Schedule

Today:

- React Component Lifecycle
- React Events
- React CSS

Read more:

- React Forms

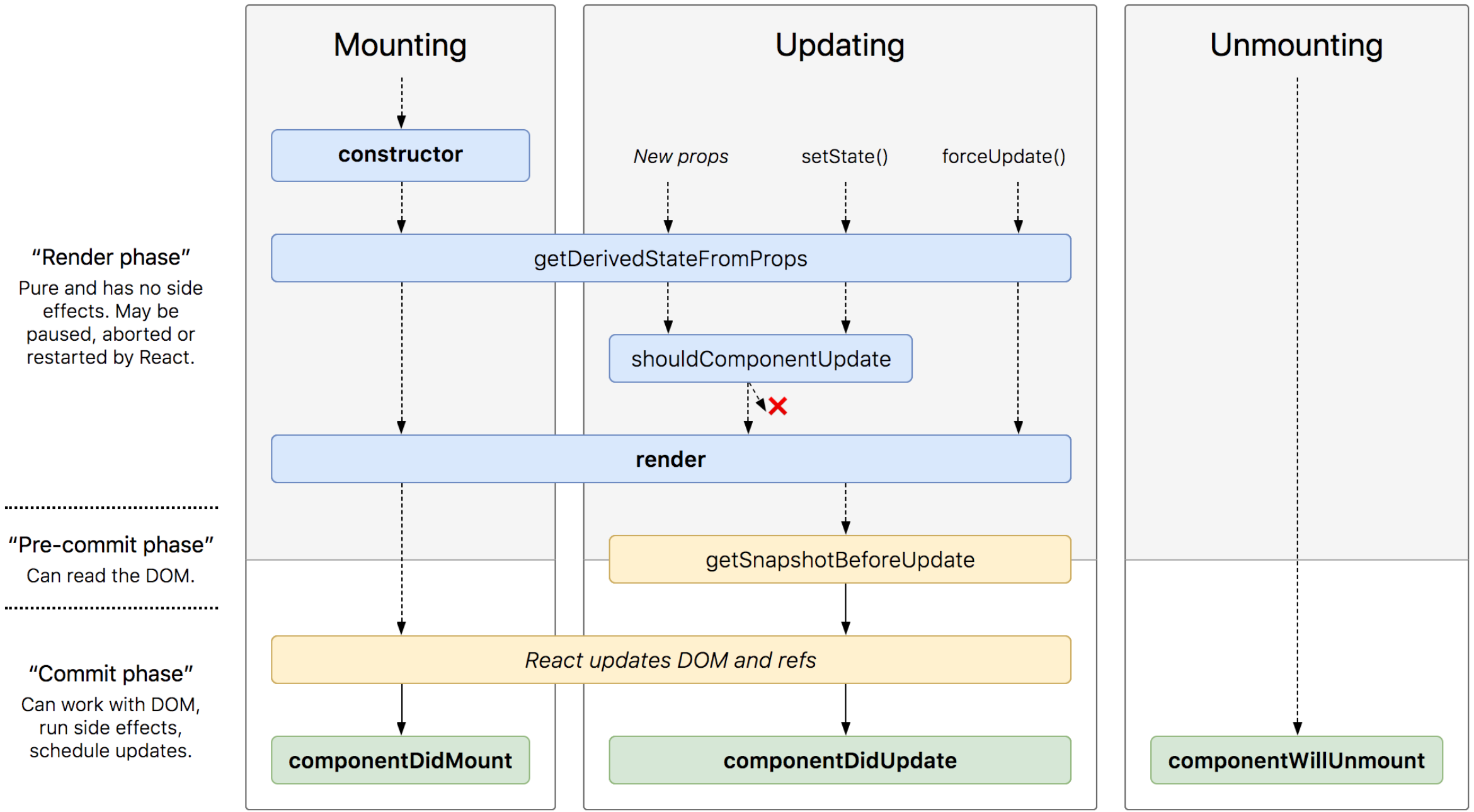
React Component Lifecyle

- Each component in React has a lifecycle which you can monitor and manipulate during its three main phases.

- The three phases are:
  - **Mounting**,
  - **Updating**,
  - and **Unmounting**.

Read:
https://www.w3schools.com/react/react_lifecycle.asp

React version 16.4 Language en-US



Mounting

- Mounting = Putting elements into the DOM
- Calling 04 built-in methods in this order:

1. `constructor()`
2. `getDerivedStateFromProps()`
3. `render()`
4. `componentDidMount()`

- The `render()` method is required and will always be called,
- The others are optional and will be called if you define them.

Constructor

- Is called, by React, before anything else every time you make a component
  - **to set up the initial `state` and other initial values**
- should always start by calling the `super(props)` to inherit methods from its parent (`React.Component`)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

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

getDerivedStateFromProps

- Is called right before rendering the element(s) in the DOM
  - **to set the `state` object based on the initial `props`**
- Takes `state` as an argument, and returns an object with changes to the `state`.

e.g. favorite color = "red“
→`getDerivedStateFromProps()`
→ favorite color = `favcol` attribute

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));
```

render

- Is **required**,
- Is the method that actual outputs HTML to the DOM.

```
class Header extends React.Component {
  render() {
    return (
      <h1>This is the content of the Header component</h1>
    );
  }
}

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

componentDidMount

- Is called after the component is rendered

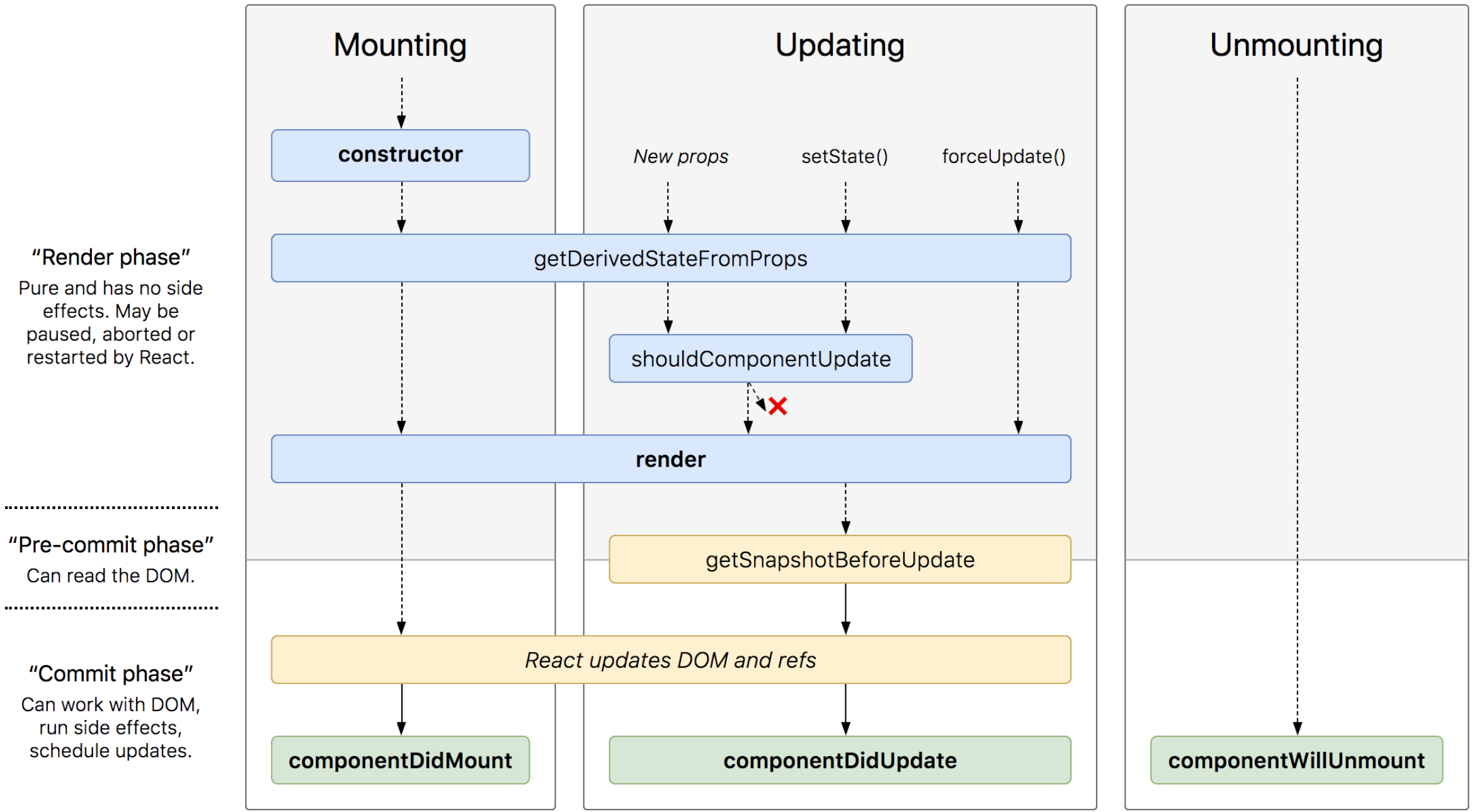
→ **to run statements that requires that the component is already placed in the DOM**

e.g. At first my favorite color is **red**, but give me a second, and it is **yellow** instead (w3schools)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  render() {
    return (
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
    );
  }
}

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

React version 16.4 Language en-US



Updating

- Component is updated whenever there is a **change** in the component's state or props.

- Calling 05 built-in methods in this order:

  1. `getDerivedStateFromProps()`
  2. `shouldComponentUpdate()`
  3. `render()`
  4. `getSnapshotBeforeUpdate()`
  5. `componentDidUpdate()`

  - The `render()` method is required and will always be called,
  - The others are optional and will be called if you define them.

getDerivedStateFromProps

- The first method is called when a component gets updated.
  - **to set the `state` object based on the initial `props`**

e.g. Button click → favorite color = blue, **BUT** `getDerivedStateFromProps()` → updates the state with the color from the `favcol` attribute,

→ favorite color is still rendered as yellow (w3schools)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  static getDerivedStateFromProps(props, state) {
    return {favoritecolor: props.favcol };
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}

ReactDOM.render(<Header favcol="yellow"/>, document.getElementById('root'));
```

shouldComponentUpdate

- Return a Boolean value that specifies whether React should continue with the rendering or not.
  - default = true.

e.g. Stop the component from rendering at any update (w3schools)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  shouldComponentUpdate() {
    return false;
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}

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

render

- Of course, called when a component gets updated,
  → to re-render the HTML to the DOM, with the new changes

e.g. Button click
→ favorite color = blue

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  changeColor = () => {
    this.setState({favoritecolor: "blue"});
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <button type="button" onClick={this.changeColor}>Change color</button>
      </div>
    );
  }
}

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

getSnapshotBeforeUpdate

- In this method, you have access to the props and state before the update,
  → even after the update, you can check what the values were before the update
- Must also include the `componentDidUpdate()` method

e.g.

- Mounting: favorite color = "red".
- Mounted: a timer changes the state (after 1s), favorite color = "yellow".

→ This triggers the update phase

→ `getSnapshotBeforeUpdate()` writes a message to the empty DIV1 element.

→ `componentDidUpdate()` writes a message in the empty DIV2 element

(w3schools)

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  getSnapshotBeforeUpdate(prevProps, prevState) {
    document.getElementById("div1").innerHTML =
    "Before the update, the favorite was " + prevState.favoritecolor;
  }
  componentDidUpdate() {
    document.getElementById("div2").innerHTML =
    "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
        <h1>My Favorite Color is {this.state.favoritecolor}</h1>
        <div id="div1"></div>
        <div id="div2"></div>
      </div>
    );
  }
}

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

componentDidUpdate

- Is called after the component is updated in the DOM

e.g.
- Mounting: favorite color = "red"
- Mounted: a timer changes the state (after 1s), favorite color = "yellow"

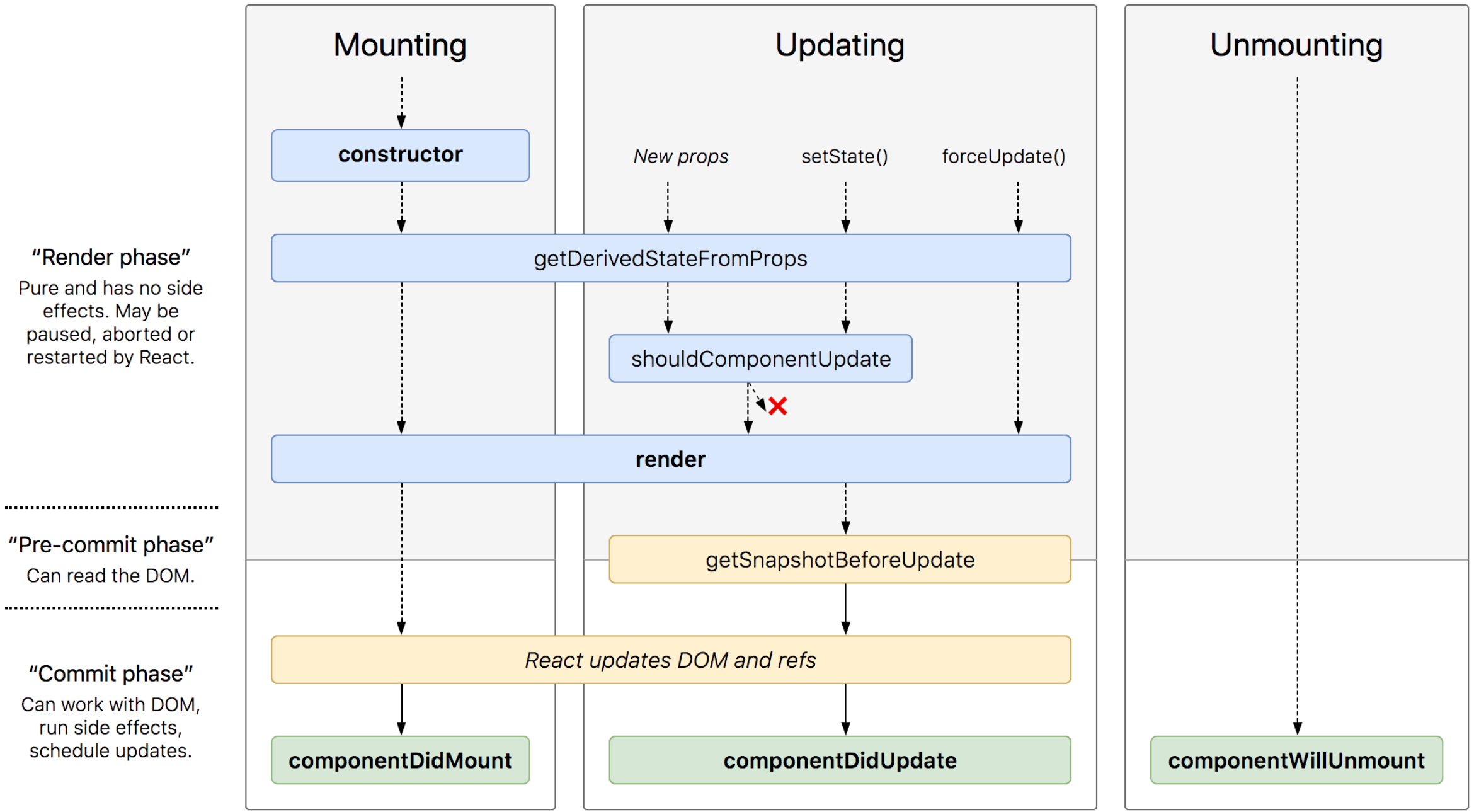
→ This triggers the update phase

→ `componentDidUpdate()` writes a message in the empty DIV element

```
class Header extends React.Component {
  constructor(props) {
    super(props);
    this.state = {favoritecolor: "red"};
  }
  componentDidMount() {
    setTimeout(() => {
      this.setState({favoritecolor: "yellow"})
    }, 1000)
  }
  componentDidUpdate() {
    document.getElementById("mydiv").innerHTML =
    "The updated favorite is " + this.state.favoritecolor;
  }
  render() {
    return (
      <div>
      <h1>My Favorite Color is {this.state.favoritecolor}</h1>
      <div id="mydiv"></div>
      </div>
    );
  }
}

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

React version 16.4 Language en-US



componentWillUnmount

- Unmounting = when a component is removed from the DOM
- Calling only 01 built-in method:
  1. `componentWillUnmount()`

e.g.

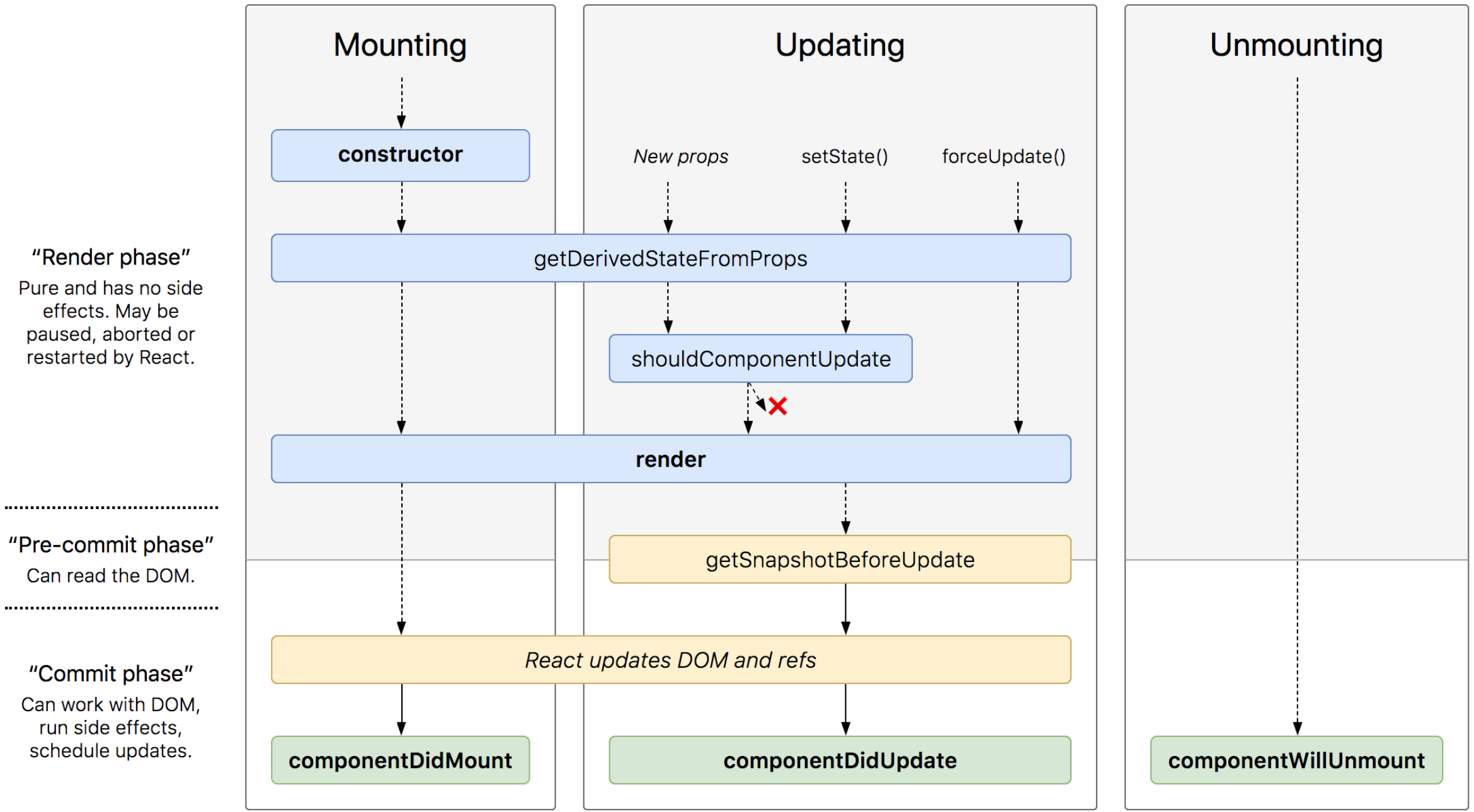
- Click button to delete header

```
class Container extends React.Component {
  constructor(props) {
    super(props);
    this.state = {show: true};
  }
  delHeader = () => {
    this.setState({show: false});
  }
  render() {
    let myheader;
    if (this.state.show) {
      myheader = <Child />;
    };
    return (
      <div>
      {myheader}
      <button type="button" onClick={this.delHeader}>Delete Header</button>
      </div>
    );
  }
}

class Child extends React.Component {
  componentWillUnmount() {
    alert("The component named Header is about to be unmounted.");
  }
  render() {
    return (
      <h1>Hello World!</h1>
    );
  }
}

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

React version 16.4 Language en-US



React events

- Just like HTML, React can perform actions based on user events.
  - Similar events as HTML: `click`, `change`, `mouseover` etc.

- **Adding events**
  - React events are written in **camelCase** syntax
    - `onClick` instead of `onclick`.

  - React event handlers are written inside curly braces
    - `onClick={shoot}` instead of `onClick="shoot()"`

HTML

```
<button onclick="shoot()">Take the Shot!</button>
```

React

```
<button onClick={shoot}>Take the Shot!</button>
```

React events

- **Event handlers**
  - Good practice: to put the event handler as a method in the component class

```
class Football extends React.Component {
  shoot() {
    alert("Great Shot!");
  }
  render() {
    return (
      <button onClick={this.shoot}>Take the shot!</button>
    );
  }
}

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

React events

- **Bind `this`?**
  - with regular functions the `this` = the object that called the method
    - global window object, a HTML button, etc.

  → Use the `bind()` method

  OR

  → Use arrow functions - `this` = **always** the object that defined the arrow function

Passing Arguments

- **02 options** to send parameters into an event handler:

1. Make an anonymous arrow function:

e.g. Send "Goal" as a parameter to the shoot function, using arrow function:

```
class Football extends React.Component {
  shoot = (a) => {
    alert(a);
  }
  render() {
    return (
      <button onClick={() => this.shoot("Goal")}>Take the shot!</button>
    );
  }
}

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

Passing Arguments

- **02 options** to send parameters into an event handler:

1. Make an anonymous arrow function:

OR

2. Bind the event handler to `this`.

Note: the first argument has to be `this`.

```
class Football extends React.Component {
  shoot(a) {
    alert(a);
  }
  render() {
    return (
      <button onClick={this.shoot.bind(this, "Goal")}>Take the shot!</button>
    );
  }
}

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

React Event Object

- Event handlers have access to the React event that triggered the function
  - **With** arrow function, you have to send the event argument *manually*
  - **Without** arrow function, the React event object is sent *automatically* as the last argument when using the bind() method

```
class Football extends React.Component {
  shoot = (a, b) => {
    alert(b.type);
    /*
    'b' represents the React event that triggered the function,
    in this case the 'click' event
    */
  }
  render() {
    return (
      <button onClick={(ev) => this.shoot("Goal", ev)}>Take the shot!</button>
    );
  }
}

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

React CSS

- There are many ways to style React with CSS, this tutorial will take a closer look at **inline styling**, and **CSS stylesheet**.
- Inline styling:
  - the value must be a JavaScript object

```
class MyHeader extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{color: "red"}}>Hello Style!</h1>
      <p>Add a little style!</p>
      </div>
    );
  }
}
```

- **Note**: In JSX, JavaScript expressions are written inside curly braces, and since JavaScript objects also use curly braces → {{}}

JavaScript Object

- You can also create an object with styling information, and refer to it in the style attribute

```
class MyHeader extends React.Component {
  render() {
    const mystyle = {
      color: "white",
      backgroundColor: "DodgerBlue",
      padding: "10px",
      fontFamily: "Arial"
    };
    return (
      <div>
      <h1 style={mystyle}>Hello Style!</h1>
      <p>Add a little style!</p>
      </div>
    );
  }
}
```

camelCased Property Names

- Since the inline CSS is written in a JavaScript object, properties with two names, like background-color, must be written with camel case syntax

```
class MyHeader extends React.Component {
  render() {
    return (
      <div>
      <h1 style={{backgroundColor: "lightblue"}}>Hello Style!</h1>
      <p>Add a little style!</p>
      </div>
    );
  }
}
```

CSS Stylesheet

- Write your CSS styling in a separate .css file → import it in your application

App.css

```css
body {
  background-color: #282c34;
  color: white;
  padding: 40px;
  font-family: Arial;
  text-align: center;
}
```

index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './App.css';

class MyHeader extends React.Component {
  render() {
    return (
      <div>
      <h1>Hello Style!</h1>
      <p>Add a little style!.</p>
      </div>
    );
  }
}

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

React Forms

- Read more: w3schools

Next week: **Wrap Up!**