Method 1: static getDerivedStateFromProps

In this lesson, we'll discuss a React lifecycle method in which we use state and render objects according to our example's requirement.

Before explaining how this lifecycle method works, let me show you how the method is used.

The basic structure looks like this:

```
const MyComponent extends React.Component {

  static getDerivedStateFromProps() {
    //do stuff here
  }
}
```

The method takes in `props` and `state`:

```
static getDerivedStateFromProps(props, state) {
    //do stuff here
}
```

And you can either return an object to update the state of the component:

```
static getDerivedStateFromProps(props, state) {

  return {
    points: 200 // update state with this
  }
}
```

Or return `null` to make no updates:

```
static getDerivedStateFromProps (props, state) {
    return null;
}
```

I know what you’re thinking. Why exactly is this lifecycle method important?

Well, it is one of the rarely used lifecycle methods, but it comes in handy in certain scenarios.

First, this method is called (or invoked) before the component is rendered to the `DOM` on the initial mount.

Example: Football Team Points

Consider a simple component that renders the number of `points` scored by a football team.

As you may have expected, the number of `points` is stored in the component `state` object:

```
.App {
  text-align: center;
  display: flex;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 40vmin;
}

.App-header {
  border: 1px solid #282c34;
  flex: 1;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
}

.App-header img {
  border: 0
}

.App-link {
  color: #61dafb;
}
```

```
.App-chat {
  min-width: 400px;
  /* background: linear-gradient(

      -45deg,
      #183850 0,
      #183850 25%,
      #192c46 50%,
      #22254c 75%,
      #22254c 100%
    )
    no-repeat; */
}

/**
  chat
**/
.chat-thread {
  padding: 0 20px 0 20px;
  list-style: none;
  display: flex;
  flex-direction: column;
  overflow-y: scroll;
  max-height: calc(100vh - 50px);
}

.chat-bubble {
  position: relative;
  background-color: rgba(25, 147, 147, 0.2);
  padding: 16px 40px 16px 20px;
  margin: 0 0 20px 0;
  border-radius: 10px;
}

.chat-bubble:nth-child(n) {
  margin-right: auto;
}

.chat-btn {
  padding: 5px;
  margin: 0 0 0 10px;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Note that the text tells you that you have scored 10 points, where 10 is the number of points in the state object.

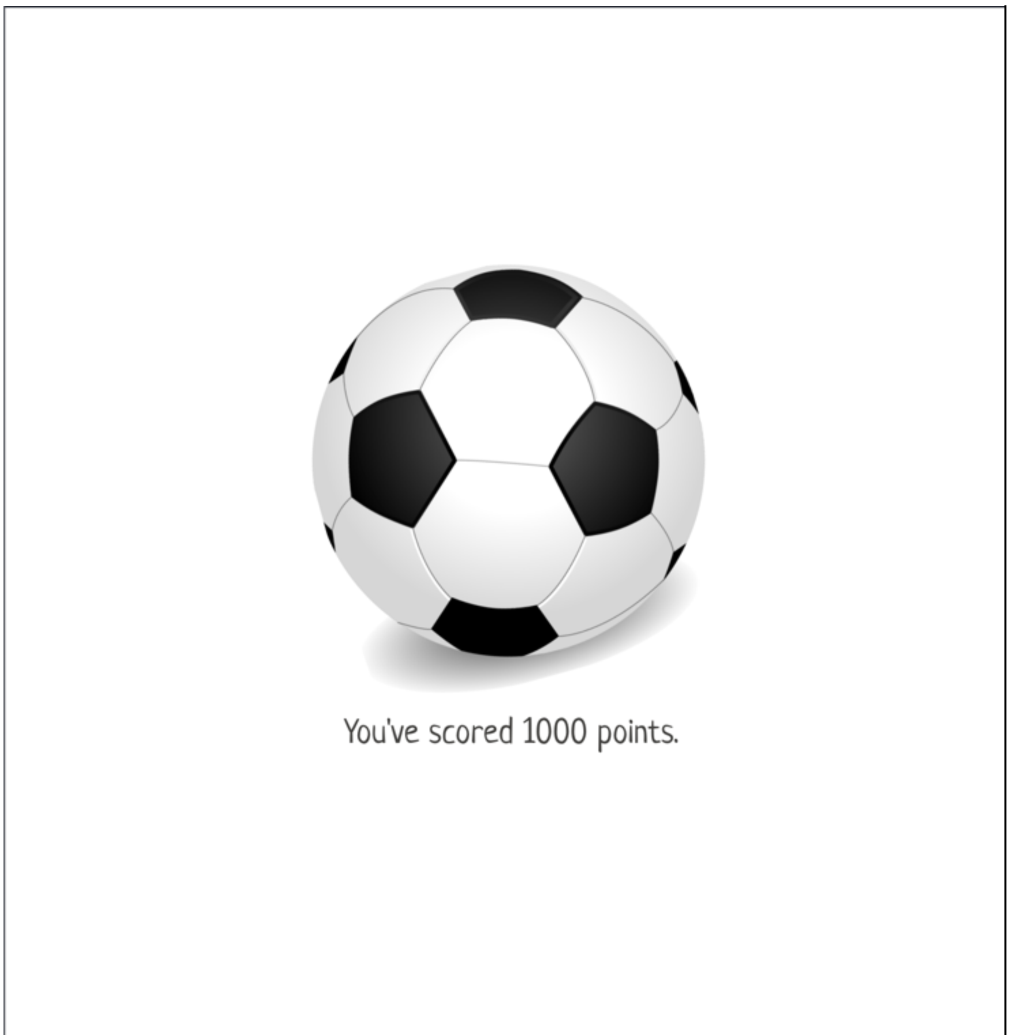
Calling the `static getDerivedStateFromProps` Method

Just as an example: if you put in the `static getDerivedStateFromProps` method as shown below, guess what number of points will be rendered.

as shown below, guess what number of points will be rendered.

```
.App {
  text-align: center;
  display: flex;
}

.App-logo {
  animation: App-logo-spin infinite 20s linear;
  height: 40vmin;
}

.App-header {
  border: 1px solid #282c34;
  flex: 1;
  min-height: 100vh;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
}

.App-header img {
  border: 0
}

.App-link {
  color: #61dafb;
}

.App-chat {
  min-width: 400px;
  /* background: linear-gradient(
      -45deg,
      #183850 0,
      #183850 25%,
      #192c46 50%,
      #22254c 75%,
      #22254c 100%
    )
    no-repeat; */
}

/**
  chat
**/
.chat-thread {
  padding: 0 20px 0 20px;
  list-style: none;
  display: flex;
  flex-direction: column;
  overflow-y: scroll;
  max-height: calc(100vh - 50px);
}

.chat-bubble {
  position: relative;
  background-color: rgba(25, 147, 147, 0.2);
  padding: 16px 40px 16px 20px;
  margin: 0 0 20px 0;
  border-radius: 10px;
}


.chat-bubble:nth-child(n) {
  margin-right: auto;
}

.chat-btn {
  padding: 5px;
  margin: 0 0 0 10px;
}

@keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
}
```

Right now, we have the `static getDerivedStateFromProps` component lifecycle method in there. If you remember from the previous explanation, this method is called before the component is mounted to the `DOM`. By returning an object, we update the `state` of the component before it is even rendered. And here's what we get:



You've scored 1000 points.

With 1000 coming from updating `state` within the `static getDerivedStateFromProps` method.

This example is a bit contrived and not the way you'd use the `static getDerivedStateFromProps` method. I just wanted to make sure you understood the basics first.

With this lifecycle method, just because you can update `state` doesn't mean you should go ahead and do so. There are specific use cases for the `static getDerivedStateFromProps` method, otherwise, you'll be solving a problem with the wrong tool.

So, when should you use the `static getDerivedStateFromProps` lifecycle method?

The method name, `getDerivedStateFromProps`, is comprised of five different words, **Get Derived State From Props**.

The component `state` in this manner is referred to as **Derived State**. As a rule of thumb, the derived state should be used sparingly as you can introduce

subtle bugs into your application if you aren’t sure of what you’re doing.

In the next lesson, we’ll discuss another lifecycle method, `getSnapshotBeforeUpdate`.