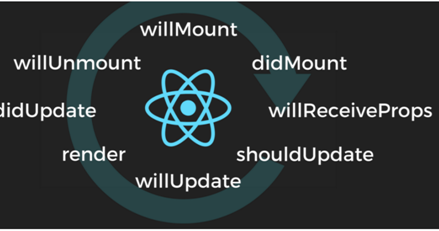


****

* **React component lifecycle has three categories – Mounting, Updating and Unmounting.**
* **The *render()*is the most used lifecycle method.**
  + **It is a pure function.**
  + **You cannot set state in render()**
* **The *componentDidMount*() happens as soon as your component is mounted.**
  + **You can set state here but with caution.**
* **The *componentDidUpdate()*happens as soon as the updating happens.**
  + **You can set state here but with caution.**
* **The *componentWillUnmount()*happens just before the component unmounts and is destroyed.**
  + **This is a good place to cleanup all the data.**
  + **You cannot set state here.**
* **The *shouldComponentUpdate()*can be used rarely.**
  + **It can be called if you need to tell React not to re-render for a certain state or prop change.**
  + **This needs to be used with caution only for certain performance optimizations.**
* **The two new lifecycle methods are *getDerivedStateFromProps()*and *getSnapshotBeforeUpdate().***
  + **They need to be used only occasionally.**
  + **Not many examples are out there for these two methods and they are still being discussed and will have more references in the future.**

**===============**

**import React from 'react';**

**class App extends React.Component {**

**constructor(props) {**

**super(props);**

**this.state = {**

**data: 0**

**}**

**this.setNewNumber = this.setNewNumber.bind(this)**

**};**

**setNewNumber() {**

**this.setState({data: this.state.data + 1})**

**}**

**render() {**

**return (**

**<div>**

**<button onClick = {this.setNewNumber}>INCREMENT</button>**

**<Content myNumber = {this.state.data}></Content>**

**</div>**

**);**

**}**

**}**

**class Content extends React.Component {**

**componentWillMount() {**

**console.log('Component WILL MOUNT!')**

**}**

**componentDidMount() {**

**console.log('Component DID MOUNT!')**

**}**

**componentWillReceiveProps(newProps) {**

**console.log('Component WILL RECIEVE PROPS!')**

**}**

**shouldComponentUpdate(newProps, newState) {**

**return true;**

**}**

**componentWillUpdate(nextProps, nextState) {**

**console.log('Component WILL UPDATE!');**

**}**

**componentDidUpdate(prevProps, prevState) {**

**console.log('Component DID UPDATE!')**

**}**

**componentWillUnmount() {**

**console.log('Component WILL UNMOUNT!')**

**}**

**render() {**

**return (**

**<div>**

**<h3>{this.props.myNumber}</h3>**

**</div>**

**);**

**}**

**}**

**export default App;**

**Understanding the React Component Lifecycle**

**Introduction**

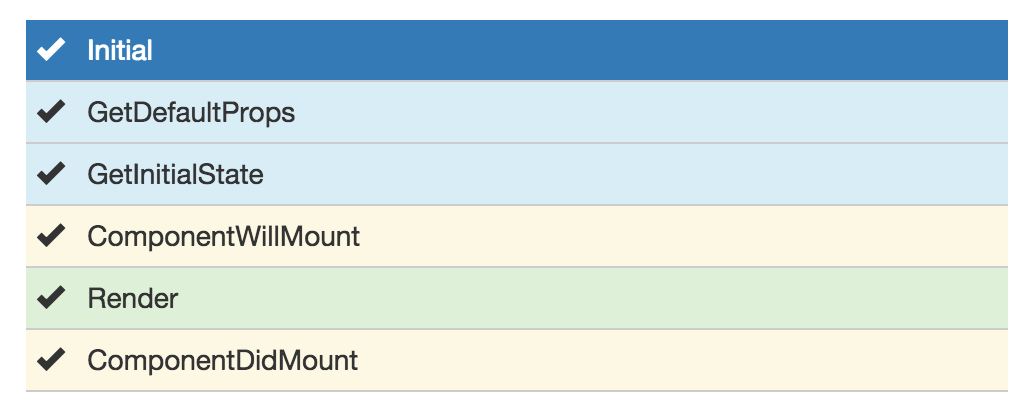
**React** enables to create components by invoking the React.createClass() method which expects a *render* method and triggers a lifecycle that can be hooked into via a number of so called lifecycle methods.

Understanding the component lifecycle will enable you to perform certain actions when a component is created or destroyed. Further more it gives you the opportunity to decide if a component should be updated in the first place and to react to props or state changes accordingly.

**The Lifecycle**

To get a clear idea of the lifecycle we will need to differentiate between the **initial** creation phase, where the component is created, and state and props changes triggered **updates** as well as the component **unmoutings** phase.

**Initialization**



From looking at the image above we can see that the first two methods being called are *getDefaultProps* and *getInitialState*. Both methods are only called once when initially rendering the component.

The ***getInitialState*** method enables to set the initial state value, that is accessible inside the component via *this.state*.

getInitialState**:function**(){

**return** { */\* something here \*/*};

}

Analogously ***getDefaultProps*** can be used to define any default props which can be accessed via *this.props*.

getDefaultProps**:function**(){

**return** { */\* something here \*/*};

}

Another two methods that only get called when initializing a component are *componentWillMount*and *componentDidMount*.

***componentWillMount*** is called before the *render* method is executed. It is important to note that setting the state in this phase will not trigger a re-rendering.

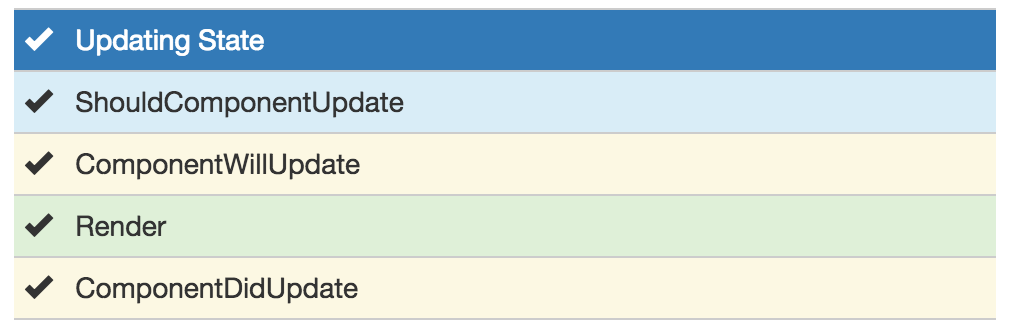
The ***render*** method returns the needed component markup, which can be a single child component or null or false (in case you don't want any rendering).

This is the part of the lifecycle where props and state values are interpreted to create the correct output. Neither props nor state should should be modified inside this function. This is important to remember, as by definition the *render* function has to be pure, meaning that the same result is returned every time the method is invoked.

As soon as the render method has been executed the ***componentDidMount*** function is called. The DOM can be accessed in this method, enabling to define DOM manipulations or data fetching operations. Any DOM interactions should always happen in this phase not inside the *render* method.

**State Changes**

State changes will trigger a number of methods to hook into.



***shouldComponentUpdate*** is always called before the *render* method and enables to define if a re-rendering is needed or can be skipped. Obviously this method is never called on initial rendering. A boolean value must be returned.

shouldComponentUpdate**:function**(nextProps, nextState){

*// return a boolean value*

**returntrue**;

}

Access to the upcoming as well as the current props and state ensure that possible changes can be detected to determine if a rendering is needed or not.

***componentWillUpdate*** gets called as soon as the the *shouldComponentUpdate* returned true. Any state changes via this.setState are not allowed as this method should be strictly used to prepare for an upcoming update not trigger an update itself.

componentWillUpdate**:function**(nextProps, nextState){

*// perform any preparations for an upcoming update*

}

Finally ***componentDidUpdate*** is called after the *render* method. Similar to the *componentDidMount*, this method can be used to perform DOM operations after the data has been updated.

componentDidUpdate**:function**(prevProps, prevState){

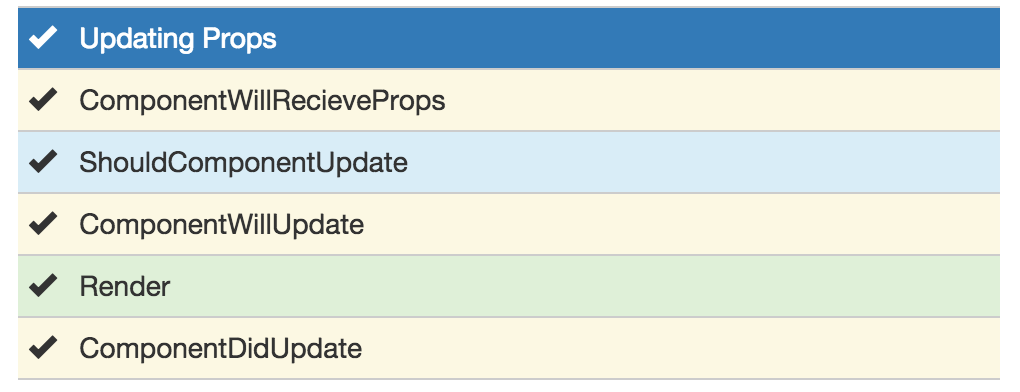
*//*

}

Check the [code](http://plnkr.co/edit/t3if5Z1hDapXFp4O5QPa) example for more detailed insights.

**Props Changes**

Any changes on the props object will also trigger the lifecycle and is almost identical to the state change with one additional method being called.



***componentWillReceiveProps*** is only called when the props have changed and when this is not an initial rendering. *componentWillReceiveProps* enables to update the state depending on the existing and upcoming props, without triggering another rendering. One interesting thing to remember here is that there is no equivalent method for the state as state changes should never trigger any props changes.

componentWillReceiveProps**:function**(nextProps) {

**this**.setState({

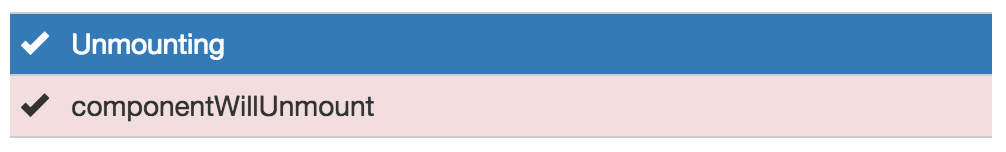
*// set something*

});

}

The rest of the lifecycle reveals nothing new here and is identical to the state change triggered cycle.

**Unmounting**



The only method we haven't touched yet is the *componentWillUnmount* which gets called before the component is removed from the DOM. This method can be beneficial when needing to perform clean up operations, f.e. removing any timers defined in *componentDidMount*.