height of the device
* orientation (is the tablet/phone in landscape or portrait mode?)
* resolution

Using media queries are a popular technique for delivering a tailored style sheet to desktops, laptops, tablets, and mobile phones (such as iPhone and Android phones).

Syntax Example:

@media screen and (min-width: 480px) {  
  body {  
    background-color: lightgreen;  
  }  
}

1. Having Radium gives us the power to even do that.

For that we will be writing this in Styles object:

'@media(min-width:500px)':{

            width:'450px',

            color:'orange',

            backgroundColor:'salmon'

        }

Also we would need to import a wrapper from radium:

import Radium ,{StyleRoot} from 'radium';

StyleRoot is required to wrap the whole div which we are returning in render function.

**Styles Package:**

Styled-components lets you write actual CSS in your JavaScript. This means you can use all the features of CSS you use and love, including (but by far not limited to) media queries, all pseudo-selectors, nesting, etc. The functions are given as tagged literals.

This styled component has an object for all the html components that you can create.

For ex: styled.button,styled.div,styled.p etc

**All these medthods always return a react componenet**.

And inside quotation marks we write normal css and not javascript.So things like backgroundColor will become background-color.

const StyledButton= styled.button`

      background-color:green;

      color:white;

      cursor:pointer;

      padding:16px;

      border: 1px solid blue;

       font:inherit;

      &:hover{

        background-color:lightgreen;

        color:black;

      }

`;

Use:

 <StyledButton onClick= {this.ShowPersonHandler}>

        Show All persons

          </StyledButton>

To do dynamic styling:

background-color:${props=> props.alt ? 'red' : 'green'};

Props will be send aS:

<StyledButton alt={this.state.showPerson} onClick= {this.ShowPersonHandler}>

While all this is fine and working well. There is one more thing that we need to study about. That is a CSS modules.

We can change the webpack file and get some extra ordinary features.

That is:

We can import css classes as objects and can dynamically bind them to our components.

What are CSS modules:

*CSS files in which all class names and animation names are scoped locally by default.*

We know how CSS and HTML works by binding css classes to html elements.



*[the classes] are dynamically generated, unique, and mapped to the correct styles.*

**Why should we use CSS Modules?**

With CSS Modules, it’s a guarantee that all the styles for a single component:

1. Live in one place
2. Only apply to that component and nothing else

Plus, any component can have a true dependency, like:

import buttons from "./buttons.css";

import padding from "./padding.css";

element.innerHTML = `<div class="${buttons.red} ${padding.large}">`;

**This approach is designed to fix the problem of the global scope in CSS.**

**In order to use CSS modules we would need to run npm run eject.**

**What doe it do ?**

**Answer:**

*create-react-app encapsulates all of the npm modules it is using internally, so that your package.json will be very clean and simple without you having to worry about it.*

*However, if you want to start doing more complex things and installing modules that may interact with modules create-react-app is using under the hood, those new modules need to know what is available and not, meaning you need to have create-react-app un-abstract them.*

*That, in essence, is what react-scripts eject does. It will stop hiding what it's got installed under the hood and instead eject those things into your project's package.json for everyone to see.*

Now this is all require if we are using react scripts version 1 or lower. For 2 or above we can simlply add .module in css file name and we can use it anywhere we want.

For versions above just save your css file as: App.module.css.

**CSS Modules** are a relatively new concept (you can dive super-deep into them here: <https://github.com/css-modules/css-modules>). With CSS modules, you can write normal CSS code and make sure, that it only applies to a given component.

It's not using magic for that, instead it'll simply **automatically generate unique CSS class names** for you. And by importing a JS object and assigning classes from there, you use these dynamically generated, unique names. So the imported JS object simply exposes some properties which hold the generated CSS class names as values.

**Example:**

**In Post.css File**

1. .Post {
2. color: red;
3. }

**In Post Component File**

1. import classes from './Post.css';
3. const post = () => (
4. <div className={classes.Post}>...</div>
5. );

Here, classes.Post  refers to an automatically generated Post  property on the imported classes  object. That property will in the end simply hold a value like Post\_\_Post\_\_ah5\_1 .

So your .Post  class was automatically transformed to a different class (Post\_\_Post\_\_ah5\_1 ) which is unique across the application. You also can't use it accidentally in other components because you don't know the generated string! You can only access it through the classes  object. And if you import the CSS file (in the same way) in another component, the classes  object there will hold a Post  property which yields a **different** (!) CSS class name. Hence it's scoped to a given component.

By the way, if you somehow also want to define a global (i.e. un-transformed) CSS class in such a .css  file, you can prefix the selector with :global .

**Example:**

:global .Post { ... }

Now you can use className="Post"  anywhere in your app and receive that styling.

# **Debugging React Apps**

React developer tools can be installed in chrome to get the state and props of various components.

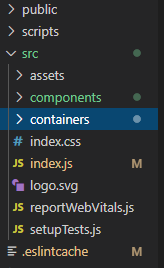
Apart from that react 16 comes up with a new feature called react error boundry tool. This is a higher component which lets us wrap our code,which might be faulty,inside a error boundry div,so that a proper message can be displayed.

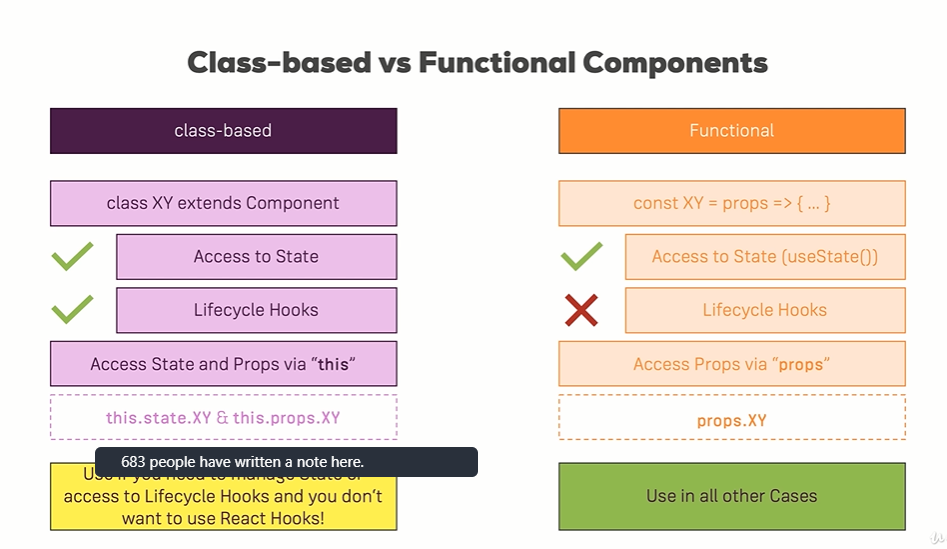
To read about error boundry tools:

<https://reactjs.org/docs/error-boundaries.html>

# Deeper into React Components

* 1. We should have clear project structure. Components folder to have all thefunctional components. Container s having all class components.
  2. Also make sure your app.js (class component)’s render function is clean meaning it shouldn’t really have many things. Just an object or few and divs.



* 1. 

We can note that after react 16 there is not much difference between class based components and functional components.To use props in app based components **we need to use this.props.variable**.

<App appTitle='This is person manager'/>

Sending props to app.js from index.js.

Sending it to cockpit.js from app.js

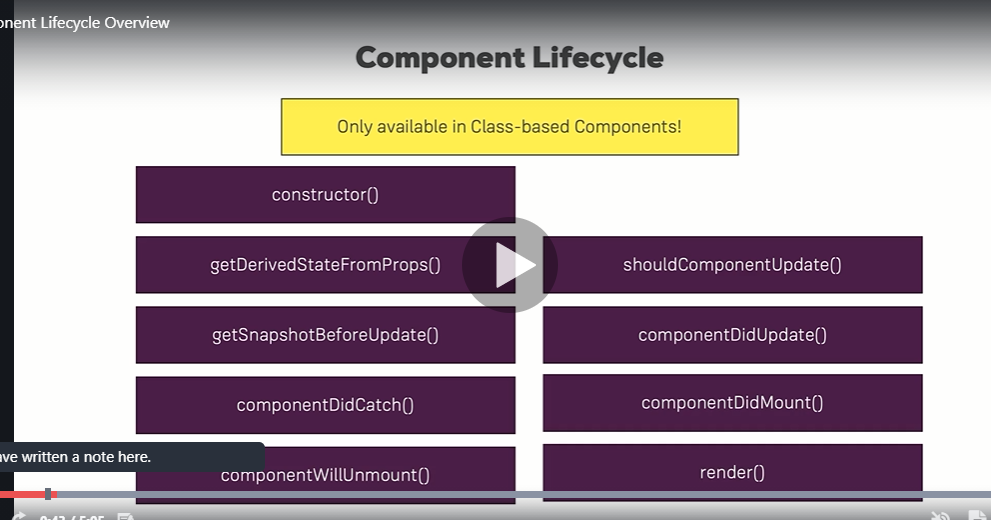
 <Cockpit

      title={this.props.appTitle}

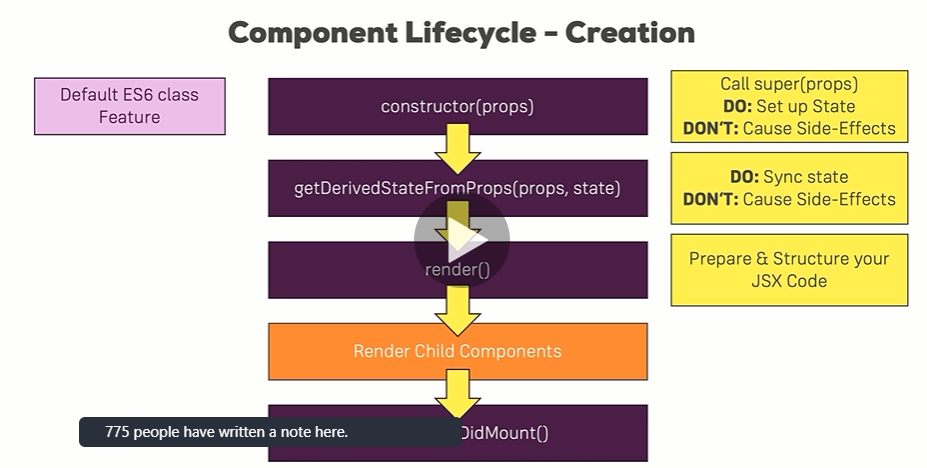
* 1. Class component lifecycle:

Only available in react 16.0.

A number of methods are available by class components which can be used by us to carry out a number of functions. These methods are:



These get called in an sequence:



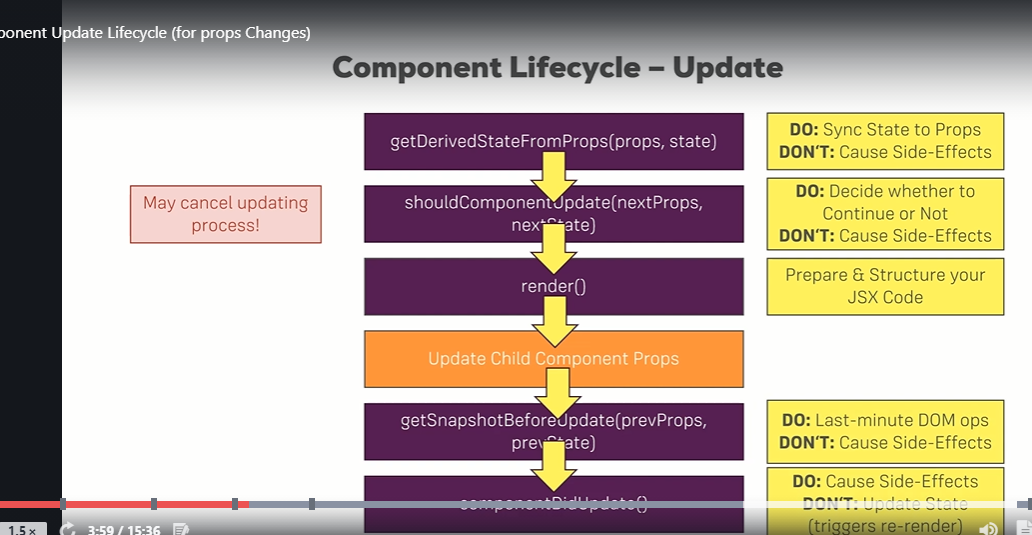
**Cause side effects mean you should not make a http request in that method.**

* 1. Call the constructor. Make sure you call the super(props) because this will make sure the component of the class that you are extending gets called.
* class App extends Component {
* constructor(props)
* {
* super(props);
* console.log('[App.js] cosnstructor is fired up!');
* /\* this.state=
* \*/
* }

Constructor most of the times is used to set the state. Remember we can’t use setState here as there is no state to be merged into.

1. Component lifecycle-update.

Just like how a number of methods get fired up when a component is created just like this a number of methods get fired up when a component is updated.



Last one is Componentdidupdate().

One of the lifecycle hook-> **shouldComponentUpdate(nextProps,nextState)** is very imp as it tells reacts weather our react app will update or not. If we return false in it then react will not update the dom even if there is state or prop change to trigger it.

These Lifecycle hooks are there in class components but what about functional components ?

As we can manange state in functional compnents now we should have an equivalent lifecycle hooks in them as well.

Functional Components now come with->

useEffect(()=>{

      console.log('[cockpit.js] redering');

    });

This react hook alone contains all the functionality of class based life cycle hooks. This is called everytime the dom renders.

1. **Cotrolling the behaviour of Use effect.**

Right now UseEffect gets called everytime after we render.

 setTimeout(()=>{

        alert('Data Saved');

      },1000);

If we put this inside useffect then we will get the alert after every time dom re-render.

Now what if we want to see this alert only when person component changes. In that case we can **give an array of objects to UseEffect and tell the function to re-run only when the given list of components change**.

Like this:

useEffect(()=>{

      setTimeout(()=>{

        alert('Data Saved');

      },1000);

    },[props.persons]);

This will make useEffect to re-run every time persons change.

What if we want it to run only at start and never again ?

We can give an empty array. This will mean there are no dependencies and it will never re-run.

useEffect(()=>{

      setTimeout(()=>{

        alert('Data Saved');

      },1000);

    },[]);

1. Cleanup work-> Doing some clean up work in react.

After render cycle we might one to destroy connection to some live server or anything. For that we might need a place where we can do this cleanup work.

In class based components we can write->

componentWillUnmount(){

      console.log('[Persons.js] Cleanup work in Appjs');

    }

In hooks->

We can return an anon function instead.

  useEffect(()=>{

      setTimeout(()=>{

        alert('Data Saved');

      },1000);

      return ()=>{

        console.log('[Cockpit.js] cleanup rendering');

      };

    },[props.persons]);

This will make return anon function run everytime perosn component is changed. If we give empty array as an argument then it will run *only when the component is rendered for the last time.*

1. Shouldcomponentupdate and React.memo() can be used to optimize our components so that they render only when they are changed.

We can use these two to tell react to re render the current compoment only when props change.

export default React.memo(Cockpit);

What this does is memoize the component. And check if prevProps are different than current props. If yes then use the previous memoized version other wise re render.

We can also have customm comparator in React.memo like this->

function moviePropsAreEqual(prevMovie, nextMovie) {

return prevMovie.title === nextMovie.title

&& prevMovie.releaseDate === nextMovie.releaseDate;

}

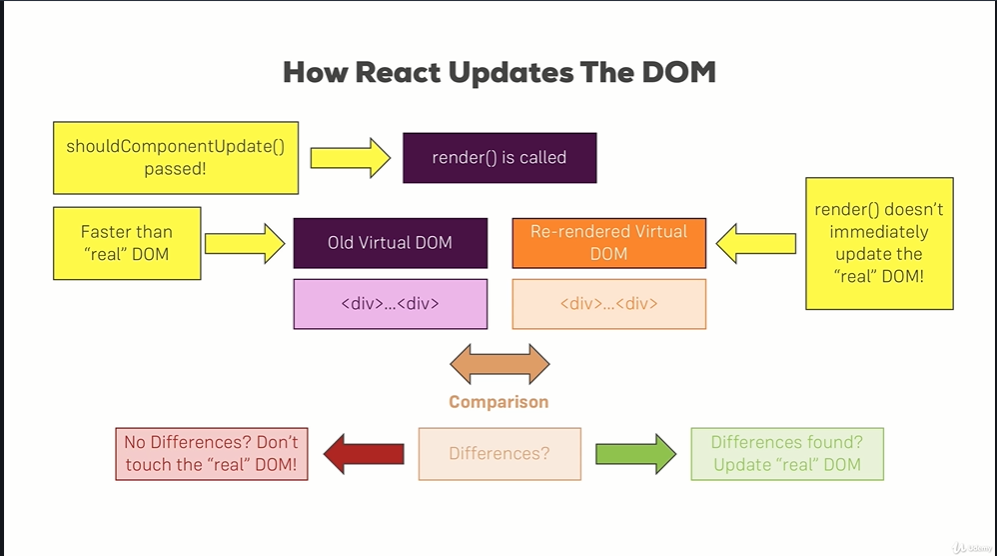
const MemoizedMovie2 = React.memo(Movie, moviePropsAreEqual);

1. When should we optimize ?

It is not advisable to use React memo and shouldcomponentupdate all the time. As it is an extra check. If our component always update when its parent compnent update then we should avoid these checks.

1. PureComponent:

This is just like componet but it implements shouldcomponentupdate and check for every single prop check. So if we have large number of props we want to check all of them we can just simply inherit PureComponent class.

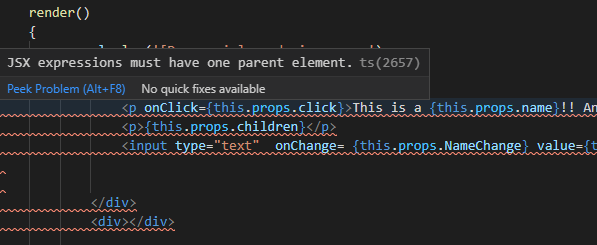
1. 

This means React never really touch the real DOM as accesing it is very slow. So it just makes comparision and check if there is any change. And then makethe necessary changes but never really changes the whole DOM.

1. Rendering adjacent JSX elements.

We all know it is not allowed to return more than one root element in react. All the html we are returing must be enclosed in a single div.

**For example:**



But what we can actually do is return an array.

See this: return (

                [<p onClick={this.props.click}>This is a {this.props.name}!! And he is {this.props.age} years old</p>,

                <p>{this.props.children}</p>,

                <input type="text"  onChange= {this.props.NameChange} value={this.props.name} />]

        )

But react does want each array element to have a unique key.

 [<p key="1A" onClick={this.props.click}>This is a {this.props.name}!! And he is {this.props.age} years old</p>,

                <p key="1B">{this.props.children}</p>,

                <input key="1C" type="text"  onChange= {this.props.NameChange} value={this.props.name} />]

Like this. Key is necesaary for react to identify each array element in a unique way.

But that is not the only way to do this. There lies one more->more subtle and structured way.

We can create a Higher Order Component and wrap everything inside it.

const Auxillary = props=> props.children

export default Auxillary;

Here Auxillary is doing nothing but taking the props and rendering them.

<Auxillary>

                    <p  onClick={this.props.click}>This is a {this.props.name}!! And he is {this.props.age} years old</p>

                    <p>{this.props.children}</p>

                    <input  type="text"  onChange= {this.props.NameChange} value={this.props.name} />

                </Auxillary>

Now this is fine because even now we are making a single react.Ceateelement call.

1. Since React 16.2 react comes with a built in Auxillary HOC. It is called React Fragment. It does exactly the same work.

Just like this->

<React.Fragment>

                    <p  onClick={this.props.click}>This is a {this.props.name}!! And he is {this.props.age} years old</p>

                    <p>{this.props.children}</p>

                    <input  type="text"  onChange= {this.props.NameChange} value={this.props.name} />

                </React.Fragment>

1. Having classes in HOC.

import React from 'react';

const WithClass = props=>

(

    <div className={props.classes}> {props.children}</div>

);

export default WithClass;

This higher order component does the same work as a div but is cutsom made. And we caan do a lot of things here like error handling etc.

1. Another form OF HOC

One other way of using HOC is making a simple javascipt function which returns a functional component.

const withClass = (WrappedComponent,className)=> {

    return porps=> (

        <div className={className}>

            <WrappedComponent/>

        </div>

    )

}

Here we get two arguments but we can take as many as we would like.

And this how we wrap the main App class->

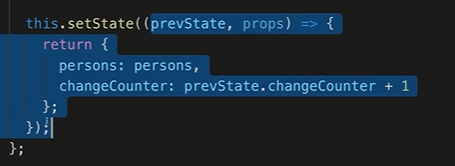
export default withClass(App,classes.App);

1. Paassing unknown props.

In order to use props passed to HOC. We should use spread operator. As HOC does get all the props but we need to use the spread operator in order to make new key value pairs there.

1. Setting state correctly

SetState fo react doesn’t work instantly. Instead react call setState when the appropriate resources are available. This means some setState calls can be made before others. In order to tackle this we should use setState like this:



1. In order to use some of the props and operate some operation on them we can use Ref.

It is a special property provided by React in order work on porps.

 constructor(props)

    {

        super(props);

        this.inputElementRef=React.createRef();

    }

    componentDidMount(){

        this.inputElementRef.current.focus();

    }

//And creating ref inside the input element.

 <input

                     ref={this.inputElementRef}

.current is used in order to access the current reference.

1. Using ref in functional based components.

For this we first need to import useRef->

import React, {useEffect,useRef} from  'react';

Then we make a js object and make it ref type->

const togglebtnref= useRef(null);

And simply use ref inside button->

 <button

             ref={togglebtnref}

             className= {btnClass}

            onClick= {props.ShowPersonHandler}>

            Show All persons

            </button>

Now where we should use **togglebtnref ?**

It should only be called inside useEffect because useEffect runs after every render cycle. We don’t want to use it outside as it will be called even before a ref is made.

1. Props chain problem.

Suppose you have a state a varaible in **A.**

**From A** you are sending props to B

**Then to C**.

**And then to D**.

*We only use the prop in D*.

Doesn’t it make sense then to find a way by which we can skip sending props to B and C and can somehow directly send it to D.

In our project we have passed authenticed from *App.js* to *Persons.js* and using it in *Person.js*.

1. Context API->

This gives us a globally created object/string/array or whatever you want which can be used anywhere you want.

import React from 'react';

const authContext= React.createContext({

    authenticated:false,

    login:()=>{}

});

export default authContext;

We can initiailize it with any value we want. And after that we need to wrap all the components where we want to use this.

Like this.

      <Auxillary>

        <AuthContext.Provider>

        <Cockpit

      title={this.props.appTitle}

        showPersons={this.state.showPerson}

        personslength={this.state.persons.length}

        ShowPersonHandler={this.ShowPersonHandler}

        login={this.LoginBtnHandler}

      />

    {morePersons}

        </AuthContext.Provider>

    </Auxillary>

Now what we set as initial value doesn’t really matter because->

<AuthContext.Provider value ={{authenticated:this.state.Authenticated ,login:this.LoginBtnHandler}}>

We any way set the value from where we are providing like this.

Now wherever we want to use it> We will have

<AuthContext.Consumer>

</AuthContext.Consumer>

And what it return is a function which get context as props.

{(context)=> {context.authenticated ?

 <p>User is Authenticated</p> :<p>Please Login</p>}}

1. This is one way of using context and can be really strict. To use a varaible any where it would require me to write <AuthContext.Consumer> everytime.

And this can be very strict. React 16.0 gives us a new feature, namely contextType.

We can write->

 static contextType= AuthContext;

And there is no other way around it. We should use contextTypename only and declare it static. Doing this know we can use all the values declared in context anywhere without any ristriction.

Using: *this.context.property\_name*

{this.context.authenticated ? <p>User is Authenticated</p> :<p>Please Login</p>}

But this is only available in class based commponents. What about functional components ?

There we can use the the hook useContext() and pass our context component as paramter.