React.js Cheat Sheet in Simple Terms

React.js

React Home: The main portal for official React documentation, tutorials, and community resources.
React Intro: Introduction to React, a library for building interactive user interfaces.
Example : Simple React element rendered into the DOM.

```
ReactDOM.render(
  <h1>Hello, World!</h1>,
  document.getElementById('root')
);
```

React Get Started: Steps to start using React by setting up a new project with Create React App.
Command :

```
npx create-react-app my-app
```

React Upgrade: How to upgrade an existing React app to the latest version.
Command :

```
npm update react react-dom
```

Core Concepts

React ES6: Using ES6 features in React, like arrow functions, classes, and let/const.
Example : Arrow function component.

```
const Welcome = props => <h1>Hello,
{props.name}</h1>;
```

React Render HTML: Using ReactDOM to render HTML in the web browser.
Example : Rendering a component into the root div.

```
ReactDOM.render(<Welcome name="Sarah" />,
document.getElementById('root'));
```

React JSX: JavaScript syntax extension that looks similar to XML or HTML.
Example : JSX syntax.

```
const element = <h1>Welcome to React!</h1>;
```

React Components: Small, reusable pieces of code that return a React element to be rendered to the page.
Example : Functional component.

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

React Class: ES6 class components in React.
Example : Class component.

```
class Welcome extends
React.Component {
  render() {
    return <h1>Hello, {this.props.name}
</h1>;
  }
}
```

React Props: Way to pass data from parent to child components.
Example : Passing and accessing props.

```
<Welcome name="Sara" />
```

React Events: Handling user interactions in React.
Example : Handling a click event.

```
class Button extends React.Component {
  handleClick = () => {
    alert('Button clicked');
  };

  render() {
    return <button onClick={this.handleClick}>Click me</
button>;
  }
}
```

React Conditionals: Conditionally rendering components based on different logic.
Example : Using ternary operators for conditional rendering.

```
function Welcome(props) {
  return <div>{props.isLoggedIn ? <LogoutButton /> :
  <LoginButton />}</div>;
}
```

React Lists: Rendering lists of data using the map method.
Example : Rendering a list of items.

```
function NumberList(props) {
  const numbers = props.numbers;
  const listItems = numbers.map((number) =>
    <li key={number.toString()}>{number}</li>
  );
  return <ul>{listItems}</ul>;
}
```

React Forms: Handling form inputs and submissions.
Example : Controlled component for form inputs.

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) {
    this.setState({value: event.target.value});
  }
  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }
  render() {
    return (
      <form onSubmit={this.handleSubmit}>
        <label>
          Name:
          <input type="text" value={this.state.value}
  onChange={this.handleChange} />
        </label>
        <input type="submit" value="Submit" />
      </form>
    );
  }
}
```

React Router: Managing app routes and linking components.
Example : Basic routing.

```
import { BrowserRouter as Router, Route, Link } from "react-
router-dom";
function App() {
  return (
    <Router>
      <div>
        <nav>
          <ul>
            <li>
              <Link to="/">Home</Link>
            </li>
            <li>
              <Link to="/about">About</Link>
            </li>
            <li>
              <Link to="/users">Users</Link>
            </li>
          </ul>
        </nav>
        <Route path="/" exact component={Home} />
        <Route path="/about" component={About} />
        <Route path="/users" component={Users} />
      </div>
    </Router>
  );
}
```

React Memo: Optimizing performance by memorizing components.
Example : Using React.memo for a functional component.

```
const MyComponent = React.memo(function
MyComponent(props) {
  /* render using props */
});
```

React CSS Styling: Applying CSS styles in React components.
Example : Inline styling.

```
const divStyle = {
  color: 'blue',
  backgroundImage: 'url(' + imgUrl + ')',
};

function HelloWorldComponent() {
  return <div style={divStyle}>Hello World!</
div>;
}
```

React Sass Styling: Using Sass for styling in React projects.
Example : Importing a Sass stylesheet.

```
import './App.scss';
```

React Hooks

What is a Hook?: Hooks are functions that let you hook into React state and lifecycle features from function components.
useState: Manages state in a functional component.
Example :

```
const [age, setAge] = useState(42);
```

useEffect: Allows you to perform side effects in function components; it's used for data fetching, subscriptions, or manually modifying the DOM.
Example :

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
});
```

useContext: Provides a way to pass data through the component tree without having to pass props down manually at every level, ideal for maintaining themes, user preferences, and other global settings.
Example :

```
const theme = useContext(ThemeContext);
```

useRef: An alternative to useState, perfect for managing state logic that is more complex than a single value, such as handling multiple sub-values or when the next state depends on the previous one.
Example :

```
const [state, dispatch] = useReducer(reducer, initialState);
```

useCallback: Returns a memoized version of the callback that only changes if one of the dependencies has changed, useful for passing callbacks to optimized child components.
Example :

```
const memoizedCallback = useCallback(() => {
  doSomething(a, b);
}, [a, b]);
```

useMemo: Optimizes performance by memoizing expensive function results between renders, only recomputing the memoized value when a dependency changes.
Example :

```
const memoizedValue = useMemo(() =>
  computeExpensiveValue(a, b), [a, b]);
```

Custom Hooks: Building your own hooks to share reusable logic between components.
Example :

```
function useCustomHook() {
  const [value, setValue] = useState(null);
  // Logic here
  return value;
}
```

useActionState: Manages state that is updated based on the result of a form action. It's particularly useful with React Server Components.
Example :

```
const [state, formAction] = useActionState(actionFn, initialState);
```

useDebugValue: Adds a label to custom hooks in React DevTools, making it easier to identify hook state in complex components.
Example :

```
useDebugValue(value);
```

useDeferredValue: Creates a deferred version of a value that might change rapidly, like input values, delaying updates to the UI for better performance during transitions.
Example :

```
const deferredValue = useDeferredValue(value);
```

useId: Generates a unique ID that is stable across server and client, ensuring consistent IDs for server-rendered markup.
Example :

```
const componentId = useId();
```

useImperativeHandle: Customizes the instance value that is exposed to parent components when using refs.
Example :

```
useImperativeHandle(ref, () => ({
  focus: () => { /* focus logic */ }
}));
```

useInsertionEffect: Similar to useEffect, but it is specifically for styling and runs synchronously before all DOM mutations.
Example :

```
useInsertionEffect(() => {
  document.body.style.backgroun
dColor = 'blue';
});
```

useOptimistic: Allows you to optimistically update the UI before an asynchronous action completes.
Example :

```
const [optimisticState, setOptimisticState] = useOptimistic(initialState,
updateFn);
```

useSyncExternalStore: Subscribes to an external store and ensures the component updates when the store changes.
Example :

```
const state = useSyncExternalStore(store.subscribe, store.getState);
```

useTransition: Allows for state updates that do not block the user interface, marking updates as transitions that can be interrupted.

```
const [isPending, startTransition] = useTransition();
startTransition(() => {
  setProfileData(largeDataset);
});
```