React 101
Codecademy
Lifecycle Methods

Component Lifecycle Methods

The Component Lifecycle

We know that React Components are highly dynamic. They get created, rendered, added to the DOM, updated, and removed. All of these steps are part of a component's lifecycle.

The component lifecycle has three high-level parts:

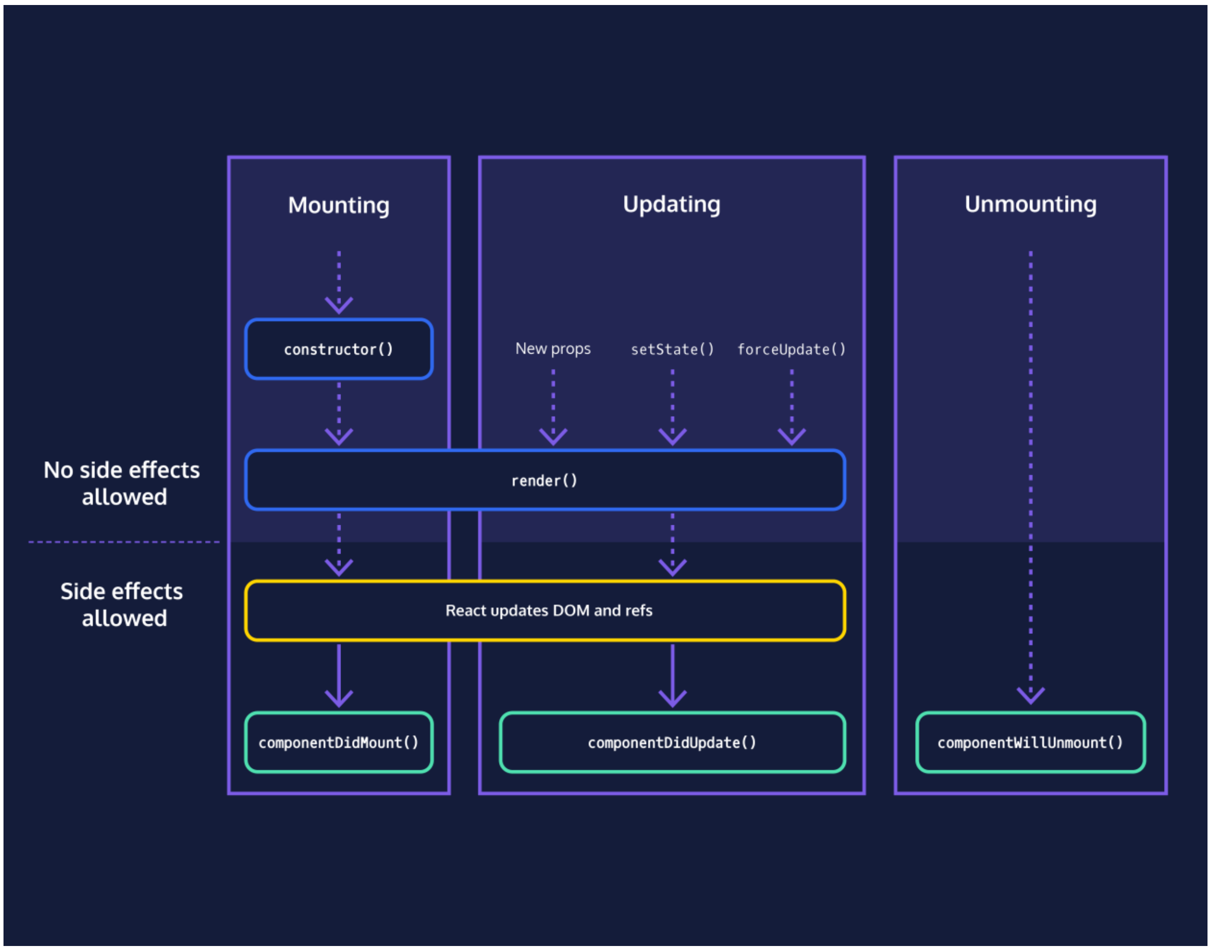
1. *Mounting*, when the component is being initialized and put into the DOM for the first time.
2. *Updating*, when the component updates as a result of changed state or changed props.
3. *Unmounting*, when the component is being removed from the DOM.

Every React component does the first step at a minimum. If a component never mounted, you'd never see it!

When a component's state changes or if different props are passed to component it updates.

Finally, a component is unmounted when it's removed from the DOM.

It's worth noting that each component instance has its own lifecycle. However, once a component instance is unmounted, that's it – it will never be re-mounted, or updated again, or unmounted.



Introduction to Lifecycle Methods

React components have several methods, called lifecycle methods, that are called at different parts of a component's lifecycle.

Two most common lifecycle methods are: `constructor()` and `render()`.

`constructor()` is the first method called during the mounting phase. `render()` is called later during the mounting phase, to render the component for the first time, and during the updating phase, to re-render the component.

componentDidMount()

Lets say we are making a clock component which shows the current time. But what use would it be if after showing the intial time when the component mounted it does not updates to current time.

Lets say we have made a function which will update the clock state per second but where should we place this function so that it is rendered onto the screen.

`render()` is not a good place as it can create bugs and executes during mounting and updating phase – too often for us.

`constructor()` is also not a good place. Yes it executes only during mounting phase but side effects should be avoided in constructors because it violates something called the Single Responsibility Principle.

So where?

`componentDidMount()` is the final method called during the mounting phase.

The order is :

1. `constructor()`
2. `getDerivedStateFromProps()`   {rarely used}
3. `render()`
4. `componentDidMount()`

```
import React from 'react';
import ReactDOM from 'react-dom';

class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }
  render() {
    return <div>{this.state.date.toLocaleTimeString()}</div>;
  }
  componentDidMount() {
    const oneSecond = 1000;
setInterval(() => {
  this.setState({ date: new Date() });
}, oneSecond);
  }
}

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

componentWillUnmount()

When the component is unmounted, and we have a component that has some function updating its state constantly in set intervals, it will keep doing so even after the component has been removed from the screen. This will hurt the performance of the app.

React will return a warning as:

```
Warning: Can't perform a React state update on an unmounted component. This is a no-op, but it indicates a memory leak in your application. To fix, cancel all subscriptions and asynchronous tasks in the componentWillUnmount method.
```

In general, when a component produces a side-effect, one should remember to clean it up.

JS gives us the `clearInterval()` function. `setInterval()` can return an ID, which can be passed into `clearInterval()` to clear it.

`componentWillUnmount()` is called in the unmounting phase, right before the component is completely destroyed. It’s a useful time to clean up any of the component’s side effects.

```
import React from 'react';

export class Clock extends React.Component {
  constructor(props) {
    super(props);
    this.state = { date: new Date() };
  }
  render() {
    return <div>{this.state.date.toLocaleTimeString()}</div>;
  }
  componentDidMount() {
    const oneSecond = 1000;
    this.intervalID =  setInterval(() => {
      this.setState({ date: new Date() });
    }, oneSecond);
  }
  componentWillUnmount() {
    clearInterval(this.intervalID);
  }
}
```

componentDidUpdate()

An update is caused by changes to props or state. Every time we called `setState()` with a new data, we have triggered an update. Every time we change the props passed to a component, it caused an update.

When a component updates it calls several methods but the most commonly used ones are: `render()` and `componentDidUpdate()`.