Components cont.

State and Lifecycle

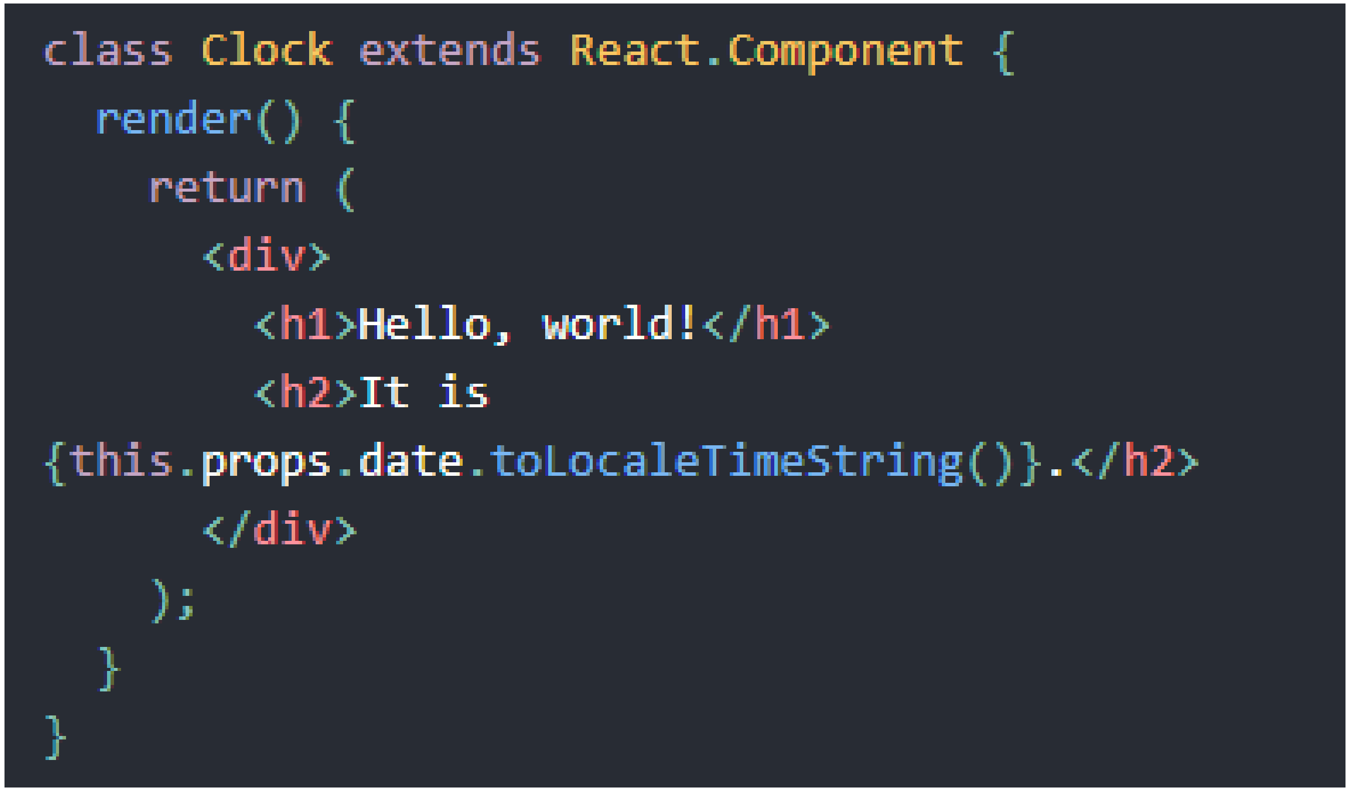
LEARNING OBJECTIVES

At the end of this module, the learner will be able to:

- convert functions to classes.
- add local state to a class
- add lifecycle method to a class

CONVERTING A FUNCTION TO A CLASS

1. Create a class with the same name, that extends React.Component
2. Add a single empty method to it called render( )
3. Move the body of the function int the render( )
4. Replace props with this.props in the render body
5. Delete the remaining empty function declared

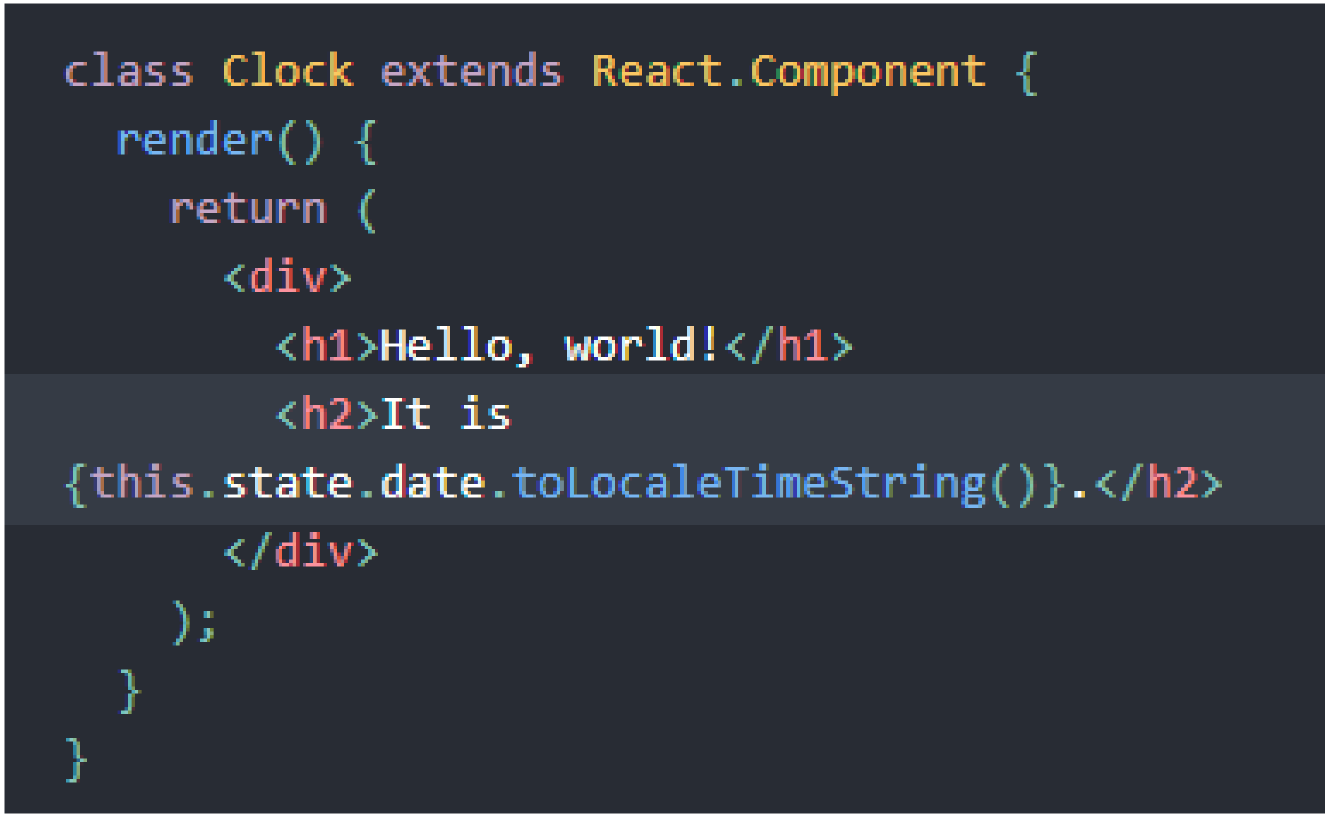


```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
{this.props.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

Try in CodePen

ADDING LOCAL STATE TO A CLASS

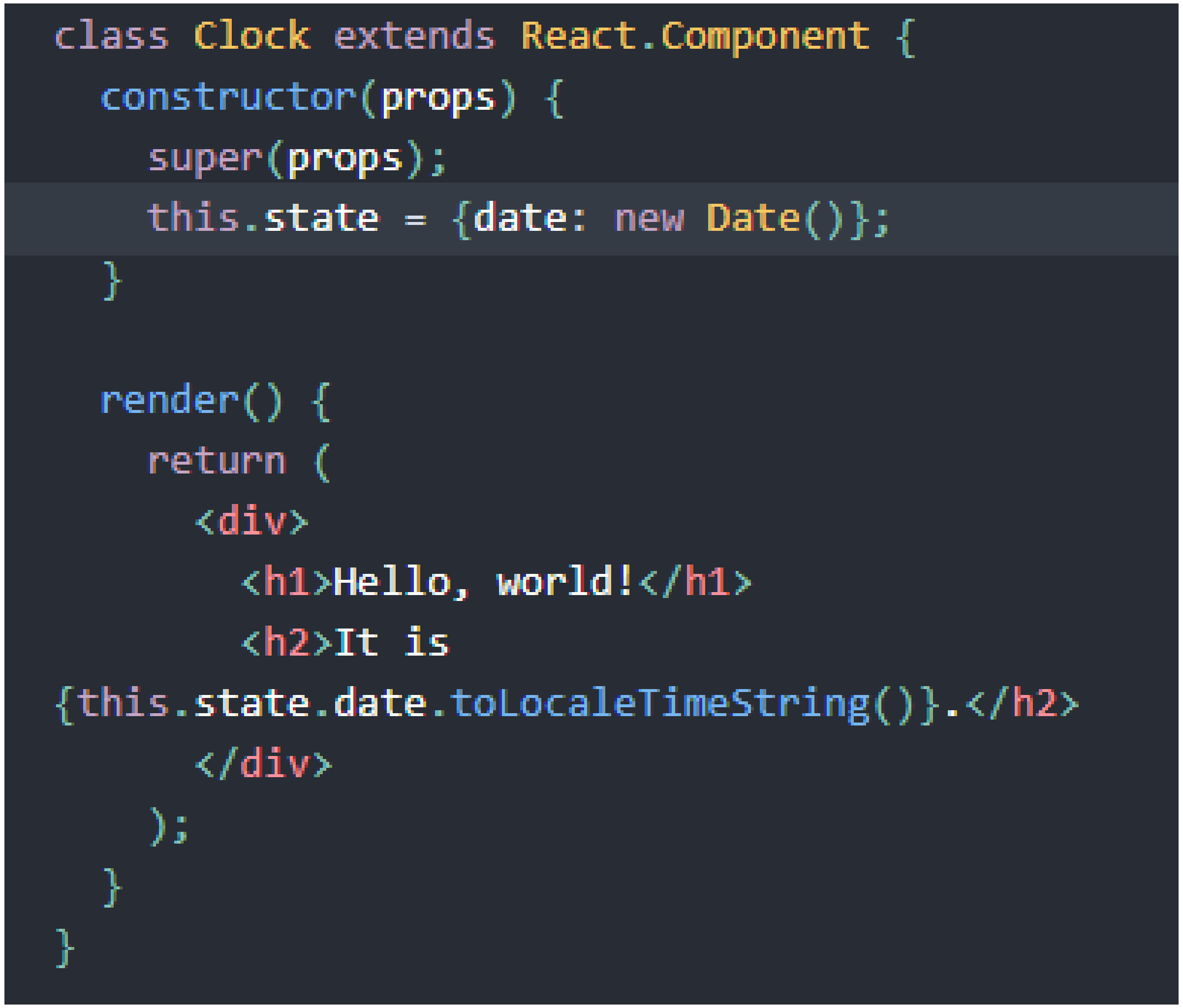
1. Replace this.date with this state.data in the **render()** method



```
class Clock extends React.Component {
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
{this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

ADDING LOCAL STATE TO A CLASS

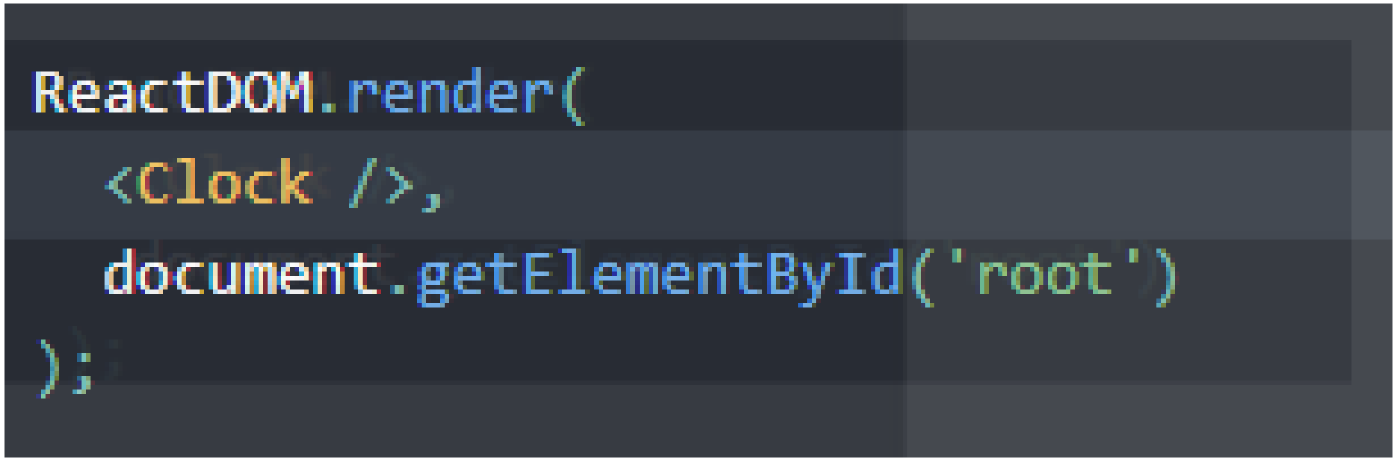
2. Add a constructor that assigns the initial **this.state**



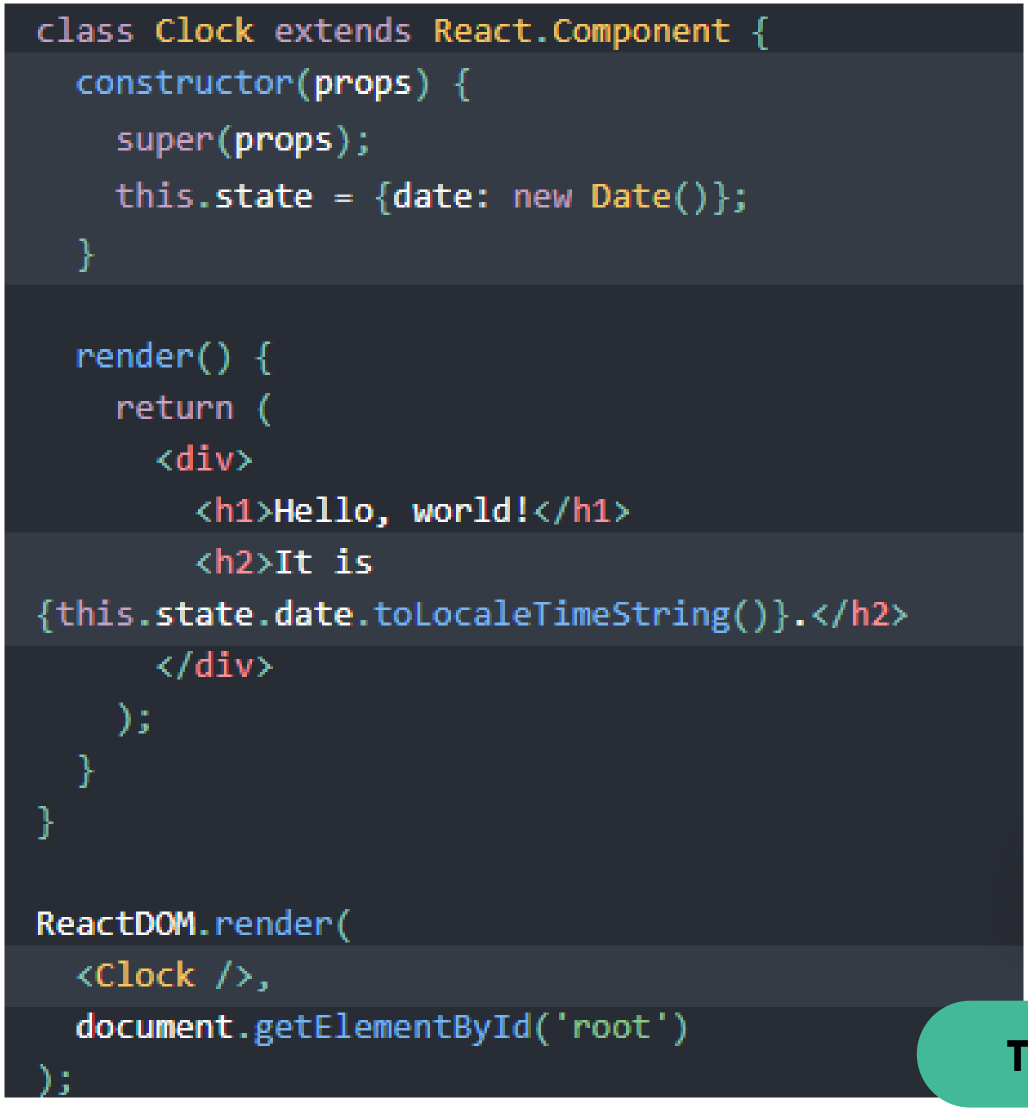
```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
{this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

ADDING LOCAL STATE TO A CLASS

3. Remove the app from <cClock /> element



```
ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

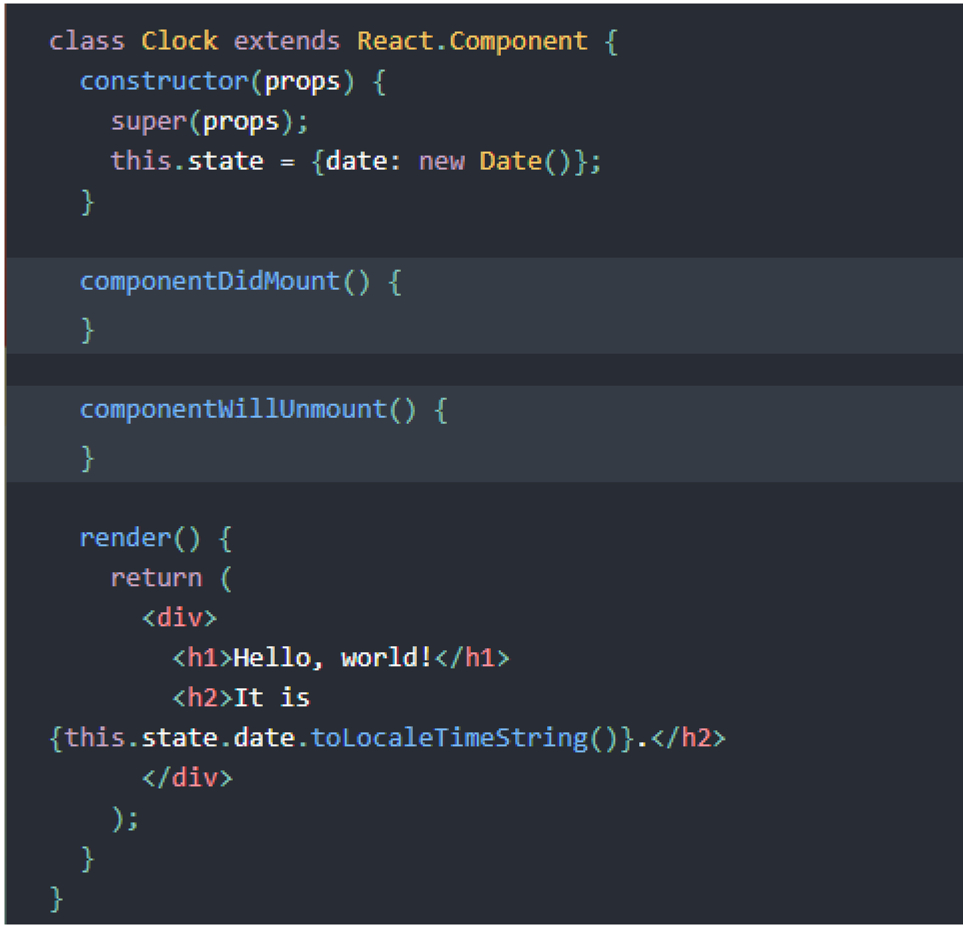


```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
{this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

Try in CodePen

ADD LIFECYCLE METHODS TO A CLASS

1. Declare special methods on the component class to run some code when a component mounts and unmounts

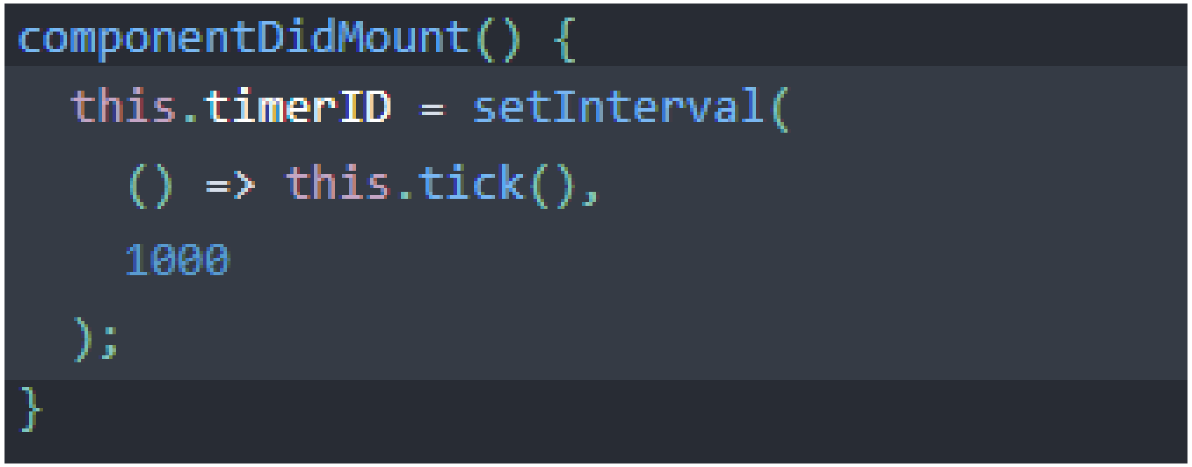


```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
  }

  componentWillUnmount() {
  }

  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is
{this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}
```

LIFECYCLE METHODS

componentDidMount()

method runs after the component output has been rendered to the DOM.

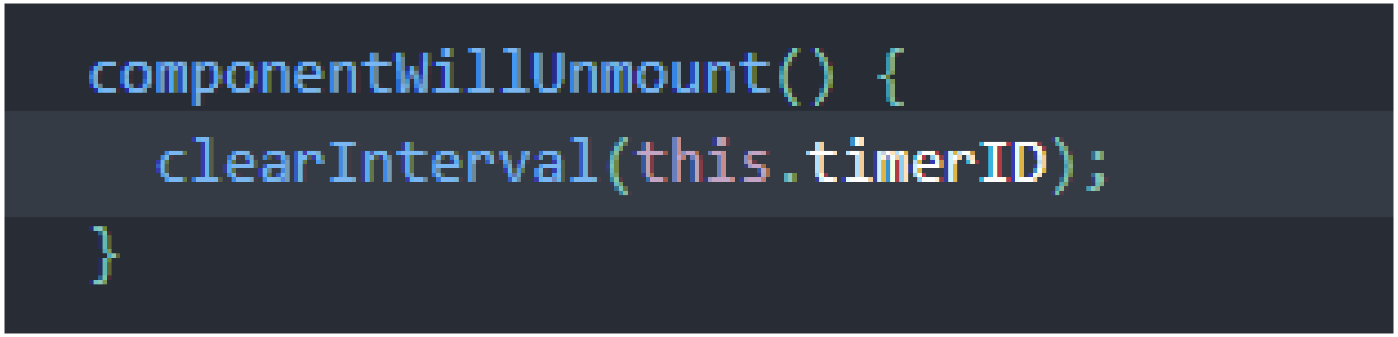


```
componentDidMount() {
  this.timerID = setInterval(
    () => this.tick(),
    1000
  );
}
```

LIFECYCLE METHODS

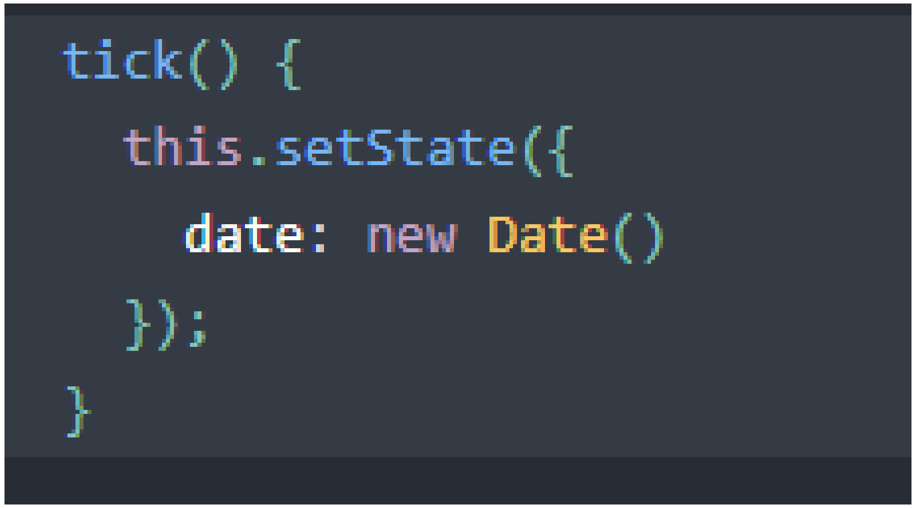
componentWillUnmount()

is invoked immediately before a component is unmounted and destroyed.

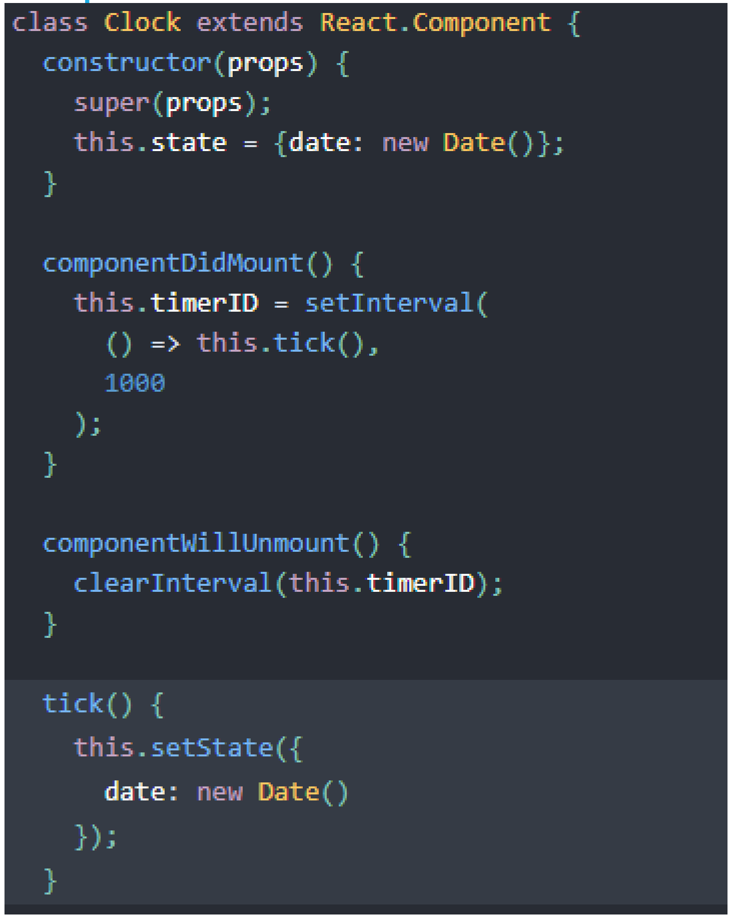


```
componentWillUnmount() {
  clearInterval(this.timerID);
}
```

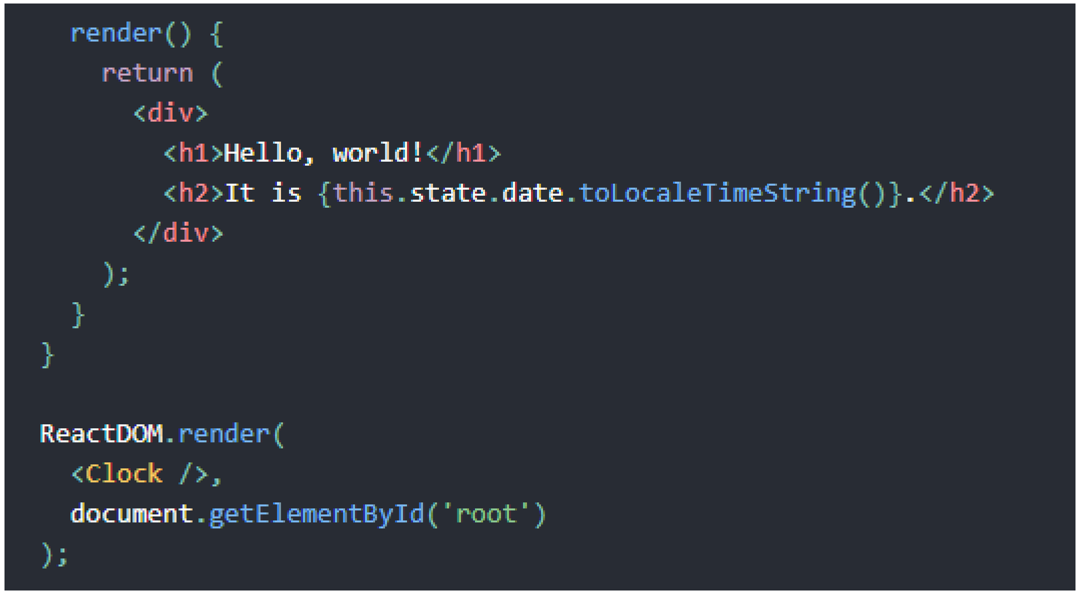
2. Implement the method tick( )



```
tick() {
  this.setState({
    date: new Date()
  });
}
```



```
class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = {date: new Date()};
  }

  componentDidMount() {
    this.timerID = setInterval(
      () => this.tick(),
      1000
    );
  }

  componentWillUnmount() {
    clearInterval(this.timerID);
  }

  tick() {
    this.setState({
      date: new Date()
    });
  }
```



```
  render() {
    return (
      <div>
        <h1>Hello, world!</h1>
        <h2>It is {this.state.date.toLocaleTimeString()}.</h2>
      </div>
    );
  }
}

ReactDOM.render(
  <Clock />,
  document.getElementById('root')
);
```

Try in CodePen

USING SETSTATE( )

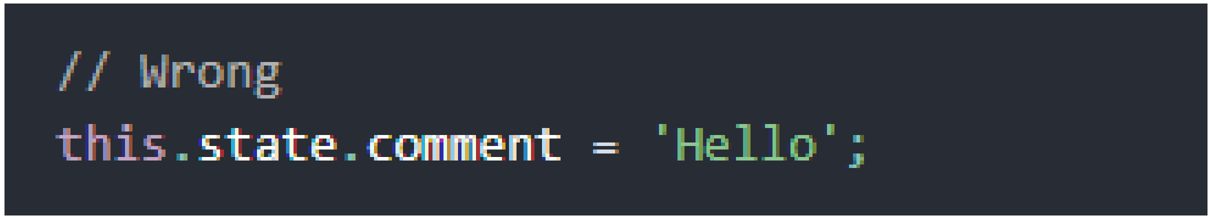
Do not modify state directly

State updates may be asynchronous

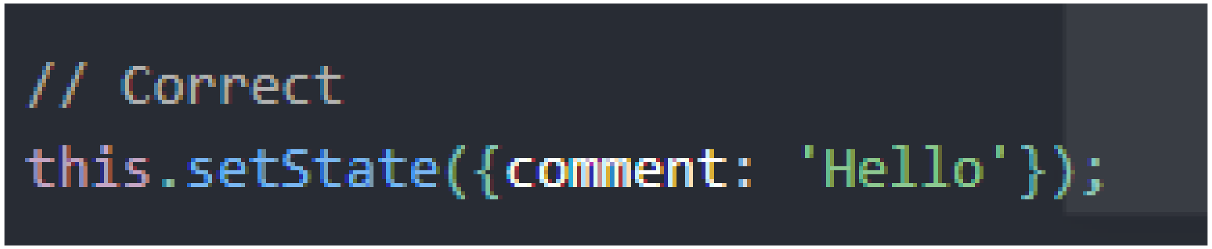
State updates are merge

USING SETSTATE( )

Do not modify state directly



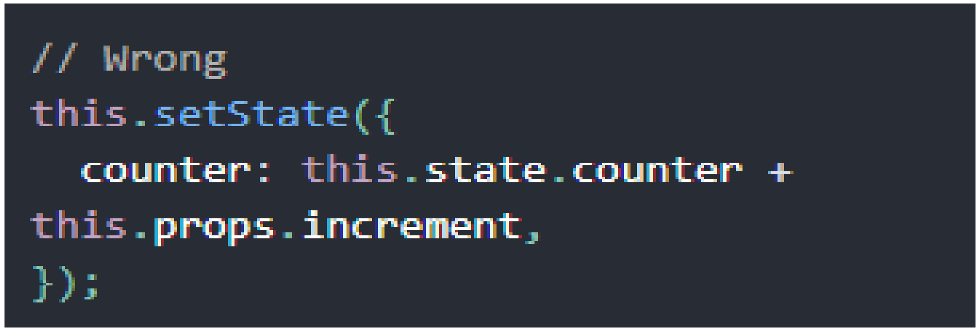
```
// Wrong
this.state.comment = 'Hello';
```



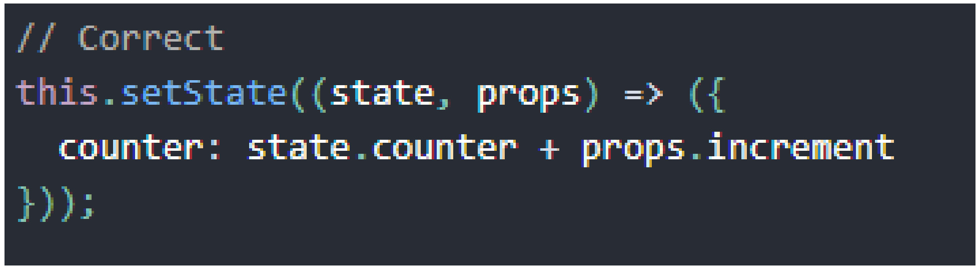
```
// Correct
this.setState({comment: 'Hello'});
```

USING SETSTATE( )

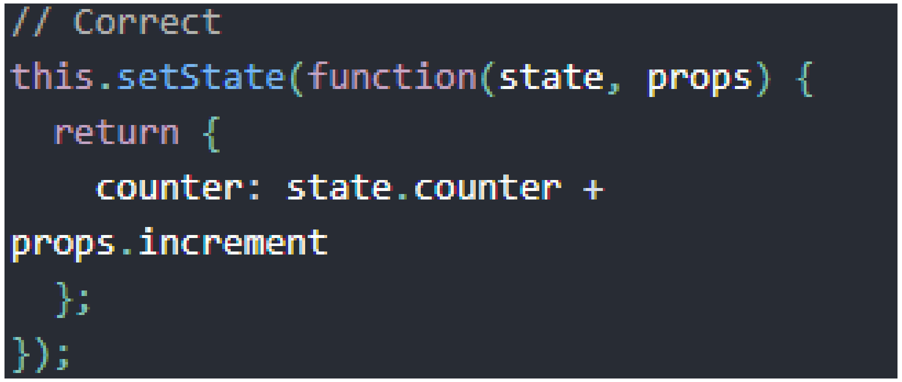
State updates may be asynchronous



```
// Wrong
this.setState({
  counter: this.state.counter +
this.props.increment,
});
```



```
// Correct
this.setState((state, props) => ({
  counter: state.counter + props.increment
}));
```

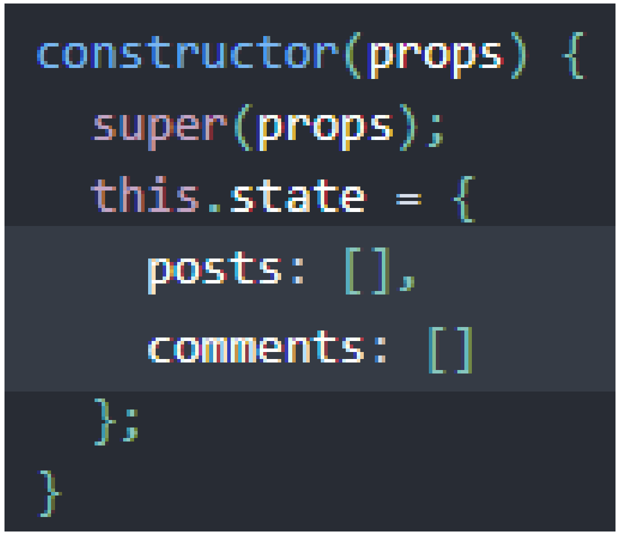


```
// Correct
this.setState(function(state, props) {
  return {
    counter: state.counter +
props.increment
  };
});
```

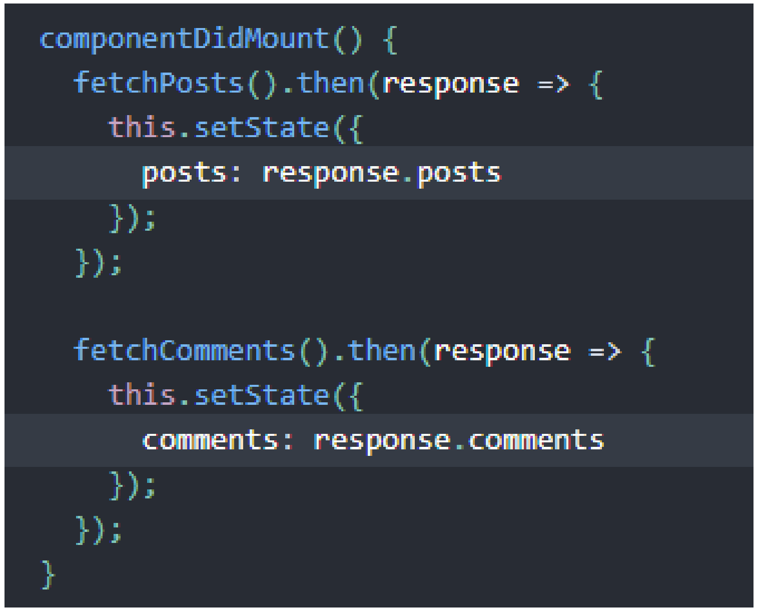
USING SETSTATE( )

State updates are merge

When you call setState(), React merges the object you provide into the current state.



```
constructor(props) {
  super(props);
  this.state = {
    posts: [],
    comments: []
  };
}
```



```
componentDidMount() {
  fetchPosts().then(response => {
    this.setState({
      posts: response.posts
    });
  });

  fetchComments().then(response => {
    this.setState({
      comments: response.comments
    });
  });
}
```

REFERENCE

SimpliLearn. (n.d.). ReactJS Tutorial: A Step-by-Step Guide To Learn React. https://www.simplilearn.com/tutorials/reactjs-tutorial

ReactJS Documentation: https://reactjs.org/docs/