REACT

CONCEPTS

SETTING UP REACT PROJECT

Setting up the react project • npx create-react-app my-app will install react, react dom, react scripts with cra-template • You can now edit the files in src directory • npm start will run the app in dev. mode	npx create-react-app my-app cd my-app npm start

REACT'S RENDER() METHOD AND CREATEROOT() METHOD

ReactDom.render()

- Render's html elements based on id of specified element
- Takes 3 parameter's
  1. What to show (html element)
  2. Where to show (class/id of element)
  3. optional callback to tell when render function completed

```
// import react and react-dom
import React from "react";
import ReactDOM from "react-dom";
// ReactDOM.render() method
ReactDOM.render(
   <div>
      <h1>Hello world</h1>
      <p>Paragraph</p>
  </div>,
document.getElementById("root")
);
```

createRoot().render()

- ReactDom.render() is deprecated as of react 18, so the following method can be used to render elements based on id of element

```
import React from "react";
import {createRoot} from "react-dom/client";
// Root id element
const Root = document.getElementById("root");
// createRoot().render() method
createRoot(Root).render(
  <div>
    <h1>Hello world</h1>
  </div>
);
```

JSX JAVASCRIPT EXPRESSIONS

JavaScript expressions

- We can specify JS expressions in html inside of {}

```
const fname = "John";
const lname = "Holmes";
const number = 8;
// Specify JavaScript inside html
ReactDOM.render(
  <div>
     <h1>Hello {fname + " " + lname}!</h1>
     <p>Your lucky number is {4 * 2}</p>
 </div>,
document.getElementById("root")
);
```

JSX ATTRIBUTES AND INLINE CSS

JSX attributes
• Attributes should be in camelCase
	`<img className ="sample-img" src={img} />`
	`<h1 className="heading" contentEditable="true" spellCheck="false">`

JSX inline styling
• Inline css to be provided through JS objects
	// Method 1
	`<h1 style={{color:"red"}}>Hello World!</h1>`
	
	// Method 2
	`const styleOBJ = {`
	    `color:”Blue”,`
	    `fontSize: “20px”`
	    `border:  “1px solid black”`
	`}`
	`<h1 style={StyleOBJ}>Hello World!</h1>`
	
	// Changing styles is equivalent to changing value’s of object key’s
	`styleOBJ.color = “red”`

REACT FUNCTIONAL COMPONENTS

React Components

- Components can be declared with the .js file or in an external file with .jsx extension

// Declaring components internally

```
function Heading(){
  return <h1>My Favourite Foods</h1>;
}

ReactDOM.render(<Heading />, document.getElementById("root"));
```

// Declaring components in external .jsx file

Components/Heading.jsx

```
import React from "react";

export default function Heading() {
  return <h1>My Favourite Foods</h1>
}
```

Components/App.jsx

```
import React from "react";
import Heading from "./Heading";
import FoodList from "./list";

export default function App() {
  return (<div>
      <Heading />
      <FoodList />
     </div>);}
```

Index.js

```
import App from "./Components/App";
ReactDOM.render(<App />, document.getElementById("root"));
```

REACT CLASS COMPONENTS

React Class Components

- We can also declare components in classes in react

// Simple Example

```
import React from "react";

class App extends React.Component {
  render() {
    return <h1>Hello</h1>;
  }
}

export default App;
```

REACT FRAGMENTS

React Fragments

- We can specify <div> as <React.Fragment> or <> to simplify the DOM tree

```
function App() {
  return (
    <div>
      <Child />
    </div>
  );
}
```

```
function Child() {
  return (
    <div>
      <h1>Child component</h1>
    </div>
  );
}
```

// Html upon inspecting the page

```
<div id="root">
  <div>
    <div>
      <h1>Child component</hl>
    </div>
  </div>
</div>
```

```
function App() {
  return (
<React.Fragment>
      <Child />
</React.Fragment>
  );
}
```

```
function Child() {
  return (
    <React.Fragment>
      <h1>Child component</h1>
    </React.Fragment>
  );
}
```

OR

```
function App() {
  return (
<>
   <Child />
</>
  );
}
```

OR

```
function Child() {
  return (
    <>
      <h1>Child component</h1>
    </>
  );
}
```

// Html upon inspecting the page

```
<div id="root">
    <h1>Child component</h1>
</div>
```

REACT PROPS FOR FUNCTIONAL COMPONENTS

React props for components

- We can declare properties for components in react and access those properties inside the component using **props.property-name**

```
// Card component
function Card(props) {
 //Returns JS object of properties passed while using the component
 console.log(props);
 return (
  <div>
   <h2>{props.name}</h2>
   <img src={props.src} />
  </div>
);}

// Using card component by specifying name and src properties
ReactDOM.render(
 <div>
  <Card
   name="John"
   src="image-link"
  />
</div>
```

REACT PROPS FOR CLASS COMPONENTS

React props for components
• We can declare properties for components in react and access those properties inside the component using **props.property-name**
	// Example with props
	App.js
	`class App extends React.Component {`
	`  render() {`
	`    return <h1>{this.props.type} component</h1>`
	`  }`
	`}`
	
	Index.js
	`<App type="Class" />`

REACT CHILDREN

React children

- We can specify elements/children inside components and access those elements inside the component using **props.children**

App.js

```
import CTA from "./CTA"
function App() {
  return (
    <div>
      <CTA>
        <h1>This is an important CTA</h1>
      </CTA>
    </div>
)}
```

CTA.js

```
export default function CTA(props) {
   return (
      <div className="border">
         {props.children}
      </div>
)}
```

MAPPING DATA TO COMPONENTS

Mapping data to components

- We can use the map() keyword to create a number of same components based on the objects present in an array

```
// Array of objects
contacts = [obj1, obj2, obj3]

// Card function which renders a Card based on it's properties
function createCard(contact) {
  return (
    <Card
      key={contact.id}
      id={contact.id}
      name={contact.name}
      img={contact.imgURL}
    />
);}

// Contacts.map(createCard) will create number of card components based on the
number objects in array (here 3, so it will create 3 cards for objects in contacts array
function App() {
  return (
    <div>
      {contacts.map(createCard)}
    </div>
);}
```

CONDITIONAL RENDERING

Conditional rendering

- We can choose to render certain components when certain conditions are meet by using :-

a. Function with checks those conditions and returns a value

b. Ternary operator :-
Condition? Do if true : Do if false

c. AND (&&) operator :-
CONDITION && EXPRESSION
// Expression executes
true && EXPRESSION
// Expression skipped
false && EXPRESSION

```
// Conditional rendering using function
var isLoggedIn = 1;
const renderConditionally = () => {
  if (isLoggedIn) {
    return <Homepage />;
  } else {
    return <Login />;
}};

function App() {
  return <div>{renderConditionally()}</div>;
}
```

```
// Conditional rendering using ternary operator
function App() {
  return <div>{isLoggedIn? <h1>Hello</h1> : <Login />}</div>;
}

const currentTime = new Date().getHours();
currentTime>12? <h1>Why are you still working?</h1> : null

<button type="submit">{props.isRegistered ? "Login" : "Register"}</button>
```

```
// Conditional rendering using AND (&&)
currentTime < 12 && <h1>Good Morning</h1>
!props.isRegistered && <input type="password" placeholder="Confirm Password" />
```

REACT HOOKS & USESTATE()

React hooks

- Hooks are functions that hook into the state of app & read or modify it

useState()

- **let [start-state, function]= useState(start-state)**
- start-state -> any value passed inside useState which indicates the initial state
- function -> function which can be used to modify the value of start-state
- If you log useState() you can see an array with value passed in useState() and a function() inside an array

```
import React, { useState } from "react";

function App() {
 // count is the value passed inside useState()
 // setCount can be used to update value of count
 const [count, setCount] = useState(0);

 // setCount used to update the count
 const increase = () => {
  setCount(count + 1);
 };

 // Call increase() every second and increase count value dynamically
 setInterval(increase, 1000);

// button which on clicking update's value of count
 return (
  <div>
   <h1>{count}</h1>
   <button onClick={increase}>+</button>
  </div>
);}
```

EVENT HANDLING USING REACT HOOKS

React hooks

- React Hooks can also be used to perform event handling

- The following code shows how to change color of button when we mouse over and mouse out over it using useState()→

- In the following example active can take true or false values
- setActive() can be used to modify the value of active
- If we mouseover or mouseout the button we call the Mouse() function which calls setActive() to change value of active
- Now we can use ternary operator to check value of active and if value is true we will change background color to black else white

```
import React, { useState } from "react";

function App() {
  const [active, setActive] = useState(false);

  function Mouse() {
    setActive(!active);
  };

  return (
    <div className="container">
      <button
        onMouseOver={Mouse}
        onMouseOut={Mouse}
        style={{ backgroundColor: active ? "black" : "white" }}
      >
       Submit
      </button>
    </div>
  );
}
```

FORMS WITH REACT HOOKS

React forms

- The following code shows the use of useState() in react forms→
- In the following example we will display the value entered in input box as heading
- We declare 2 hooks one for keeping track of change made in input box and other for handling change in heading
- Upon typing something in input box the handleChange() will be executed which will access the value typed in input box using event.target.value & save it under name using setName()
- Now When the submit button is clicked the handleChange() will be triggered which will change the heading to the value of name
- The event.preventDefault() will prevent the default behavior of the form

```
import React, { useState } from "react";

function App() {
  const [name, setName] = useState("");
  const [headingText, setHeading] = useState("");

  const handleChange = event => setName(event.target.value);

  const handleClick = event =>{
     setHeading(name);
     event.preventDefault();
}

return (
    <div>
      <h1>Hello {headingText}</h1>
      <form onSubmit={handleClick}>
      <input onChange={handleChange}  type="text“ value={name}  />
      <button type="submit">Submit</button>
      </form>
    </div>
);}
```

CHANGING COMPLEX STATE WITH REACT HOOKS

Complex state with react hooks

- The following code shows the use of useState() in complex state→
- In the following example we will display the fname & lname entered in 2 input boxes as heading
- We declare 1 hook and pass an object having fname & lname keys in useState() for keeping track of change made in input box
- Upon typing something in input box the handleChange() will be executed which will access the value typed in input box & name of input using event.target object
- Now we will check if the input was coming from box lname or fname & accordingly pass value to it
- The prevValue is the previous value that react remembers of state

```
const [fullName, setFullName] = useState({fName: "", lName: ""});

const handleChange = (event) => {
          const { value, name } = event.target;

        setFullName((prevValue) => {
              if (name === "fName") {
                  return {fName: value, lName: prevValue.lName};
              } else if (name === "lName") {
                  return {fName: prevValue.fName,lName: value};
          }
     });
     or
     setFullName(prevValue => ({...prevValue, [name]: value }));
};

return (
   <div> <h1> Hello {fullName.fName} {fullName.lName}</h1>
     <form>
     <input onChange={handleChange} name="fName" value={fullName.fName} />
     <input onChange={handleChange} name="lName" value={fullName.lName} />
     <button>Submit</button>
     </form> </div>
);
```

REACT MANAGING STATES BETWEEN COMPONENTS

Managing states between 2 or more components

Items = [item1, item2, item3]

- In the following example the list items are rendered from TodoItem.jsx and input fields from inputArea.jsx and items array to store the list items are present in App.jsx
- Suppose we want to delete a list item from TodoItem.jsx, then the change must reflect in items array present in App.jsx
- For this to happen we declare a deleteItem function in App.jsx, then we can pass this function as a prop in </TodoItem /> which can be accessed inside TodoItem.jsx to delete item
- To add an item in array we pass the addItem function as prop to <inputArea /> which can be accessed inside inputArea.jsx to add items in items array

App.jsx

```
const [items, setItems] = useState([]);
function addItem(inputText) {
   setItems((prevItems) => {
         return [...prevItems, inputText];
});}
function deleteItem(id) {
  setItems((prevItems) => {
    return prevItems.filter((item, index) => {
      return index !== id;
}); }); }
<InputArea onAdd={addItem} />
<ul> {items.map((todoItem, index) => (
    <TodoItem key={index} id={index} value={todoItem} onChecked={deleteItem}/>
))} </ul>
```

TodoItem.jsx

```
<li onClick={() => props.onChecked(props.id)}>
   {props.value}
</li>
```

inputArea.jsx

```
const [inputText, setInputText] = useState("");
const handleChange = event=> setInputText(event.target.value);
<input onChange={handleChange} />
<button onClick={()=>{
                props.onAdd(inputText);
                setInputText("");}} >
Add</button>
```

REACT MANAGING STATES BETWEEN COMPONENTS

Managing states between 2 or more components

Notes = [obj, obj, obj]

- This example is similar to previous, except it stores data in objects in array

CreateArea.jsx

```
const [inputText, setInputText] = useState({ title: "", content: "" });

function handleChange(event) {
  const { name, value } = event.target;
  setInputText((prevValue) => {
    return {
      ...prevValue,
      [name]: value
};});;}

<form onSubmit={(event) => {
  props.onAdd(inputText);
  setInputText({ title: "", content: "" });
  event.preventDefault();
}}>

<input onChange={handleChange} name="title"
value={inputText.title}/>
<textarea onChange={handleChange} name="content"
value={inputText.content} />
```

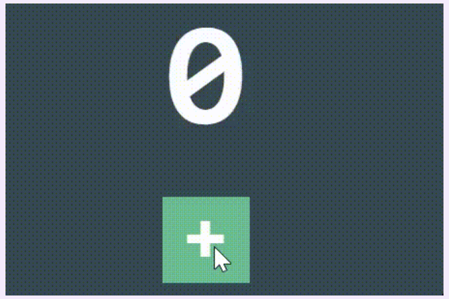
App.jsx

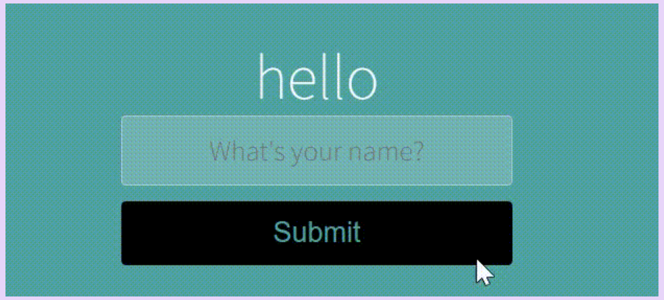
```
const [notes, setNotes] = useState([]);

function AddNote(inputText) {
  setNotes((prevNotes) => {
    return [...prevNotes, inputText];
});;}
function deleteNote(id) {
  setNotes((prevNotes) => {
    return prevNotes.filter((note, index) => {
       return index !== id;
});});;}

<CreateArea onAdd={AddNote} />
{notes.map((Noteitem, index) => (
 <Note key={index} id={index} title={Noteitem.title content={Noteitem.content}
onChecked={deleteNote}/>
))}
```

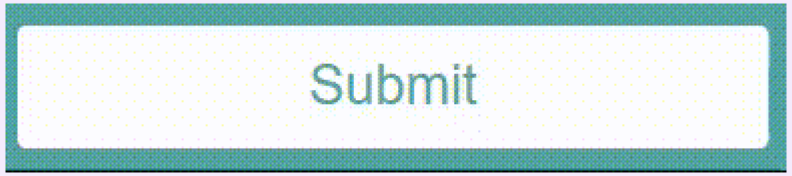
Note.jsx

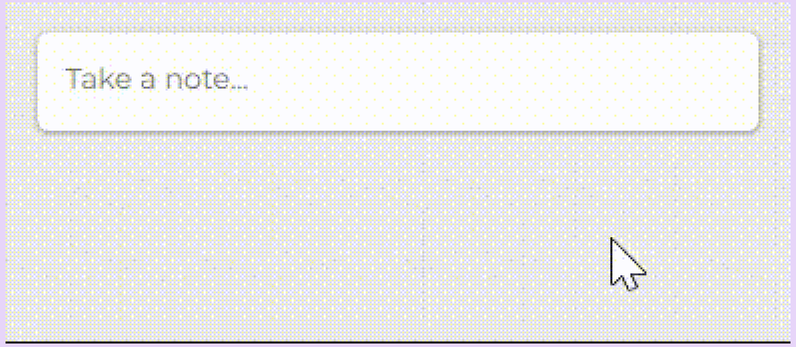
```
<h1>{props.title}</h1>
<p>{props.content}</p>
<button onClick={() => props.onChecked(props.id)}>DELETE</button>
```

REACT HOOKS SUMMARY

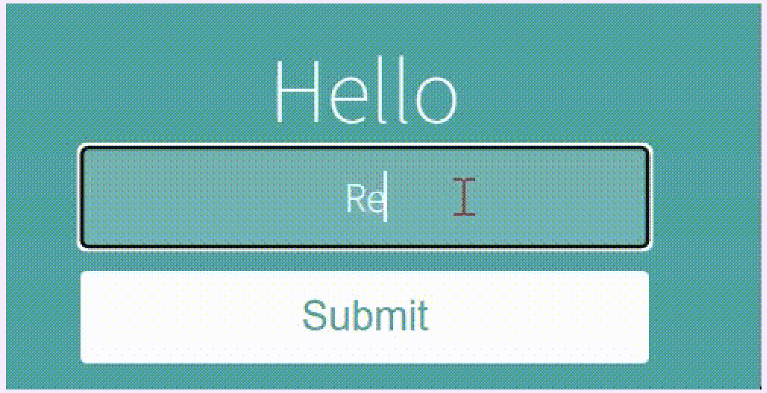
Change in state of count when button is clicked in counter app	const [count, setCount] = useState(0);
	
	const increase = prevValue => setCount(prevValue + 1);
	
	<h1>{count}</h1>
	<button onClick={increase}>+</button>
		

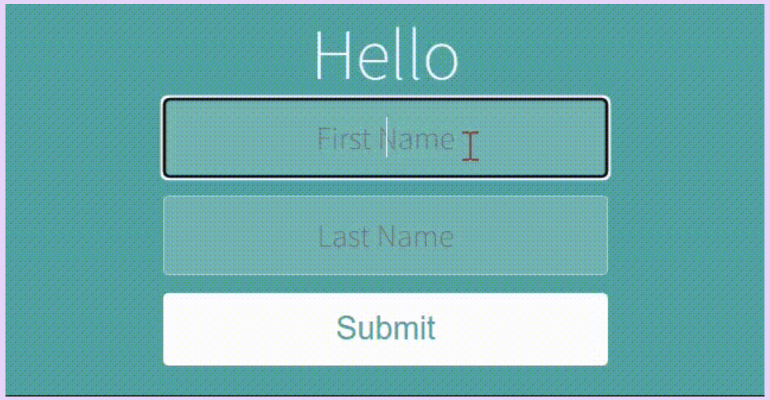
Change in heading when button clicked	const [headingText, setHeadingText] = useState("hello");
	
	const handleClick = () => setHeadingText("Submitted");
	
	<h1>{headingText}</h1>
	<button onClick={handleClick}>
		

Change in state of button when hovered upon	const [active, setActive] = useState(false);
	
	const Mouse = prevValue => setActive(!prevValue);
	
	<button onMouseOver={Mouse} onMouseOut={Mouse}
	style={{ backgroundColor: active ? "black" : "white" }}>
		

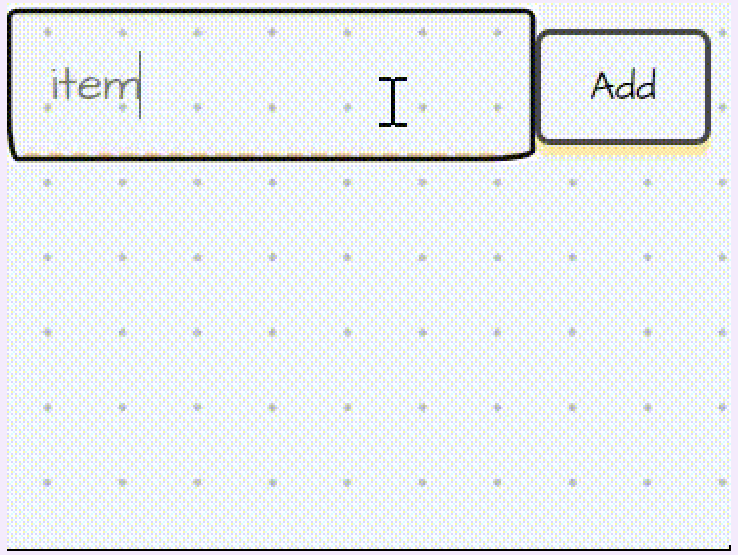
Change in state of textarea component when clicked upon
 	
* Zoom, Fab, AddIcon are components from material UI
	const [zoom, setZoom] = useState(false);
	
	const Click = () => setZoom(true);
	
	<input type={!zoom && "hidden"}/>
	<textarea rows={zoom ? 3 : 1} onClick={Click}/>
	<Zoom in={zoom}> <Fab> <AddIcon /></Fab> </Zoom>
		

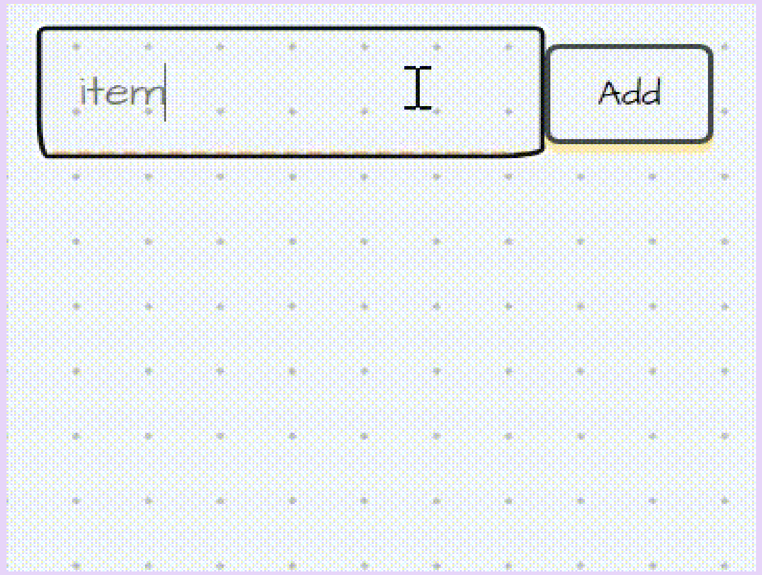
REACT HOOKS SUMMARY

Change in heading based on input from a input box when button clicked	`const [name, setName] = useState("");`
	`const [headingText, setHeading] = useState("");`
	
	`const handleChange = event => setName(event.target.value);`
	`const  handleClick = event => setHeading(name);`
	
	`<h1>Hello {headingText} </h1>`
	`<input onChange={handleChange} value={name} />`
	`<button onSubmit={handleClick} >Submit</button>`
		

Change in heading based on input's from multiple input box	`const [contact, setContact] = useState({ fName: "", lName: ""});`
	
	`function handleChange(event) {`
	`   const { name, value } = event.target;`
	`   setContact(prevValue => {`
	`   return {`
	`      ...prevValue,`
	`      [name]: value`
	`      };`
	`   });`
	`}`
	
	`<h1> Hello {contact.fName} {contact.lName} </h1>`
	`<input onChange={handleChange} name="fName" />`
	`<input onChange={handleChange} name="lName" />`
		

REACT HOOKS SUMMARY

Change in state of array when items are added	const [inputText, setInputText] = useState("");
	const [items, setItems] = useState([]);
	
	const handleChange = event=> setInputText(event.target.value);
	
	function addItem() {
	    setItems((prevItems) => {
	          return [...prevItems, inputText];
	});}
	
	<input onChange={handleChange} />
	<button onClick={addItem}>Add</button>
	<ul> {items.map((todoItem) => (<li>{todoItem}</li>))} </ul>
		

Change in state of array when item is clicked and get's deleted	function deleteItem(id) {
	  setItems((prevItems) => {
	    return prevItems.filter((item, index) => {
	      return index !== id;
	}); }); }
	
	<ul>
	    {items.map((todoItem, index) => (
	     <li key={index} id={index} onClick={() => deleteItem(index)}>
	        {todoItem} </li>
	     ))}
	</ul>
		

REACT STATES IN CLASS COMPONENTS

States in class Componets

- States are object in class components & is stored in state variable
- The value of state variable can be accessed using this
- The setState() function that comes with class component can be used to change value of state
- The setState() function can be accessed using this

```
import React from "react";

export default class App extends React.Component {
 // state variable initialized to it’s starting value
 state = {
  goOut: "Yes"
 };

// setState()  which returns a object to modify value of state
 toggleGoOut = () => {
  this.setState((prevState) => {
   return {
    goOut: prevState.goOut === "Yes" ? "No" : "Yes"
   };});};

 render() {
  return (
   <div className="state--value" onClick={this.toggleGoOut}>
    <h1>{this.state.goOut}</h1>
   </div>
  );}}
```

REACT STATES IN CLASS COMPONENTS USING CONSTRUCTOR METHOD

States in class Componets

- The state object is to be initialized using the constructor() method
- The super() function connects class component to prototype chain of React.Component
- You need to bind your class methods inside constructor functions, if you can't use arrow functions

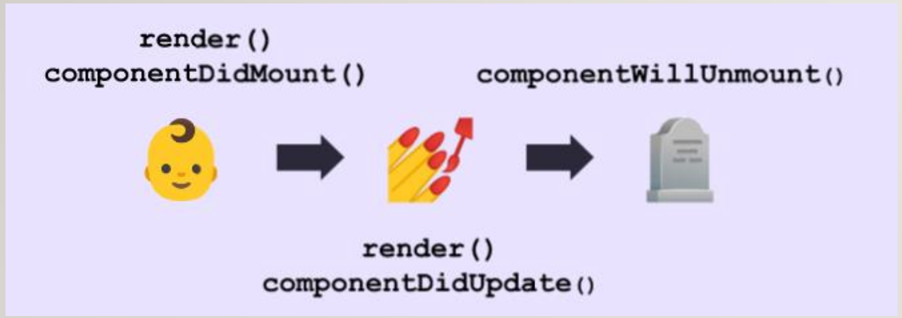
```
export default class App extends React.Component {
 constructor() {
   super();
  this.state = {
     goOut: "Yes"
  };
  this.toggleGoOut = this.toggleGoOut.bind(this);
}

toggleGoOut() {
  this.setState((prevState) => {
    return {
      goOut: prevState.goOut === "Yes" ? "No" : "Yes"
 };});}

 render() {
  return (
     <div className="state--value" onClick={this.toggleGoOut}>
       <h1>{this.state.goOut}</h1>
     </div>
  );}}
```

UPDATING COMPLEX STATES IN CLASS COMPONENTS

Complex States in class Componets

- You can skip using …prevContact in class components

```
export default class App extends React.Component {

  state = { firstName:"John",  lastName:"Doe", isFavorite:false};

  toggleFavorite = () => {
    this.setState((prevContact) => {
      return {
          isFavorite: !prevContact.isFavorite
      };});};

  render() {
    let starIcon = this.state.isFavorite ?"filled.png“ : "empty.png";
    return (
          <div>
              <img
                  src={`../images/${starIcon}`}
                  onClick={this.toggleFavorite}
              />
           <h2>
                  {this.state.firstName} {this.state.lastName}
           </h2>
          </div>
    );}}
```


render()
componentDidMount()
componentWillUnmount()
render()
componentDidUpdate()