# React Assignment

Imagine a scenario, there's a Kid who's going to perform in front of Judges and People. He's nervous as well.
Here's the component of Kid.

## Kid.js

```
import React from 'react';

export default class Kid extends React.Component {

 constructor(props) {
   super(props);
   this.state = { emotion: 'nervous', danceSteps: [], currentStepIndex: 0,
startedPerforming: false } ;
 }

 qualified() {
   this.setState({startedPerforming: false})
 }

 render() {
   const {dressColor} = this.props;
   const {danceSteps, emotion, startedPerforming, currentStepIndex} =
this.state;

   return (
   <div>
     <div>dressColor: { dressColor }</div>
```

```
      <div style={{backgroundColor: dressColor, width: 50, height:
50}}></div>
    <div>Emotion: { emotion }</div>
    {startedPerforming &&
    <div>
      <div>Current Step: {danceSteps[currentStepIndex]}</div>
      <button onClick={() => this.setState({currentStepIndex:
currentStepIndex + 1})}>Perform Next Step</button>
    </div>}
</div>
   );
 }
}
Kid.defaultProps = { dressColor: 'red', applaud: false, furtherSteps: [] };
```

If for some reason our kid fails to get his own dress for the performance then he can wear the default red colored dress provided by his school. If the parent Component fails to send dressColor prop to Kid class then it's value will be set to *red*.

**TASK 1:**
Import this component into your **App.js** and provide a prop `dressColor` to the Kid with any color value and check if the Kid's component is affected with that color!

Now the kid is standing behind the stage, doing all last-minute preparations and patiently waiting for curtains to be raised. Meanwhile the stage management asked the sound operator to increase the volume.

**TASK 2:**
In **App.js** before rendering the Kid Component, increase the volume to 5 (state: *volume=5*).
Hint: getDerivedStateFromProps()

Now the curtain is just raised, our kid is just seen by the audience and the song is about to start. Our kid now needs to get over his nervousness, now he's trying to remember his first steps of the dance and start the performance.

**TASK 3:**
So programmatically in **Kid.js**, you need to add the steps into a state (state: *danceSteps=['step1', 'step2']*).
Hint: ComponentDidMount

Now our kid had trouble remembering further steps so he looked at his teacher standing at the corner of the stage who prompted further steps to him.

**Teacher.js**

```
import React from 'react';

export default class Teacher extends React.Component {
 sendDataToKid() {
      const furtherSteps = [‘step3’, ‘step4’, ‘step5’]
      //Send this data to Kid.js
 }

 render() {

   return (
     <button onClick={this.sendDataToKid}>Get Help From Teacher</button>
   );
 }
}
```

**TASK 4:**
Render this component in App.js, you'll get the button for help, on clicking it, it should send the steps to the Kid.js component and kid should remember these steps and start performing as well.
Hint: getDerivedStateFromProps()

Now the Kid started performing, but he's still nervous, what happened now, the Judges applaud for him, this changed his emotions to `happy` now.

**Judge.js**

```
export default class Judge extends React.Component {
 constructor() {
  super()

  this.state = {stars: 0, available: false}
 }

 applaud() {
   //Send this applaud status to Kid.js
 }

 provideStars() {
   const {stars} = this.state;
```

```
    this.setState({stars: stars + 1})
  }

  render() {
    const {stars, available} = this.state;

    return (
      <div>
        <button type="button" onClick={this.applaud.bind(this)}>
         Appreciate performance
        </button>
        <button type="button" onClick={this.provideStars.bind(this)}>
         Provide stars
        </button>

        Kid is available: {available}

        Stars gained: {stars}
      </div>
    );
  }
}
```

**TASK 5:**

Render this component in App.js, you'll get the button for Appreciate Performance, on clicking it, it should send the applaud status to the Kid.js component and kid's emotion should change to 'happy'.

Hint: getDerivedStateFromProps()

Now, the Judges are providing stars but the stars can't be provided more than 5.

**TASK 6:**

In **Judges.js**, use `Provide stars` button to increase the stars, but make sure if the increment is going for more than 5, don't show the increase one in the render. Render should show 5 stars only

Hint: shouldComponentUpdate()

After 5 stars provided, the kid should end the performance

**TASK 7:**

Using `provideStars` method, when the stars are 5, tell the Kid component that he's qualified and stop the performance.

Hint: Use ComponentDidUpdate() and call the `qualified` method from in it on **a condition to prevent infinte looping state.**

After the kid is qualified, the management asked the kid to leave.

**TASK 8:**
Provide a button “Ask the Kid to Leave the Show” in **App.js** that will unmount the Kid component.

After the kid left happily, the judges also decided to leave.

**TASK 9:**
On unmounting the Kid, unmount the Judges component as well.
Hint: Use componentWillUnmount() in **Kid.js**