

# 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 Reactjs jsfiddle to start hacking. (or the unofficial jsbin)
```

## Children

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

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

Children are passed as the `children` property.

## Import multiple exports

```
import React, {Component} from 'react'
import ReactDOM from 'react-dom'

class Hello extends Component {
  ...
}
```

## States

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

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

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

Use `this.state` to manage dynamic data.

With Babel you can use proposal-class-fields and get rid of constructor

```
class Hello extends Component {
  state = { username: undefined }
  ...
}
```

See: States

## Properties

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

render () {
  this.props.fullscreen
  const { fullscreen, autoplay } = this.props
  ...
}
```

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

See: Properties

## 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>
    )
  }
}
```

Nest components to separate concerns.

See: Composing Components

## # Defaults

### Setting default props

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

See: `defaultProps`

### Setting default state

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

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

And without constructor using Babel with proposal-class-fields.

```
class Hello extends Component {
  state = { visible: true }
}
```

See: [Setting the default state](#)

## # Other components

### 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.

See: [Function and Class Components](#)

### 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.

See: [Pure components](#)

### Component API

```
this.forceUpdate()
```

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

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

These methods and properties are available for Component instances.

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

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

See: [Component specs](#)

## # Hooks (New)

### State Hook

```
import React, { useState } from 'react';

function Example() {
  // Declare a new state variable, which we'll call "count"
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

Hooks are a new addition in React 16.8.

See: [Hooks at a Glance](#)

### Building your own hooks

Define FriendStatus

```
import React, { useState, useEffect } from 'react';

function FriendStatus(props) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      // ...
    }
  });
}
```

### Declaring multiple state variables

```
function ExampleWithManyStates() {
  // Declare multiple state variables!
  const [age, setAge] = useState(42);
  const [fruit, setFruit] = useState('banana');
  const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);
  // ...
}
```

### Effect hook

```
import React, { useState, useEffect } from 'react';

function Example() {
  const [count, setCount] = useState(0);

  // Similar to componentDidMount and componentDidUpdate:
  useEffect(() => {
    // Update the document title using the browser API
    document.title = `You clicked ${count} times`;
  }, [count]);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button onClick={() => setCount(count + 1)}>
        Click me
      </button>
    </div>
  );
}
```

```

        setisOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(props.friend.id, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(props.friend.id, handleStatusChange);
    };
}, [props.friend.id]);

if (isOnline === null) {
  return 'Loading...';
}
return isOnline ? 'Online' : 'Offline';
}

```

Effects may also optionally specify how to "clean up" after them by returning a function.

#### Use FriendStatus

```

function FriendStatus(props) {
  const isOnline = useFriendStatus(props.friend.id);

  if (isOnline === null) {
    return 'Loading...';
  }
  return isOnline ? 'Online' : 'Offline';
}

```

See: Building Your Own Hooks

If you're familiar with React class lifecycle methods, you can think of `useEffect` Hook as `componentDidMount`, `componentDidUpdate`, and `componentWillUnmount` combined.

By default, React runs the effects after every render — including the first render.

## Hooks API Reference

Also see: Hooks FAQ

Basic Hooks

`useState(initialState)`

`useEffect(() => { ... })`

`useContext(MyContext)` value returned from `React.createContext`

Full details: Basic Hooks

Additional Hooks

`useReducer(reducer, initialArg, init)`

`useCallback(() => { ... })`

`useMemo(() => { ... })`

`useRef(initialValue)`

`useImperativeHandle(ref, () => { ... })`

`useLayoutEffect` identical to `useEffect`, but it fires synchronously after all DOM mutations

`useDebugValue(value)` display a label for custom hooks in React DevTools

Full details: Additional Hooks

## # DOM nodes

### References

```

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

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

```

Allows access to DOM nodes.

See: Refs and the DOM

### 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 })
  }
}

```

Pass functions to attributes like `onChange`.

See: Events

## # Other features

### Transferring props

```

<VideoPlayer src="video.mp4" />

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

```

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

See Transferring props

### 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.

See: React top-level API

# # JSX patterns

## Style shorthand

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

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

See: [Inline styles](#)

## Conditionals

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

## Short-circuit evaluation

```
<Fragment>
  {showPopup && <Popup />}
  ...
</Fragment>
```

## Inner HTML

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

See: [Dangerously set innerHTML](#)

## Lists

```
class TodoList extends 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 (
    <Fragment>
      <li>First item</li>
      <li>Second item</li>
    </Fragment>
  )
}
```

See: [Fragments and strings](#)

## Returning strings

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

You can return just a string.

See: [Fragments and strings](#)

## Portals

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

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

See: [Portals](#)

## Errors

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

Catch errors via componentDidCatch. (React 16+)

See: [Error handling in 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.

See: [Hydrate](#)

# # Property validation

## PropTypes

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

See: [Typechecking with PropTypes](#)

any

Anything

Basic

string

number

func

Function

## 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.isRequired
}
```

## Elements

```
MyCo.propTypes = {
  // React element
  element: PropTypes.element,
  // num, string, element, or an array of those
  node: PropTypes.node
}
```

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

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

### Custom validation

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

### Arrays and objects

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

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

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

## # Also see

- [React website](#) (reactjs.org)
- [React cheatsheet](#) (reactcheatsheet.com)
- [Awesome React](#) (github.com)
- [React v0.14 cheatsheet](#) Legacy version



▶ [31 Comments](#) for this cheatsheet. [Write yours!](#)

[devhints.io](#) / Search 352+ cheatsheets



Over 352 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

[Vimdiff](#) cheatsheet

[Vim](#) cheatsheet

[Vim scripting](#) cheatsheet

[Vue.js](#) cheatsheet