**React**

**1. Create a Simple JSX Element**

const JSX = <h1>Hello JSX!</h1>;

**2. Create a Complex JSX Element**

Const JSX =

<div>

<h1>JSX</h1>

<p>Reacts own markup.</p>

<ul>

<li></li>

<li></li>

<li></li>

</ul>

</div>;

**3. Add Comments in JSX**

const JSX = (

<div>

{/\*This is a comment.\*/}

<h1>This is a block of JSX</h1>

<p>Here's a subtitle</p>

</div>

);

**4. Render HTML Elements to the DOM**

const JSX = (

<div>

<h1>Hello World</h1>

<p>Lets render this to the DOM</p>

</div>

);

// change code below this line

ReactDOM.render(JSX, document.getElementById("challenge-node"));

**5. Define an HTML Class in JSX**

const JSX = (

<div className="myDiv">

<h1>Add a class to this div</h1>

</div>

);

**6. Learn About Self-Closing JSX Tags**

const JSX = (

<div>

/\* remove comment and change code below this line

<h2>Welcome to React!</h2> <br />

<p>Be sure to close all tags!</p>

<hr />

remove comment and change code above this line \*/

</div>

);

\*\*\*removed the curly braces from the comment start and end.

**7. Create a Stateless Functional Component**

const MyComponent = function() {

// change code below this line

return (

<div>Text</div>

);

// change code above this line

}

**8. Create a React Component**

class MyComponent extends React.Component {

constructor(props) {

super(props);

}

render() {

// change code below this line

return (

<div>

<h1>Hello React!</h1>

</div>

);

// change code above this line

}

};

**9. Create a Component with Composition**

const ChildComponent = () => {

return (

<div>

<p>I am the child</p>

</div>

);

};

class ParentComponent extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h1>I am the parent</h1>

{ /\* change code below this line \*/ }

<ChildComponent />

{ /\* change code above this line \*/ }

</div>

);

}

};

Results:

**I am the parent**

I am the child

**10. Use React to Render Nested Components**

const TypesOfFruit = () => {

return (

<div>

<h2>Fruits:</h2>

<ul>

<li>Apples</li>

<li>Blueberries</li>

<li>Strawberries</li>

<li>Bananas</li>

</ul>

</div>

);

};

const Fruits = () => {

return (

<div>

{ /\* change code below this line \*/ }

<TypesOfFruit />

{ /\* change code above this line \*/ }

</div>

);

};

class TypesOfFood extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h1>Types of Food:</h1>

{ /\* change code below this line \*/ }

<Fruits />

{ /\* change code above this line \*/ }

</div>

);

}

};

Results:

# Types of Food:

## Fruits:

* Apples
* Blueberries
* Strawberries
* Bananas

**11. Compose React Components**

class Fruits extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h2>Fruits:</h2>

{ /\* change code below this line \*/ }

<NonCitrus />

<Citrus />

{ /\* change code above this line \*/ }

</div>

);

}

};

class TypesOfFood extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h1>Types of Food:</h1>

{ /\* change code below this line \*/ }

<Fruits />

{ /\* change code above this line \*/ }

<Vegetables />

</div>

);

}

};

**12. Render a Class Component to the DOM**

class TypesOfFood extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h1>Types of Food:</h1>

{/\* change code below this line \*/}

<Fruits />

<Vegetables />

{/\* change code above this line \*/}

</div>

);

}

};

// change code below this line

ReactDOM.render(<TypesOfFood />, document.getElementById("challenge-node"));

**13) Write a React Component from Scratch**

class MyComponent extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h1>My First React Component!</h1>

</div>

);

}

};

ReactDOM.render(<MyComponent />, document.getElementById("challenge-node"));

**14. Pass Props to a Stateless Functional Component**

const CurrentDate = (props) => {

return (

<div>

{ /\* change code below this line \*/ }

<p>The current date is: {props.date} </p>

{ /\* change code above this line \*/ }

</div>

);

};

class Calendar extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h3>What date is it?</h3>

{ /\* change code below this line \*/ }

<CurrentDate date = {Date()} />

{ /\* change code above this line \*/ }

</div>

);

}

};

### Result: What date is it?

The current date is: Tue Dec 04 2018 15:20:20 GMT+1000 (Australian Eastern Standard Time)

**15. Pass an Array as Props**

const List= (props) => {

{ /\* change code below this line \*/ }

return <p>{props.tasks.join(', ')}</p>

{ /\* change code above this line \*/ }

};

class ToDo extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h1>To Do Lists</h1>

<h2>Today</h2>

{ /\* change code below this line \*/ }

<List tasks = {["walk dog", "workout"]}/>

<h2>Tomorrow</h2>

<List tasks = {["walk dog", "workout", "laundry"]}/>

{ /\* change code above this line \*/ }

</div>

);

}

};

RESULT

# To Do Lists

## Today

walk dog, workout

## Tomorrow

walk dog, workout, laundry

**16. Use Default Props**

const ShoppingCart = (props) => {

return (

<div>

<h1>Shopping Cart Component</h1>

</div>

)

};

// change code below this line

ShoppingCart.defaultProps = { items: 0};

**17. Override Default Props**

const Items = (props) => {

return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>

}

Items.defaultProps = {

quantity: 0

}

class ShoppingCart extends React.Component {

constructor(props) {

super(props);

}

render() {

{ /\* change code below this line \*/ }

return <Items quantity = {10}/>

{ /\* change code above this line \*/ }

}

};

**18. Use PropTypes to Define the Props You Expect**

const Items = (props) => {

return <h1>Current Quantity of Items in Cart: {props.quantity}</h1>

};

// change code below this line

Items.propTypes = { quantity: PropTypes.number.isRequired};

// change code above this line

Items.defaultProps = {

quantity: 0

};

class ShoppingCart extends React.Component {

constructor(props) {

super(props);

}

render() {

return <Items />

}

};

**19. Access Props Using this.props**

class ReturnTempPassword extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

{ /\* change code below this line \*/ }

<p>Your temporary password is: <strong>{this.props.tempPassword}</strong></p>

{ /\* change code above this line \*/ }

</div>

);

}

};

class ResetPassword extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h2>Reset Password</h2>

<h3>We've generated a new temporary password for you.</h3>

<h3>Please reset this password from your account settings ASAP.</h3>

{ /\* change code below this line \*/ }

<ReturnTempPassword tempPassword = "Password" />

{ /\* change code above this line \*/ }

</div>

);

}

};

**20. Review Using Props with Stateless Functional Components**

class CampSite extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<Camper/>

</div>

);

}

};

// change code below this line

const Camper = (props) => {

return <p>{props.name}</p>

};

Camper.defaultProps = { name: 'CamperBot'};

Camper.propTypes = {name: PropTypes.string.isRequired};

**21. Create a Stateful Component**

class StatefulComponent extends React.Component {

constructor(props) {

super(props);

// initialize state here

this.state = {

name: "Name"

};

}

render() {

return (

<div>

<h1>{this.state.name}</h1>

</div>

);

}

};

22. Render State in the User Interface

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

name: 'freeCodeCamp'

}

}

render() {

return (

<div>

{ /\* change code below this line \*/ }

<h1>{this.state.name}</h1>

{ /\* change code above this line \*/ }

</div>

);

}

};

**23. Render State in the User Interface Another Way**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

name: 'freeCodeCamp'

}

}

render() {

// change code below this line

const name = this.state.name;

// change code above this line

return (

<div>

{ /\* change code below this line \*/ }

<h1>{name}</h1>

{ /\* change code above this line \*/ }

</div>

);

}

};

**24. Set State with this.setState**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

name: 'Initial State'

};

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

}

handleClick() {

// change code below this line

this.setState({name: 'React Rocks!'});

// change code above this line

}

render() {

return (

<div>

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

<h1>{this.state.name}</h1>

</div>

);

}

};

**25. Bind ‘this’ to a Class Method**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

itemCount: 0

};

// change code below this line

this.addItem = this.addItem.bind(this);

// change code above this line

}

addItem() {

this.setState({

itemCount: this.state.itemCount + 1

});

}

render() {

return (

<div>

{ /\* change code below this line \*/ }

<button onClick={this.addItem} >Click Me</button>

{ /\* change code above this line \*/ }

<h1>Current Item Count: {this.state.itemCount}</h1>

</div>

);

}

};

26. Use State to Toggle an Element

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

visibility: false

};

// change code below this line

this.toggleVisibility = this.toggleVisibility.bind(this);

// change code above this line

}

// change code below this line

toggleVisibility() {

if (this.state.visibility == true) {

this.setState({visibility: false});}

else {

this.setState({visibility: true});

}

}

// change code above this line

render() {

if (this.state.visibility) {

return (

<div>

<button onClick={this.toggleVisibility}>Click Me</button>

<h1>Now you see me!</h1>

</div>

);

} else {

return (

<div>

<button onClick={this.toggleVisibility}>Click Me</button>

</div>

);

}

}

};

**27. Write a Simple Counter**

class Counter extends React.Component {

constructor(props) {

super(props);

this.state = {

count: 0

};

// change code below this line

this.increment = this.increment.bind(this);

this.decrement = this.decrement.bind(this);

this.reset = this.reset.bind(this);

// change code above this line

}

// change code below this line

increment() {

this.setState({

count: this.state.count + 1

});

}

decrement() {

this.setState({

count: this.state.count - 1

});

}

reset() {

this.setState({

count: 0

});

}

// change code above this line

render() {

return (

<div>

<button className='inc' onClick={this.increment}>Increment!</button>

<button className='dec' onClick={this.decrement}>Decrement!</button>

<button className='reset' onClick={this.reset}>Reset</button>

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

</div>

);

}

};

RESULT:

Increment!Decrement!Reset

# Current Count: 0

**28. Create a Controlled Input**

class ControlledInput extends React.Component {

constructor(props) {

super(props);

this.state = {

input: ''

};

// change code below this line

this.handleChange = this.handleChange.bind(this);

// change code above this line

}

// change code below this line

handleChange(event) {

this.setState({

input: event.target.value});

}

// change code above this line

render() {

return (

<div>

{ /\* change code below this line \*/}

<input value = {this.state.input} onChange= {this.handleChange}/>

{ /\* change code above this line \*/}

<h4>Controlled Input:</h4>

<p>{this.state.input}</p>

</div>

);

}

};

**29. Create a Controlled Form**

class MyForm extends React.Component {

constructor(props) {

super(props);

this.state = {

input: '',

submit: ''

};

this.handleChange = this.handleChange.bind(this);

this.handleSubmit = this.handleSubmit.bind(this);

}

handleChange(event) {

this.setState({

input: event.target.value

});

}

handleSubmit(event) {

// change code below this line

event.preventDefault();

this.setState({

input: '',

submit: this.state.input

});

// change code above this line

}

render() {

return (

<div>

<form onSubmit={this.handleSubmit}>

{ /\* change code below this line \*/ }

<input value = {this.state.input} onChange = {this.handleChange}/>

{ /\* change code above this line \*/ }

<button type='submit'>Submit!</button>

</form>

{ /\* change code below this line \*/ }

<h1>{this.state.submit}</h1>

{ /\* change code above this line \*/ }

</div>

);

}

};

RESULT

Top of Form

Submit!

Bottom of Form

# hello

**30. Pass State as Props to Child Components**

In this challenge we are going to be passing state, but since state is local to its parent component we must use **props** to pass into the child component. Using props in child components will allow us to keep all the state data in the parent component and we can pass the data in one direction to the children components.

class MyApp extends React.Component {

constructor(props) {

super(props);

this.state = {

name: 'CamperBot'

}

}

render() {

return (

<div>

<Navbar name = {this.state.name} />

</div>

);

}

};

class Navbar extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h1>Hello, my name is: {this.props.name} </h1>

</div>

);

}

};

**31. Pass a Callback as Props**

class MyApp extends React.Component {

constructor(props) {

super(props);

this.state = {

inputValue: ''

}

this.handleChange = this.handleChange.bind(this);

}

handleChange(event) {

this.setState({

inputValue: event.target.value

});

}

render() {

return (

<div>

{ <GetInput input = {this.state.inputValue} handleChange = {this.handleChange}/> }

{ <RenderInput input = {this.state.inputValue} /> }

</div>

);

}

};

class GetInput extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h3>Get Input:</h3>

<input

value={this.props.input}

onChange={this.props.handleChange}/>

</div>

);

}

};

class RenderInput extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<div>

<h3>Input Render:</h3>

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

</div>

);

}

};

**32. Use the Lifecycle Method componentWillMount**

class MyComponent extends React.Component {

constructor(props) {

super(props);

}

componentWillMount() {

console.log();

}

render() {

return <div />

}

};

**33. Use the Lifecycle Method componentDidMount**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

activeUsers: null

};

}

componentDidMount() {

setTimeout( () => {

this.setState({

activeUsers: 1273

});

}, 2500);

}

render() {

return (

<div>

<h1>Active Users: { this.state.activeUsers }</h1>

</div>

);

}

};

**34. Add Event Listeners**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

message: ''

};

this.handleEnter = this.handleEnter.bind(this);

this.handleKeyPress = this.handleKeyPress.bind(this);

}

// change code below this line

componentDidMount() {

document.addEventListener("keydown", this.handleKeyPress);

}

componentWillUnmount() {

document.removeEventListener("keydown", this.handleKeyPress);

}

// change code above this line

handleEnter() {

this.setState({

message: this.state.message + 'You pressed the enter key! '

});

}

handleKeyPress(event) {

if (event.keyCode === 13) {

this.handleEnter();

}

}

render() {

return (

<div>

<h1>{this.state.message}</h1>

</div>

);

}

};

**35. Manage Updates with Lifecycle Methods**

class Dialog extends React.Component {

constructor(props) {

super(props);

}

componentWillUpdate() {

console.log('Component is about to update...');

}

// change code below this line

componentWillReceiveProps(nextProps) {

console.log(this.props, nextProps);

}

componentDidUpdate () {

console.log ('Component has updated');

}

// change code above this line

render() {

return <h1>{this.props.message}</h1>

}

};

class Controller extends React.Component {

constructor(props) {

super(props);

this.state = {

message: 'First Message'

};

this.changeMessage = this.changeMessage.bind(this);

}

changeMessage() {

this.setState({

message: 'Second Message'

});

}

render() {

return (

<div>

<button onClick={this.changeMessage}>Update</button>

<Dialog message={this.state.message}/>

</div>

);

}

};

**36. Optimize Re-Renders with shouldComponentUpdate**

class OnlyEvens extends React.Component {

constructor(props) {

super(props);

}

shouldComponentUpdate(nextProps, nextState) {

console.log('Should I update?');

// change code below this line

if (nextProps.value % 2 == 0) {

return true;

}else {

return false;

}

// change code above this line

}

componentWillReceiveProps(nextProps) {

console.log('Receiving new props...');

}

componentDidUpdate() {

console.log('Component re-rendered.');

}

render() {

return <h1>{this.props.value}</h1>

}

};

class Controller extends React.Component {

constructor(props) {

super(props);

this.state = {

value: 0

};

this.addValue = this.addValue.bind(this);

}

addValue() {

this.setState({

value: this.state.value + 1

});

}

render() {

return (

<div>

<button onClick={this.addValue}>Add</button>

<OnlyEvens value={this.state.value}/>

</div>

);

}

};

**37. Introducing Inline Styles**

class Colorful extends React.Component {

render() {

return (

<div style = {{color: "red", fontSize: 72}}>Big Red</div>

);

}

};

**38. Add Inline Styles in React**

const styles = {color: "purple", fontSize: 40, border: "2px solid purple"};

class Colorful extends React.Component {

render() {

return (

<div style = {styles}>Style Me!</div>

);

}

};

**39. Use Advanced JavaScript in React Render Method**

const inputStyle = {

width: 235,

margin: 5

}

class MagicEightBall extends React.Component {

constructor(props) {

super(props);

this.state = {

userInput: '',

randomIndex: ''

}

this.ask = this.ask.bind(this);

this.handleChange = this.handleChange.bind(this);

}

ask() {

if (this.state.userInput) {

this.setState({

randomIndex: Math.floor(Math.random() \* 20),

userInput: ''

});

}

}

handleChange(event) {

this.setState({

userInput: event.target.value

});

}

render() {

const possibleAnswers = [

'It is certain',

'It is decidedly so',

'Without a doubt',

'Yes, definitely',

'You may rely on it',

'As I see it, yes',

'Outlook good',

'Yes',

'Signs point to yes',

'Reply hazy try again',

'Ask again later',

'Better not tell you now',

'Cannot predict now',

'Concentrate and ask again',

'Don\'t count on it',

'My reply is no',

'My sources say no',

'Most likely',

'Outlook not so good',

'Very doubtful'

];

const answer = possibleAnswers[this.state.randomIndex]; // << change code here

return (

<div>

<input

type="text"

value={this.state.userInput}

onChange={this.handleChange}

style={inputStyle} /><br />

<button onClick={this.ask}>

Ask the Magic Eight Ball!

</button><br />

<h3>Answer:</h3>

<p>

{ /\* change code below this line \*/ }

{answer}

{ /\* change code above this line \*/ }

</p>

</div>

);

}

};

**40. Render with an If/Else Condition**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

display: true

}

this.toggleDisplay = this.toggleDisplay.bind(this);

}

toggleDisplay() {

this.setState({

display: !this.state.display

});

}

render() {

// change code below this line

if(this.state.display === true) {

return (

<div>

<button onClick={this.toggleDisplay}>Toggle Display</button>

<h1>Displayed!</h1>

</div>

);

}

else {

return (

<div>

<button onClick={this.toggleDisplay}>Toggle Display</button>

</div>

);

}

}

};

**41. Use && for a More Concise Conditional**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

display: true

}

this.toggleDisplay = this.toggleDisplay.bind(this);

}

toggleDisplay() {

this.setState({

display: !this.state.display

});

}

render() {

return (

<div>

<button onClick={this.toggleDisplay}>Toggle Display</button>

**{this.state.display && <h1>Displayed!</h1>}**

</div>

);

}

};

**42. Use a Ternary Expression for Conditional Rendering**

const inputStyle = {

width: 235,

margin: 5

}

class CheckUserAge extends React.Component {

constructor(props) {

super(props);

// change code below this line

this.state = {

input: '',

userAge: ''

}

// change code above this line

this.submit = this.submit.bind(this);

this.handleChange = this.handleChange.bind(this);

}

handleChange(e) {

this.setState({

input: e.target.value,

userAge: ''

});

}

submit() {

this.setState({

userAge: this.state.input

});

}

render() {

const buttonOne = <button onClick={this.submit}>Submit</button>;

const buttonTwo = <button>You May Enter</button>;

const buttonThree = <button>You Shall Not Pass</button>;

return (

<div>

<h3>Enter Your Age to Continue</h3>

<input

style={inputStyle}

type="number"

value={this.state.input}

onChange={this.handleChange} /><br />

{

/\* change code here \*/

(this.state.userAge >= 18) ? buttonTwo : (this.state.userAge == '') ? buttonOne : buttonThree

}

</div>

);

}

};

**43. Render Conditionally from Props**

class Results extends React.Component {

constructor(props) {

super(props);

}

render() {

return (

<h1>

{

/\* change code here \*/

this.props.fiftyFifty

}

</h1>

)

};

};

class GameOfChance extends React.Component {

constructor(props) {

super(props);

this.state = {

counter: 1

}

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

}

handleClick() {

this.setState({

counter: this.state.counter + 1 // change code here

});

}

render() {

let expression = Math.random() > 0.5; // change code here

return (

<div>

<button onClick={this.handleClick}>Play Again</button>

{ /\* change code below this line \*/ }

{(expression == 1) ? <Results fiftyFifty="You win!"/> : <Results fiftyFifty="You lose!" />}

{ /\* change code above this line \*/ }

<p>{'Turn: ' + this.state.counter}</p>

</div>

);

}

};

Play Again



# You lose!

Turn: 20

**44. Change Inline CSS Conditionally Based on Component State**

class GateKeeper extends React.Component {

constructor(props) {

super(props);

this.state = {

input: ''

};

this.handleChange = this.handleChange.bind(this);

}

handleChange(event) {

this.setState({ input: event.target.value })

}

render() {

let inputStyle = {

border: '1px solid black'

};

// change code below this line

if(this.state.input.length > 15) {

inputStyle = {

border: '3px solid red'

}

}

// change code above this line

return (

<div>

<h3>Don't Type Too Much:</h3>

<input

type="text"

style={inputStyle}

value={this.state.input}

onChange={this.handleChange} />

</div>

);

}

};

**45. Use Array.map() to Dynamically Render Elements**

const textAreaStyles = {

width: 235,

margin: 5

};

class MyToDoList extends React.Component {

constructor(props) {

super(props);

// change code below this line

this.state = {

userInput: '',

toDoList: []

}

// change code above this line

this.handleSubmit = this.handleSubmit.bind(this);

this.handleChange = this.handleChange.bind(this);

}

handleSubmit() {

const itemsArray = this.state.userInput.split(',');

this.setState({

toDoList: itemsArray

});

}

handleChange(e) {

this.setState({

userInput: e.target.value

});

}

render() {

const items = this.state.toDoList.map(i => <li>{i}</li>); // change code here

return (

<div>

<textarea

onChange={this.handleChange}

value={this.state.userInput}

style={textAreaStyles}

placeholder="Separate Items With Commas" />

<br />

<button onClick={this.handleSubmit}>Create List</button>

<h1>My "To Do" List:</h1>

<ul>

{items}

</ul>

</div>

);

}

};

**46. Give Sibling Elements a Unique Key Attribute**

const frontEndFrameworks = [

'React',

'Angular',

'Ember',

'Knockout',

'Backbone',

'Vue'

];

function Frameworks() {

const renderFrameworks = frontEndFrameworks.map(i => <li key = {i +1}>{i}</li>); // change code here

return (

<div>

<h1>Popular Front End JavaScript Frameworks</h1>

<ul>

{renderFrameworks}

</ul>

</div>

);

};

**47. Use Array.filter() to Dynamically Filter and Array**

class MyComponent extends React.Component {

constructor(props) {

super(props);

this.state = {

users: [

{

username: 'Jeff',

online: true

},

{

username: 'Alan',

online: false

},

{

username: 'Mary',

online: true

},

{

username: 'Jim',

online: false

},

{

username: 'Sara',

online: true

},

{

username: 'Laura',

online: true

}

]

}

}

render() {

const usersOnline = this.state.users.filter(user => user.online == true); // change code here

const renderOnline = usersOnline.map(u => <li key = {u.username+1}>{u.username}</li>); // change code here

return (

<div>

<h1>Current Online Users:</h1>

<ul>

{renderOnline}

</ul>

</div>

);

}

};

**48. Render React on the Server with renderToString**

class App extends React.Component {

constructor(props) {

super(props);

}

render() {

return <div/>

}

};

// change code below this line

ReactDOMServer.renderToString(<App/>)