React Cheat Sheet

Last Updated : 17 Oct, 2024

React is an open-source JavaScript library used to create user interfaces in a declarative and efficient way. It is a component-based front-end library responsible only for the view layer of a **Model View Controller (MVC)** architecture. React is used to create modular user interfaces and promotes the development of reusable UI components that display dynamic data.

React Cheat Sheet

The react cheat sheet provides you simple and quick references to commonly used react methods. This single page contains all the important concepts and features of react required for performing all the basic tasks in React. It's a great resource for both beginners and experienced developers to quickly look up React essentials.

- JSX
- React Elements
- ReactJS Import and Export
- React Components
- Lifecycle of Components
- Conditional Rendering
- React Lists
- React DOM Events
- React Hooks
- PropTypes

Basic Setup

Follow the below steps to create a boilerplate

Step 1: Create the application using the command

```
npx create-react-app <<Project_Name>>
```

Step 2: Navigate to the folder using the command

```
cd <<Project_Name>>
```

Step 3: Open the App.js file and write the below code

Related searches

```
import React from 'react';
import './App.css';
export default function App() {
    return (
        <div >
            Hello Geeks
            Lets start learning React
        </div>
    )
}
```

JSX

JSX stands for **JavaScript XML**. JSX is basically a syntax extension of JavaScript. It helps us to write HTML in JavaScript and forms the basis of React Development. Using JSX is not compulsory but it is highly recommended for programming in React as it makes the development process easier as the code becomes easy to write and read.

Sample JSX code:

```
const ele = <h1>This is sample JSX</h1>;
```

React Elements

React elements are different from DOM elements as React elements are simple JavaScript objects and are efficient to create. React elements are the building blocks of any React app and should not be confused with React components.

React Element	Description	Syntax
Class Element Attributes	Passes attributes to an element. The major change is that class is changed to className	

React Element	Description	Syntax
	in double parenthesis like {{}}	
Fragments	Used to create single parent component	<>//Other Components</>

ReactJS Import and Export

In ReactJS we use importing and exporting to import already created modules and export our own components and modules rescpectively

Type of Import/Export	Description	Syntax
Importing Default exports	imports the default export from modules	import MOD_NAME from "PATH"
Importing Named Values	imports the named export from modules	import {NAME} from "PATH"
Multiple imports	Used to import multiple modules can be user defined of npm packages	import MOD_NAME, {NAME} from "PATH"
Default Exports	Creates one default export. Each component can have onne default export	export default MOD_NAME
Named Exports	Creates Named Exports when there are	export default {NAME}

Type of Import/Export	Description	Syntax
Multiple Exports	Exports mulitple named components	export default {NAME1, NAME2}

React Components

A Component is one of the core building blocks of React. Components in React basically return a piece of JSX code that tells what should be rendered on the screen.

Component	Description	Syntax
Functional	Simple JS functions and are stateless	function demoComponent() { return (<> // CODE </>); }
Class-based	Uses JS classes to create stateful components	class Democomponent extends React.Component { render() { return <>//CODE</>; } }
Nested	Creates component inside another component	function demoComponent() { return (<> <Another_Component/>

```
// Functional Component

export default function App() {
    return (
        <div >
            Hello Geeks
            Lets start learning React
        </div>
    )
}

// Class Component with nesting
class Example extends React.Component {
  render() {
    return (
          <div >
              <App/>
            Hello Geeks
            Lets start learning React
        </div>
     )
  }
}
```

Managing Data Inside and Outside Components(State and props)

Property	Description	Syntax

props	Passes data between components and is read-only. Mainly used in functional components	// Passing
		<Comp
		prop_name="VAL"/>
		//Accessing
		<Comp>
		{this.props.prop_name}
		</Comp>

state	Manages data inside a component and is	constructor(props) {
		super(props);
		this.state = {

Property	Description	Syntax
setState	Updates the value of a state using callback function. it is an asynchronous function call	this.setState((prevState)=> ({ // CODE LOGIC }))

```
const App = () => {
  const message = "Hello from functional component!";

  return (
    <div>
      <ClassComponent message={message} />
    </div>
  );
};

class ClassComponent extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      message: this.props.message
    };
  }

  render() {
    return (
      <div>
        <h2>Class Component</h2>
        <p>State from prop: {this.state.message}</p>
      </div>
    );
  }
}
```

Lifecycle of Components

The lifecycle methods in ReactJS are used to control the components at different stages from initialization till unmounting.

Mounting Phase methods

Method	Description	Syntax
constructor	Runs before component rendering	constructor(props){}
render	Used to render the component	render()
componentDidMount	Runs after component is rendered	componentDidMount()
componentWillUnmount	Runs before a component is removed from DOM	comoponentWillUnmount()
componentDidCatch	Used to catch errors in component	componentDidCatch()

Updating Phase Methods

Method	Description	Syntax
componentDidUpdate	Invokes after component is updated	componentDidUpdate(prevProp, prevState, snap)
	Used to avoid call in	shouldComponentUpdate(newPro

Method	Description	Syntax
render	Render component after update	render()

```
import React from 'react';
import ReactDOM from 'react-dom';

class Test extends React.Component {
    constructor(props) {
        super(props);
        this.state = { hello: "World!" };
    }

    componentWillMount() {
        console.log("componentWillMount()");
    }

    componentDidMount() {
        console.log("componentDidMount()");
    }

    changeState() {
        this.setState({ hello: "Geek!" });
    }

    render() {
        return (
            <div>
                <h1>GeeksForGeeks.org, Hello{this.state.hello}</h1>
                <h2>
                    <a onClick={this.changeState.bind(this)}>Press
Here!</a>
                </h2>
            </div>);
    }

    shouldComponentUpdate(nextProps, nextState) {
        console.log("shouldComponentUpdate()");
        return true;
    }

    componentWillUpdate() {
        console.log("componentWillUpdate()");
```

```
}

ReactDOM.render(
    <Test />,
    document.getElementById('root'));
```

Conditional Rendering

In React, conditional rendering is used to render components based on some conditions. If the condition is satisfied then only the component will be rendered. This helps in encapsulation as the user is allowed to see only the desired component and nothing else.

Type	Description	Syntax

if-else	Component is rendered using if-else block	if (condition) {
		return <COMP1 />;
		}else{
		return <COMP2/>;
		}

Logical && Operator	Used for showing/hiding single component based on condition	{condition && <Component/>}
Ternary Operator	Component is rendered using if-else block	{Condition ? <COMP1/> : <COMP2/> }

JavaScript JavaScript JavaScript

```
// Conditional Rendering Using if-else

import React from 'react';
import ReactDOM from 'react-dom';
```

```
        return null;
    else
        return <h1>Component is rendered</h1>;
}

ReactDOM.render(
    <div>
        <Example toDisplay = {true} />
        <Example toDisplay = {false} />
    </div>,
    document.getElementById('root')
);
```

React Lists

We can create lists in React in a similar manner as we do in regular JavaScript i.e. by storing the list in an array. In order to traverse a list we will use the map() function.

Keys are used in React to identify which items in the list are changed, updated, or deleted. Keys are used to give an identity to the elements in the lists. It is recommended to use a string as a key that uniquely identifies the items in the list.

Code Snippet:

```
const arr = [];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>
    {number}
  </li>
);
```

```
import React from 'react';
import ReactDOM from 'react-dom';

const numbers = [1,2,3,4,5];

const updatedNums = numbers.map((number)=>{
    return <li>{number}</li>;
});
```

```
    </ul>,
    document.getElementById('root')
);
```

React DOM Events

Similar to HTML events, React DOM events are used to perform events based on user inputs such as click, onChange, mouseOver etc

Method	Description	Syntax
Click	Triggers an event on click	<button **onClick**= {func}>CONTENT</button>
Change	Triggers when some change is detected in component	<input **onChange**= {handleChange} />
Submit	Triggers an event when form is submitted	<form onSubmit={(e) => {//LOGIC}}></form>

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

const App = () => {
// Counter is a state initialized to 0
const [counter, setCounter] = useState(0)

// Function is called everytime increment button is clicked
const handleClick1 = () => {
    // Counter state is incremented
    setCounter(counter + 1)
}

// Function is called everytime decrement button is clicked
const handleClick2 = () => {
    // Counter state is decremented
    setCounter(counter - 1)
}
```

```
            position: 'relative',
            top: '10vh',
        }}>
            {counter}
        </div>
        <div className="buttons">
            <button onClick={handleClick1}>Increment</button>
            <button onClick={handleClick2}>Decrement</button>
        </div>
    </div>
)
}

export default App
```

React Hooks

Hooks are used to give functional components an access to use the states and are used to manage side-effects in React. They were introduced React 16.8. They let developers use state and other React features without writing a class For example- State of a component It is important to note that hooks are not used inside the classes.

Hook	Description	Syntax
useState	Declares state variable inside a function	const [var, setVar] = useState(Val);
useEffect	Handle side effect in React	useEffect(<FUNCTION>, <DEPENDECY>)
useRef	Directly creates reference to DOM element	const refContainer = useRef(initialValue);
useMemo	Returns a memoized value	const memVal = useMemo(function, arrayDependencies)

```
function App() {
    const [click, setClick] = useState(0);
    // using array destructuring here
    // to assign initial value 0
    // to click and a reference to the function
    // that updates click to setClick
    return (
        <div>
            <p>You clicked {click} times</p>

            <button onClick={() => setClick(click + 1)}>
                Click me
            </button>
        </div>
    );
}

export default App;

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
<React.StrictMode>
    <App />
</React.StrictMode>
);
```

PropTypes

PropTypes in React are used to check the value of a prop which is passed into the component. These help in error hanling and are very useful in large scale applications.

Primitive Data Types

Type	Class/Syntax	Example
String	PropTypes.string	"Geeks"
Object	PropType.object	{course: "DSA"}
Number	PropType.number	15,

Type	Class/Syntax	Example
Function	PropType.func	const GFG ={return "Hello"}
Symbol	PropType.symbol	Symbol("symbole_here"

Array Types

Type	Class/Syntax	Example
Array	PropTypes.array	[]
Array of strings	PropTypes.arrayOf([type])	[15,16,17]
Array of numbers	PropTypes.oneOf([arr])	["Geeks", "For", "Geeks"
Array of objects	PropTypes.oneOfType([types])	PropTypes.instanceOf()

Object Types

Type	Class/Syntax	Example
Object	PropTypes.object()	{course: "DSA"}
Number Object	PropTypes.objectOf()	{id: 25}
Object Shape	PropTypes.shape()	{course: PropTypes.string, price:

Type	Class/Syntax	Example
Instance	PropTypes.objectOf()	new obj()

```
import PropTypes from 'prop-types';
import React from 'react';
import ReactDOM from 'react-dom/client';

// Component
class ComponentExample extends React.Component{
    render(){
        return(
                <div>

                    {/* printing all props */}
                    <h1>
                        {this.props.arrayProp}
                        <br />

                        {this.props.stringProp}
                        <br />

                        {this.props.numberProp}
                        <br />

                        {this.props.boolProp}
                        <br />
                    </h1>
                </div>
            );
    }
}

// Validating prop types
ComponentExample.propTypes = {
    arrayProp: PropTypes.array,
    stringProp: PropTypes.string,
    numberProp: PropTypes.number,
    boolProp: PropTypes.bool,
}

// Creating default props
ComponentExample.defaultProps = {

    arrayProp: ['Ram', 'Shyam', 'Raghav'],
    stringProp: "GeeksforGeeks",
    numberProp: "10",
```

```
<React.StrictMode>
    <ComponentExample />
</React.StrictMode>
);
```

Error Boundaries

Error Boundaries basically provide some sort of boundaries or checks on errors, They are React components that are used to handle JavaScript errors in their child component tree.

React components that catch JavaScript errors anywhere in their child component tree, log those errors, and display a fallback UI. It catches errors during rendering, in lifecycle methods, etc.

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

class ErrorBoundary extends Component {
    constructor(props) {
        super(props);
        this.state = { hasError: false };
    }

    componentDidCatch(error, info) {
        // Log the error to an error reporting service
        console.error('Error:', error);
        console.error('Info:', info);
        this.setState({ hasError: true });
    }

    render() {
        if (this.state.hasError) {
            // Fallback UI when an error occurs
            return <div>Something went wrong!</div>;
        }
        return this.props.children;
    }
}

export default ErrorBoundary;

//Apply Error Boundary

import React from 'react';
import ErrorBoundary from './ErrorBoundary';

function App() {
```

```
        </ErrorBoundary>
    );
}

export default App;
```

This React Cheat Sheet gives you quick access to the most commonly used React concepts and methods. From setting up project to managing state and handling events, everything to build dynamic and responsive user interfaces with React.

Next Article

HTML Complete Guide – A to Z
HTML Concepts

Python Tutorial

Python Examples
Django Tutorial
Python Projects
Python Tkinter
Web Scraping
OpenCV Tutorial
Python Interview Question

Computer Science

GATE CS Notes
Operating Systems
Computer Network
Database Management System
Software Engineering
Digital Logic Design
Engineering Maths

DevOps

Git
AWS
Docker
Kubernetes
Azure
GCP
DevOps Roadmap

System Design

High Level Design
Low Level Design
UML Diagrams
Interview Guide
Design Patterns
OOAD
System Design Bootcamp
Interview Questions

School Subjects

Mathematics
Physics
Chemistry
Biology
Social Science
English Grammar

Databases

SQL
MYSQL
PostgreSQL
PL/SQL
MongoDB

Preparation Corner

Company-Wise Recruitment Process
Aptitude Preparation
Puzzles
Company-Wise Preparation

More Tutorials

Software Development
Software Testing
Product Management
Project Management
Linux
Excel
All Cheat Sheets

Courses

IBM Certification Courses
DSA and Placements
Web Development
Data Science
Programming Languages

Programming Languages

C Programming with Data Structures
C++ Programming Course
Java Programming Course
Python Full Course