

React Cheat Sheet

Components

```
import React from 'react'
import ReactDOM from 'react-dom'
class Hello extends React.Component {
  render () {
    return <div className='message-box'>
      Hello {this.props.name}
    </div>
  }
}
const el = document.body
ReactDOM.render(<Hello name='John' />, el)
```

Import multiple exports

```
import React, {Component} from 'react'
import ReactDOM from 'react-dom'
class Hello extends Component {
  ...
}
```

Properties

```
<Video fullscreen={true} autoplay={false} />
render () {
  this.props.fullscreen
  const { fullscreen, autoplay } = this.props
  ...
}
```

States

```
constructor(props) {
  super(props)
  this.state = { username: undefined }
}
this.setState({ username: 'rstacruz' })
render () {
  this.state.username
  const { username } = this.state
  ...
}
```

Nesting

```
class Info extends Component {  
  render () {  
    const { avatar, username } = this.props  
  
    return <div>  
      <UserAvatar src={avatar} />  
      <UserProfile username={username} />  
    </div>  
  }  
}
```

As of React v16.2.0, fragments can be used to return multiple children without adding extra wrapping nodes to the DOM.

```
import React, {  
  Component,  
  Fragment  
} from 'react'  
  
class Info extends Component {  
  render () {  
    const { avatar, username } = this.props  
  
    return (  
      <Fragment>  
        <UserAvatar src={avatar} />  
        <UserProfile username={username} />  
      </Fragment>  
    )  
  }  
}
```

Setting default props

```
Hello.defaultProps = {  
  color: 'blue'  
}
```

Setting default state

```
class Hello extends Component {
  constructor (props) {
    super(props)
    this.state = { visible: true }
  }
}
```

References

```
class MyComponent extends Component {
  render () {
    return <div>
      <input ref={el => this.input = el} />
    </div>
  }

  componentDidMount () {
    this.input.focus()
  }
}
```

DOM Events

```
class MyComponent extends Component {
  render () {
    <input type="text"
      value={this.state.value}
      onChange={event => this.onChange(event)} />
  }

  onChange (event) {
    this.setState({ value: event.target.value })
  }
}
```

Functional components

```
function MyComponent ({ name }) {
  return <div className='message-box'>
    Hello {name}
  </div>
}
```

Functional components have no state. Also, their props are passed as the first parameter to a function.

Pure components

```
import React, {PureComponent} from 'react'

class MessageBox extends PureComponent {
  ...
}
```

Performance-optimized version of `React.Component`. Doesn't rerender if props/state hasn't changed.

Component API

```
this.forceUpdate()
this.setState({ ... })
this.setState(state => { ... })
this.state
this.props
```

These methods and properties are available for Component instances.

Defaults

Setting default props

```
Hello.defaultProps = {
  color: 'blue'
}
```

Setting default state

```
class Hello extends Component {
  constructor (props) {
    super(props)
    this.state = { visible: true }
  }
}
```

Lifecycle

Mounting

<code>constructor (props)</code>	Before rendering #
<code>componentWillMount()</code>	Don't use this #
<code>render()</code>	Render #
<code>componentDidMount()</code>	After rendering (DOM available) #
<code>componentWillUnmount()</code>	Before DOM removal #
<code>componentDidCatch()</code>	Catch errors (16+) #
Set initial the state on <code>constructor()</code> . Add DOM event handlers, timers (etc) on <code>componentDidMount()</code> , then remove them on <code>componentWillUnmount()</code> .	

Updating

<code>componentDidUpdate (prevProps, prevState, snapshot)</code>	Use <code>setState()</code> here, but remember to compare props
<code>shouldComponentUpdate (newProps, newState)</code>	Skips <code>render()</code> if returns false
<code>render()</code>	Render
<code>componentDidUpdate (prevProps, prevState)</code>	Operate on the DOM here
Called when parents change properties and <code>.setState()</code> . These are not called for initial renders.	