

**Hooked
on**

useState

hooks

```
import {  
  useState,      // store data as state  
  useEffect,    // "react" to state changes  
  useRef,       // reference data outside of state  
  useMemo,      // cache a value for performance  
  useCallback,  // cache a function for performance  
  useReducer,   // handle complex state updates  
} from 'react'
```

useState

Stores data as state

```
const [count, setCount] = useState(0)
```

```
const inc = () => setCount(count + 1)
```

```
const dec = () => setCount(count - 1)
```

```
// with callbacks:
```

```
const [count, setCount] = useState(() => 0)
```

```
const inc = () => setCount(current => current + 1)
```

```
const dec = () => setCount(current => current - 1)
```



useEffect

"React" to state changes

```
useEffect(() => {  
  // runs if either values change (including initial)  
}, [value, otherValue])
```

```
useEffect(() => {  
  // runs only once in the life of the component  
}, [])
```

```
useEffect(() => {  
  // handle value here  
  return () => { /* fn returned will be ran before the next effect */ }  
}, [value])
```



useRef

Reference data outside of state

```
const ref = useRef(null) // { current: null }  
  
<input ref={ref} /> // { current: <input> }  
  
ref.current = 3 // { current: 3 }
```



useMemo

Cache a value for performance

```
// expensiveFn is called every render  
const expensiveResult = expensiveFn(value)
```

```
// expensiveFn is only called when the value changes  
const memoizedResult = useMemo(() => expensiveFn(value), [value])
```



useCallback

Cache a function for performance

```
const handleClick = () => setCount(count + 1)
```

```
// harder to read with the function returning a function  
const handleClick = useMemo(() => () => setCount(count + 1), [count])
```

```
const handleClick = useCallback(() => setCount(count + 1), [count])
```



useReducer

Handle complex (or coupled) state

```
function reducer(state, action) {
  const { count, initialCount } = state
  switch (action.type) {
    case 'inc': return { count: count + 1, initialCount }
    case 'dec': return { count: count - 1, initialCount }
    case 'reset': return { count: state.initialCount, initialCount }
  }
}

// inside of component
const [state, dispatch] = useReducer(reducer, { count: initialCount, initialCount })
const inc = () => dispatch({ type: 'inc' })
const dec = () => dispatch({ type: 'dec' })
const reset = () => dispatch({ type: 'reset' })
```





Demo Time



[useState](#)



["the rest"](#)

Gabe Dayley

[@gmdayley](https://twitter.com/gmdayley)

Kyle West

[@KyleWestCS](https://twitter.com/KyleWestCS)

Slides may be found at:

