React Cheat Sheet

CREATE PROJECT

- With next.js: npx create-next-app@latest
- npx create-react-app my-app

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)
```

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.

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
  ...
}
```

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

React Cheat Sheet

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.

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>
    )
  }
}
```

Declarar props en componente hijo

```
export const TravelDetail = ({currentTravel}: TravelDetailProps) =>
```

Pasar props a componente hijo

```
<TravelDetail currentTravel={travel}></TravelDetail>
```

Declarar propiedad que emite en hijo

```
export default function Child({childToParent}) {}
```

Emitir en hijo

```
<Button primary onClick={() => childToParent(data)}>Click Child</Button>
```

Escuchar en el padre

```
<Child childToParent={childToParent}/>
```

React Cheat Sheet

States

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

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

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

Use states (`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 };
  ...
}
```

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 }
}
```

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>
  );
}
```

Declaring multiple state variables

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

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

React Cheat Sheet

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>
  );
}
```

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

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

Building your own hooks

A custom Hook is a JavaScript function whose name starts with ”use” and that may call other Hooks. For example, useFriendStatus below is our first custom Hook

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

function useFriendStatus(friendID) {
  const [isOnline, setIsOnline] = useState(null);

  useEffect(() => {
    function handleStatusChange(status) {
      setIsOnline(status.isOnline);
    }

    ChatAPI.subscribeToFriendStatus(friendID, handleStatusChange);
    return () => {
      ChatAPI.unsubscribeFromFriendStatus(friendID, handleStatusChange);
    };
  });

  return isOnline;
}
```

Using custom hook

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

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

React Cheat Sheet

DOM

DOM node

```
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 })
  }
}
```

Pass functions to attributes like onChange.

Other features

Transferring props

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

```
class VideoPlayer extends 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

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

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

Innerhtml

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

Use data in html tags

```
<span>{ currentTravel.country }</span>
```

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.

Conditionals

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

Short-circuit evaluation

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

React Cheat Sheet

Forms

Vincular un input en un form

```
<input
type="text"
id="country"
name="country"
value={currentTravel.country}
onChange={(e) => setCurrentTravel({...currentTravel,
country: e.target.value})}/>
```

Validaciones con react-hook-form

```
npm i react-hook-form

import React from "react"
import { useForm } from "react-hook-form"

export default function App() {
  const {
    register,
    handleSubmit,
    formState: { errors },
  } = useForm()
  const onSubmit = (data) => console.log(data)
```

```
  return (
    <form onSubmit={handleSubmit(onSubmit)}>
      <label htmlFor="name">Name</label>
      <input
        id="name"
        {...register("name", { required: true, maxLength: 30
})}
      />
      {errors.name && errors.name.type === "required" &&
(
        <span>This is required</span>
      )}
      {errors.name && errors.name.type === "maxLength"
&& (
        <span>Max length exceeded</span>
      )}
      <input type="submit" />
    </form>
  )
}
```

Zustand Store

```
import { create } from 'zustand'

const useStore = create((set) => ({
  count: 1,
  inc: () => set((state) => ({ count: state.count + 1 })),
}))

function Counter() {
  const { count, inc } = useStore()
  return (
    <div>
      <span>{count}</span>
      <button onClick={inc}>one up</button>
    </div>
  )
}
```

React Cheat Sheet

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 () {
  return (
    <Fragment>
      <li>First item</li>
      <li>Second item</li>
    </Fragment>
  )
}
```

Returning strings

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

Errors

```
class MyComponent extends 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.

React Cheat Sheet

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.