React.js cheatsheet

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)
```

Properties

```
<Video fullscreen={true} />
```

```
render () {
  this.props.fullscreen
  ...
}
```

Use `this.props` to access properties passed to the component.

States

```
constructor(props) {
  super(props)
  this.state = {}
}
```

```
this.setState({ username: 'rstacruz' })
```

```
render () {
  this.state.username
  ...
}
```

Use states (`this.state`) to manage dynamic data.

Nesting

```
class Info extends React.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.

```
class Info extends React.Component {
  render () {
    const { avatar, username } = this.props

    return (
      <React.Fragment>
        <UserAvatar src={avatar} />
        <UserProfile username={username} />
      </React.Fragment>
    )
  }
}
```

Nest components to separate concerns.

Children

```
<AlertBox>
  <h1>You have pending notifications</h1>
</AlertBox>
```

```
class AlertBox extends React.Component {
  render () {
    return <div className='alert-box'>
      {this.props.children}
    </div>
  }
}
```

Children are passed as the `children` property.

Defaults

Setting default props

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

Setting default state

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

Set the default state in the `constructor()`.

Other components

Function 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

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

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

Component API

```
this.forceUpdate()
```

```
this.setState({ ... })
```

```
this.state
this.props
```

These methods and properties are available for `Component` instances.

Lifecycle

Mounting

`constructor` (props)	Before rendering #
`componentWillMount()`	Don't use this #
`render()`	Render #
`componentDidMount()`	After rendering (DOM available) #
`componentWillUnmount()`	Before DOM removal #
`componentDidCatch()`	Catch errors (16+) #

Set initial the state on `constructor()`. Add DOM event handlers, timers (etc) on `componentDidMount()`, then remove them on `componentWillUnmount()`.

Updating

`componentWillReceiveProps` (newProps)	Use `setState()` here
`shouldComponentUpdate` (newProps, newState)	Skips `render()` if returns false
`componentWillUpdate` (newProps, newState)	Can't use `setState()` here
`render()`	Render
`componentDidUpdate` (prevProps, prevState)	Operate on the DOM here

Called when parents change properties and `.setState()`. These are not called for initial renders.

DOM nodes

References

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

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

Allows access to DOM nodes.

DOM Events

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

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

Pass functions to attributes like onChange.

Other features

Transferring props

```
<VideoPlayer src="video.mp4" />
```

```
class VideoPlayer extends React.Component {
  render () {
    return <VideoEmbed {...this.props} />
  }
}
```

Propagates `src="..."` down to the sub-component.

Top-level API

```
React.createClass({ ... })
React.isValidElement(c)
```

```
ReactDOM.render(<Component />, domnode, [callback])
ReactDOM.unmountComponentAtNode(domnode)
```

```
ReactDOMServer.renderToString(<Component />)
ReactDOMServer.renderToStaticMarkup(<Component />)
```

There are more, but these are most common.

JSX patterns

Style shorthand

```
var style = { height: 10 }
return <div style={style}></div>
```

```
return <div style={{ margin: 0, padding: 0 }}></div>
```

Inner HTML

```
function markdownify() { return "<p>...</p>"; }
<div dangerouslySetInnerHTML={{__html: markdownify()}} />
```

Conditionals

```
<div>
  {showMyComponent
    ? <MyComponent />
    : <OtherComponent />}
</div>
```

Short-circuit evaluation

```
<div>
  {showPopup && <Popup />}
</div>
```

Lists

```
class TodoList extends React.Component {
  render () {
    const { items } = this.props

    return <ul>
      {items.map(item =>
        <TodoItem item={item} key={item.key} />)}
    </ul>
  }
}
```

Always supply a key property.

New features

Returning multiple elements

You can return multiple elements as arrays or fragments.

Arrays

```
render () {
  // Don't forget the keys!
  return [
    <li key="A">First item</li>,
    <li key="B">Second item</li>
  ]
}
```

Fragments

```
render () {
  // Fragments don't require keys!
  return (
    <React.Fragment>
      <li>First item</li>
      <li>Second item</li>
    </React.Fragment>
  )
}
```

Returning strings

```
render() {
  return 'Look ma, no spans!';
}
```

You can return just a string.

Portals

```
render () {
  return React.createPortal(
    this.props.children,
    document.getElementById('menu')
  )
}
```

This renders `this.props.children` into any location in the DOM.

Errors

```
class MyComponent extends React.Component {
  ...
  componentDidCatch (error, info) {
    this.setState({ error })
  }
}
```

Catch errors via `componentDidCatch`. (React 16+)

Hydration

```
const el = document.getElementById('app')
ReactDOM.hydrate(<App />, el)
```

Use `ReactDOM.hydrate` instead of using `ReactDOM.render` if you're rendering over the output of ReactDOMServer.