

## WHAT ARE REFS?

### REF WITH useRef

👉 "Box" (object) with a **mutable** `.current` property that is **persisted across renders** ("normal" variables are always reset)

```
const myRef = useRef(23);
```

myRef

ref stands for reference and it's like a box into which we can put any data that we want to be preserved between renders.

## WHAT ARE REFS?

### REF WITH useRef

👉 "Box" (object) with a **mutable** `.current` property that is **persisted across renders** ("normal" variables are always reset)

```
const myRef = useRef(23);
```

myRef

`.current`  
(23)

We can write to and read from the ref using `.current`

```
myRef.current = 1000;
```

Now, in technical terms, when we use `useRef`, React will give us an object with a mutable `current` property and we can then write any data into this `current` property and, of course, also read from it.

In this small example, the `current` property was first set to the initial value of 23 and we then changed it to 1000. So, as you can see, this `current` property is actually mutable, so unlike everything else in React.


refs are persisted across renders. So, their current property value stays the same between multiple renders, so just like state.

## WHAT ARE REFS?

### REF WITH `useRef`

- 👉 "Box" (object) with a **mutable** `.current` property that is **persisted across renders** ("normal" variables are always reset)
- 👉 Two big use cases:

```
const myRef = useRef(23);
```



The diagram shows a blue box labeled `myRef` containing a light blue box labeled `.current (1000)`. A red arrow points from the text below to the `.current` property.

We can write to and read from the ref using `.current`

```
myRef.current = 1000;
```


This gives us two big use cases for refs.

## WHAT ARE REFS?

### REF WITH `useRef`

- 👉 "Box" (object) with a **mutable** `.current` property that is **persisted across renders** ("normal" variables are always reset)
- 👉 Two big use cases:
  - 1 Creating a variable that stays the same between renders (e.g. previous state, `setTimeout` id, etc.)

```
const myRef = useRef(23);
```



The diagram shows a blue box labeled `myRef` containing a light blue box labeled `.current (1000)`. A red arrow points from the text below to the `.current` property.

We can write to and read from the ref using `.current`

```
myRef.current = 1000;
```

First, as we just said, we can use refs to create variables that will stay the same between renders e.g. preserving the previous state or storing the ID of a `setTimeout` function.

## WHAT ARE REFS?

### REF WITH useRef

👉 "Box" (object) with a **mutable** `.current` property that is **persisted across renders** ("normal" variables are always reset)

👉 Two big use cases:

- 1 Creating a variable that stays the same between renders (e.g. previous state, `setTimeout` id, etc.)
- 2 Selecting and storing DOM elements

```
const myRef = useRef(23);
```



We can write to and read from the ref using `.current`

```
myRef.current = 1000;
```

Scammy

The second use case is actually far more important, which is to select and store DOM elements.

A DOM element is also a piece of data that we want to store and preserve across renders and so refs are perfect for this.

## WHAT ARE REFS?

### REF WITH useRef

👉 "Box" (object) with a **mutable** `.current` property that is **persisted across renders** ("normal" variables are always reset)

👉 Two big use cases:

- 1 Creating a variable that stays the same between renders (e.g. previous state, `setTimeout` id, etc.)
- 2 Selecting and storing DOM elements

👉 Refs are for **data that is NOT rendered**: usually only appear in event handlers or effects, not in JSX (otherwise use state)

```
const myRef = useRef(23);
```



We can write to and read from the ref using `.current`

```
myRef.current = 1000;
```

Scammy

Now refs are usually for data that is not rendered individual output of the component. So, usually refs only appear in event handlers or effects, but not in the JSX.

We can use refs inside JSX too but usually that's not the place for them. If you need data that participates in the visual output of the component, that's usually a good sign that you actually need state and not a ref.

## WHAT ARE REFS?

### REF WITH `useRef`

- 👉 "Box" (object) with a **mutable** `.current` property that is **persisted across renders** ("normal" variables are always reset)
- 👉 Two big use cases:
  - 1 Creating a variable that stays the same between renders (e.g. previous state, `setTimeout` id, etc.)
  - 2 Selecting and storing DOM elements
- 👉 Refs are for **data that is NOT rendered**: usually only appear in event handlers or effects, not in JSX (otherwise use state)
- 👉 Do **NOT** read write or read `.current` in render logic (like state)

```
const myRef = useRef(23);
```

**myRef**

**.current**  
(1000)

We can write to and read from the ref using `.current`

```
myRef.current = 1000;
```

Just like with state, you are not allowed to write or to read the current property in render logic as that would create an undesirable side effect. Instead, we usually perform these mutations inside a `useEffect` hook.

## STATE VS. REFS

	PERSISTS ACROSS RENDERS	UPDATING CAUSES RE-RENDER	IMMUTABLE	ASYNCHRONOUS UPDATES
STATE	✓	✓	✓	✓
REFS	✓	✗	✗	✗

Need to store data  
(Taken from lecture "Fundamentals of state management")

Will data change at some point?

YES ↓

Should it re-render component?

YES ↓

State (`useState`)

NO → Regular const variable

NO → Ref (`useRef`)

Examples: preserving previous state, or storing the id of a `setTimeout`