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)
```

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

Properties

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

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

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

States

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

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

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

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.

Nesting

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

    return <div>
      <UserAvatar src={avatar} />
      <UserProfile username={username} />
    </div>
  }
}
```

Nest components to separate concerns.

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()}} />
```

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.

Conditionals

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

Short-circuit evaluation

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

New features

Returning fragments

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

You can return multiple nodes as arrays.

Returning strings

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

You can return just a string.

Errors

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

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

Portals

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

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

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.

Property validation

PropTypes

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

See: Typechecking with PropTypes

`any`	Anything
Basic	
`string`	
`number`	
`func`	Function
`bool`	True or false
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

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

Custom validation

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

Required types

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

Elements

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

  // num, string, element, or an array of those
  node: PropTypes.node
}
```

Arrays and objects

```
MyCo.propTypes = {
  list: PropTypes.array,
  ages: PropTypes.arrayOf(PropTypes.number),
  user: PropTypes.object,
  user: PropTypes.objectOf(PropTypes.number),
  message: PropTypes.instanceOf(Message)
}
```

```
MyCo.propTypes = {
  user: PropTypes.shape({
    name: PropTypes.string,
    age:  PropTypes.number
  })
}
```

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