React Cheat Sheet — React 19, for the web

Function components + Hooks. Full reference: https://react.dev

MOUNT AN APP

```
import { StrictMode } from 'react';
import { createRoot } from 'react-dom/client';
import App from './App.jsx';

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

SSR: hydrateRoot(domNode, <App />)

JSX RULES

- One root element; group with <>…</>
- className, htmlFor, camelCase events
- Close every tag:
- {expr} in children/attrs; {{…}} = object
- true/false/null/undefined render nothing

FUNCTION COMPONENT + PROPS

```
function Avatar({ user, size = 40, children }) {
  return (
    <figure>
      <img src={user.url} width={size} />
      {children}
    </figure>
  );
}
<Avatar user={u} size={64}>…</Avatar>
```

Props are read-only. Data flows down; callbacks flow up.

USESTATE

```
const [count, setCount] = useState(0);
setCount(count + 1);            // snapshot
setCount(c => c + 1);           // updater
setUser({ ...user, name: 'Ada' });      // obj
setItems([...items, next]);             // arr
setItems(items.filter(i => i.id !== id));
```

Never mutate state; make a new value.

USEREDUCER

```
function reducer(state, action) {
  switch (action.type) {
    case 'inc': return { n: state.n + 1 };
    default: throw Error(action.type);
  }
}
const [state, dispatch] = useReducer(reducer, { n: 0
});
dispatch({ type: 'inc' });
```

USEEFFECT + CLEANUP

```
useEffect(() => {
  const c = createConnection(roomId);
  c.connect();
  return () => c.disconnect();   // cleanup
}, [roomId]);                    // deps
```

For external systems only. Often you don't need an Effect — derive during render.

USECONTEXT

```
const ThemeCtx = createContext('light');

<ThemeCtx value="dark"><App /></ThemeCtx>   //
provider (R19)

const theme = useContext(ThemeCtx);
```

USEREF

```
const ref = useRef(null);
<input ref={ref} />
ref.current.focus();      // no re-render on change
```

R19: pass ref as a normal prop; forwardRef not needed.

MEMO / USEMEMO / USECALLBACK

```
const Row = memo(function Row({ item }) { … });
const sorted = useMemo(() => sort(items), [items]);
const onPick = useCallback(id => pick(id), [pick]);
```

The React Compiler auto-memoizes — hand memo rarely needed.

EVENTS

```
<button onClick={handleClick}>Go</button>   // pass
fn
<button onClick={() => add(id)}>Add</button>
function onSubmit(e) { e.preventDefault(); }
```

SyntheticEvent: e.target, e.key, e.stopPropagation().

LISTS & KEYS · CONDITIONALS

```
{todos.map(t => <li key={t.id}>{t.text}</li>)}
{isOn ? <On /> : <Off />}
{count > 0 && <Badge n={count} />}
if (!user) return null;
```

Keys: stable, from data — not the array index.

LAZY + SUSPENSE

```
const Settings = lazy(() =>
import('./Settings.jsx'));

<Suspense fallback={<Spinner />}>
  <Settings />
</Suspense>
```

Error boundary (class) handles the failed state.

CUSTOM HOOK

```
function useToggle(init = false) {
  const [on, setOn] = useState(init);
  const toggle = useCallback(() => setOn(v => !v),
[]);
  return [on, toggle];
}
```

Shares logic, not state. Name it use*.

RULES OF HOOKS / REACT

- Call Hooks at the **top level** only — no conditions/loops
- Call Hooks only from components or custom Hooks
- Components & Hooks must be **pure**; no side effects in render
- Props & state are immutable
- Render components as <X />, never X()

REACT 19: USE + ACTIONS

```
const value = use(promise);   // or use(Context); ok
in if/loops

<form action={formAction}>
const [state, formAction, pending] =
  useActionState(async (prev, fd) => { … }, init);

const { pending } = useFormStatus();      // react-
dom
const [opt, addOpt] = useOptimistic(value);
```