DEVHINTS.IO

Edit

React.js cheatsheet

React is a JavaScript library for building user interfaces. This guide targets React v15 to v16.

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 () {

  ...
}
```

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

See: Properties

States

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

```
this.setState({ username:
```

```
render () {

  ...
}
```

Use states (`this.state`) to mai

Nesting

Use the React.js jsfiddle to start hacking. (or the unofficial jsbin)

```
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 chi
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.

See: Composing Components

Children

```
<AlertBox>

</AlertBox>
```

```
class AlertBox extends Rea
  render () {
    return <div className=

    </div>
  }
}
```

Children are passed as the chil

Defaults

Setting default props

```
  color: 'blue'
}
```

See: defaultProps

Setting default state

```
class Hello extends React.Component {
  constructor (props) {
    super(props)

  }
}
```

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

See: Setting the default state

Other components

Function components

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

Pure components

```
  ...
}
```

Component API

```
this.forceUpdate()
```

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

Functional components have no state. Also, their `props` are pass
first parameter to a function.

See: Function and Class Components

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

See: Pure components

`this.state`
`this.props`

These methods and properties

See: Component API

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 `setSt`
`shouldComponentUpdate` (newProps, newState)	Skips r re
`componentWillUpdate` (newProps, newState)	Can't use s
`render()`	
`componentDidUpdate` (prevProps, prevState)	Operate o

Called when parents change properties and `.setState()`. These called for initial renders.

See: Component specs

DOM nodes

References

```
class MyComponent extends React.Component {
  render () {
    return <div>

    </div>
  }

  componentDidMount () {

  }
}
```

Allows access to DOM nodes.

See: Refs and the DOM

DOM Events

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

  }

  onChange (event) {

  }
}
```

Pass functions to attributes like onChange.

See: Events

Other features

Transferring props

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

```
class VideoPlayer extends React.Component {
  render () {
```

Top-level API

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

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

```
  }
}
```

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

See Transferring props

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

There are more, but these are most common.

See: React top-level API

JSX patterns

Style shorthand

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

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

See: Inline styles

Inner HTML

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

See: Dangerously set innerHTML

Conditionals

```
<div>
  {showMyComponent
    ? <MyComponent />
```

Lists

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

    return <ul>
```

```
    : <OtherComponent />}
</div>
```

Short-circuit evaluation

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

```
    </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!



}
```

Fragments

```
render () {
  // Fragments don't require keys!
```

Returning strings

```
render() {

}
```

You can return just a string.

See: Fragments and strings

Errors

```
class MyComponent extends
  ...



}
```

Catch errors via componentDid

See: Error handling in React 16

Portals

```
render () {
```

Hydration

```
  ,
}
```

See: Fragments and strings

```
}
```

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

See: Portals

```
const el = document.getEl
```

Use `ReactDOM.hydrate` instead rendering over the output of Re

See: Hydrate

Property validation

PropTypes

```
import PropTypes from 'prop-types'
```

See: Typechecking with PropTypes

`any`

Basic

`string`

`number`

`func`

Basic types

```
MyComponent.propTypes = {
  email:      PropTypes.string,
  seats:      PropTypes.number,
  callback:   PropTypes.func,
  isClosed:   PropTypes.bool,
  any:        PropTypes.any
}
```

Enumerables (oneOf)

```
MyCo.propTypes = {
  direction: PropTypes.oneOf([
```

Required types

```
MyCo.propTypes = {
  name:  PropTypes.string
}
```

Elements

```
MyCo.propTypes = {
  // React element
  element: PropTypes.elem
```

```
string, element
```

`bool`	
Enum	
`oneOf`(any)	Enum types
`oneOfType`(type array)	Union
Array	
`array`	
`arrayOf`(...)	
Object	
`object`	
`objectOf`(...)	Object with values of a certain type
`instanceOf`(...)	Instance of a class
`shape`(...)	
Elements	
`element`	React element
`node`	DOM node
Required	
`(···).isRequired`	Required

```
    'left', 'right'
  ])
}
```

```
ropTypes.node
```

Custom validation

```
MyCo.propTypes = {
  customProp: (props, key, componentName) => {
    if (!/matchme/.test(props[key])) {
      return new Error('Validation failed!')
    }
  }
}
```

Arrays and objects

```
MyCo.propTypes = {
  ropTypes.array,
  ropTypes.arrayOf
  ropTypes.object,
  ropTypes.objectO
  : PropTypes.insta
```

```
Types = {
  user: PropTypes.shape({
    name: PropTypes.string
    age:  PropTypes.numbe
  })
}
```

```
Use .array[Of], .object[Of]
```

Also see

- React website (reactjs.org)
- React cheatsheet (reactcheatsheet.com)

Awesome React (github.com)

React v0.14 cheatsheet Legacy version

10 Comments for this cheatsheet. Write yours!

devhints.io / Search 368+ cheatsheets

Over 368 curated cheatsheets, by developers for developers.

Devhints home

Other React cheatsheets

- Redux cheatsheet
- Enzyme cheatsheet
- Enzyme v2 cheatsheet
- Awesome Redux cheatsheet
- Flux architecture cheatsheet
- React-router cheatsheet

Top cheatsheets

- Elixir cheatsheet
- ES2015+ cheatsheet
- Vim cheatsheet
- Vim scripting cheatsheet
- Capybara cheatsheet
- Date & time formats cheatsheet