SUNIL VISHWAKARMA
@linkinsunil

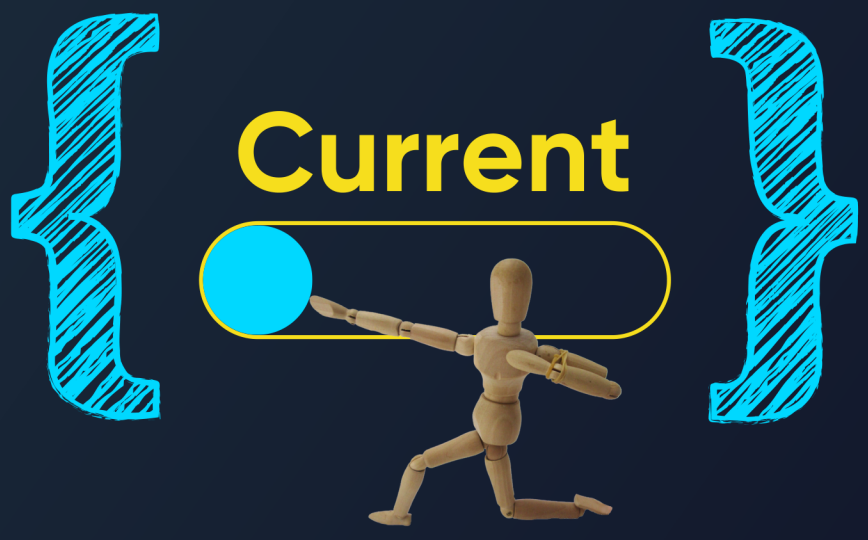
# useRef( )

## referencing values in React

When you want a component to remember some information, but you don't want that information to trigger new renders, you can use a ref.

## Lets See into

1. How to add a ref to component?
2. How to update a ref's value?
3. How refs are different from state?
4. When to use refs?
5. Best practices for using refs?




save it, like and share

swipe ⟶

# 1. Adding a ref to component

importing the useRef Hook from React:

```
import { useRef } from 'react';
```

Inside your component, call the useRef Hook and pass the initial value that you want to reference as the only argument. Here is a ref to the value 0:

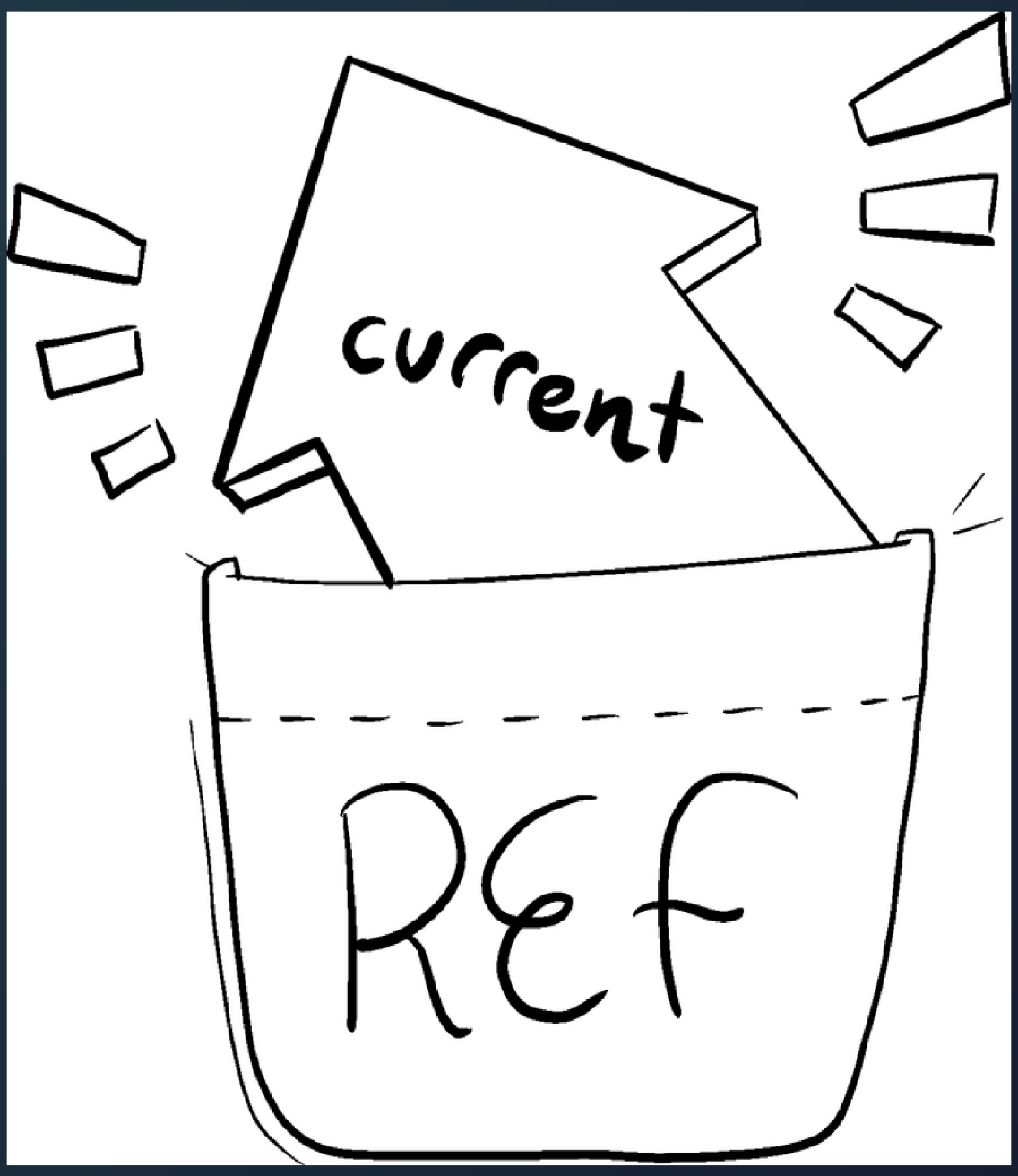
```
const ref = useRef(0);
```

useRef returns an object like this:

```
{
  current: 0 // The value you passed to useRef
}
```

You can access the current value of that ref through the ref.current property.

swipe ⟶




*Illustrated by Rachel Lee Nabors*

This current value of the ref, which stays in the ref.current property, is intentionally mutable, meaning you can both read and write to it. It's like a secret pocket of your component that React doesn't track. (This is what makes it an "escape hatch" from React's one-way data flow)

swipe ⟶

# 2. Updating a ref's value

Here, a button will increment ref.current on every click:

```
import { useRef } from 'react';

export default function Counter() {
  let ref = useRef(0);

  function handleClick() {
    ref.current = ref.current + 1;
    alert('You clicked ' + ref.current + ' times!');
  }

  return (
    <button onClick={handleClick}>
      Share this post!
    </button>
  );
}
```

Here, the ref points to a number, but, like state, you could point to anything: a string, an object, or even a function. Unlike state, ref is a plain JavaScript object with the current property that you can read and modify.

Note that the component doesn't re-render with every increment. Like state, refs are retained by React between re-renders. However, setting state re-renders a component. Changing a ref does not!

swipe ⟶

# Example: Building A Stopwatch

To display how much time has passed since the user pressed “Start”, you will need to keep track of when the Start button was pressed and what the current time is.

- This information is used for rendering, so you’ll keep it in state.
- When the user presses “Start”, you’ll use setInterval in order to update the time every 10 milliseconds.
- When the “Stop” button is pressed, you need to cancel the existing interval so that it stops updating the now state variable. You can do this by calling clearInterval, but you need to give it the interval ID that was previously returned by the setInterval call when the user pressed Start. You need to keep the interval ID somewhere. Since the interval ID is not used for rendering, you can keep it in a ref. See the code ahead.

When a piece of information is used for rendering, keep it in state. When a piece of information is only needed by event handlers and changing it doesn’t require a re-render, using a ref may be more efficient.

swipe ⟶

SUNIL VISHWAKARMA
@linkinsunil



# A Stopwatch

```jsx
import { useState, useRef } from 'react';

export default function Stopwatch() {
  const [startTime, setStartTime] = useState(null);
  const [now, setNow] = useState(null);
  const intervalRef = useRef(null);

  function handleStart() {
    // Start counting.
    setStartTime(Date.now());
    setNow(Date.now());

    clearInterval(intervalRef.current);
    intervalRef.current = setInterval(() => {
      // Update the current time every 10ms.
      setNow(Date.now());
    }, 10);
  }

  function handleStop() {
    clearInterval(intervalRef.current);
  }

  let secondsPassed = 0;
  if (startTime != null && now != null) {
    secondsPassed = (now - startTime) / 1000;
  }

  return (
    <>
      <h1>Time passed: {secondsPassed.toFixed(3)}</h1>
      <button onClick={handleStart}>
        Start
      </button>
      <button onClick={handleStop}>
        Stop
      </button>
    </>
  );
}
```

swipe ⟶

# 3. refs vs state

| refs | state |
| --- | --- |
| useRef(initialValue) returns { current: initialValue } | useState(initialValue) returns the current value of a state variable and a state setter function ( [value, setValue]) |
| Doesn't trigger re-render when you change it. | Triggers re-render when you change it. |
| Mutable—you can modify and update current's value outside of the rendering process. | "Immutable"—you must use the state setting function to modify state variables to queue a re-render. |
| You shouldn't read (or write) the current value during rendering. | You can read state at any time. However, each render has its own snapshot of state which does not change. |

swipe ⟶

# 4. When to use refs

You will use a ref when your component needs to step outside React and communicate with external APIs—often a browser API that won't impact the appearance of the component. Here are a few of these rare situations:

- Storing timeout IDs
- Storing and manipulating DOM elements (stay tuned for a separate post for this one)
- Storing other objects that aren't necessary to calculate the JSX.

If your component needs to store some value, but it doesn't impact the rendering logic, choose refs.

swipe ⟶

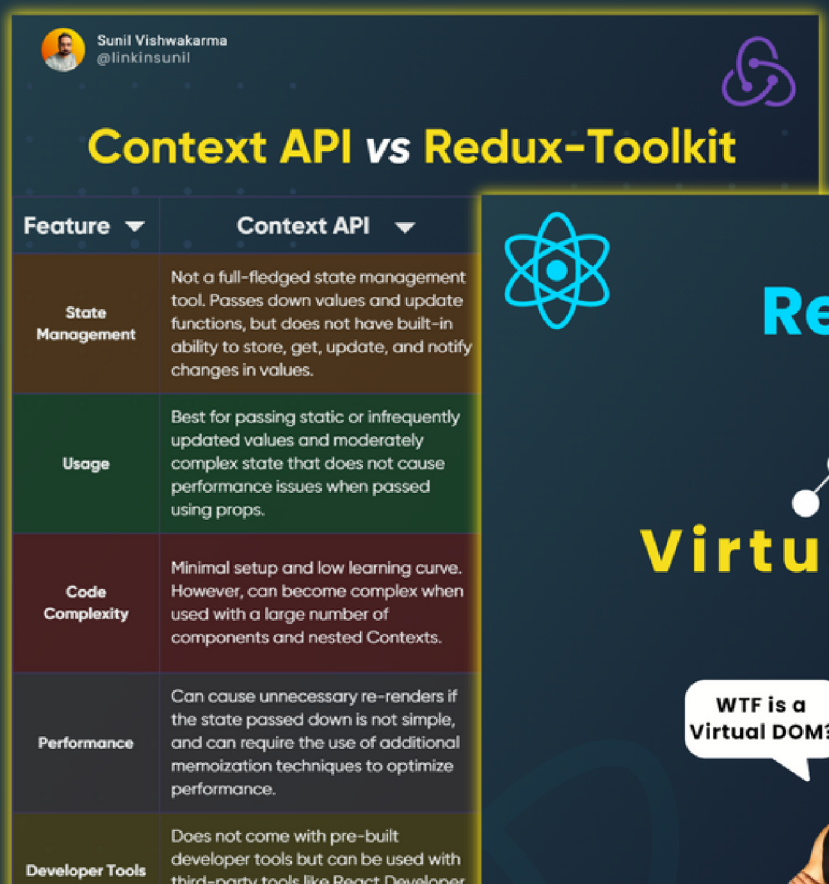
# 5. Best practices for refs

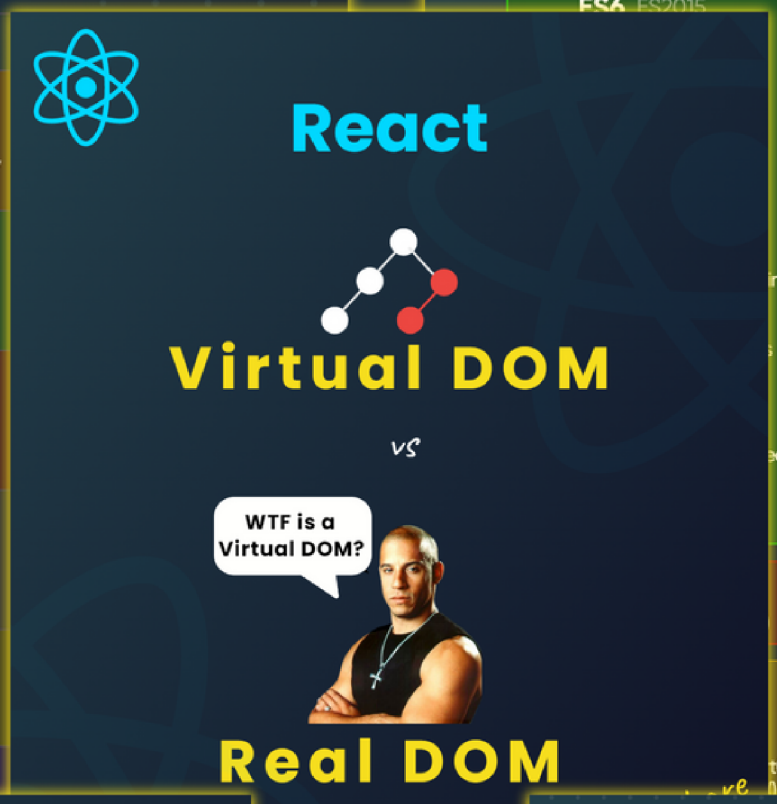
- Treat refs as an escape hatch. Refs are useful when you work with external systems or browser APIs. If much of your application logic and data flow relies on refs, you might want to rethink your approach.
- Don’t read or write ref.current during rendering. If some information is needed during rendering, use state instead. Since React doesn’t know when ref.current changes, even reading it while rendering makes your component’s behavior difficult to predict. (The only exception to this is code like if (!ref.current) ref.current = new Thing() which only sets the ref once during the first render.)
- Limitations of React state don’t apply to refs. For example, state acts like a snapshot for every render and doesn’t update synchronously. But when you mutate the current value of a ref, it changes immediately. This is because the ref itself is a regular JavaScript object, and so it behaves like one.
- No need to worry about avoiding mutation when you work with a ref. As long as the object you’re mutating isn’t used for rendering, React doesn’t care what you do with the ref or its contents.

swipe ⟶

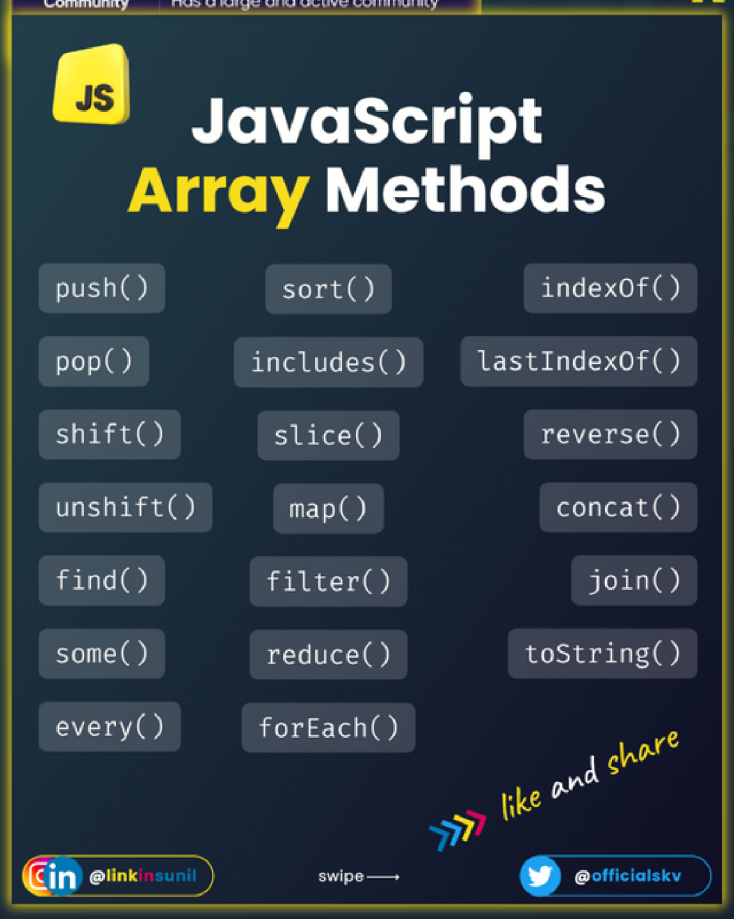
# Thats a Wrap!

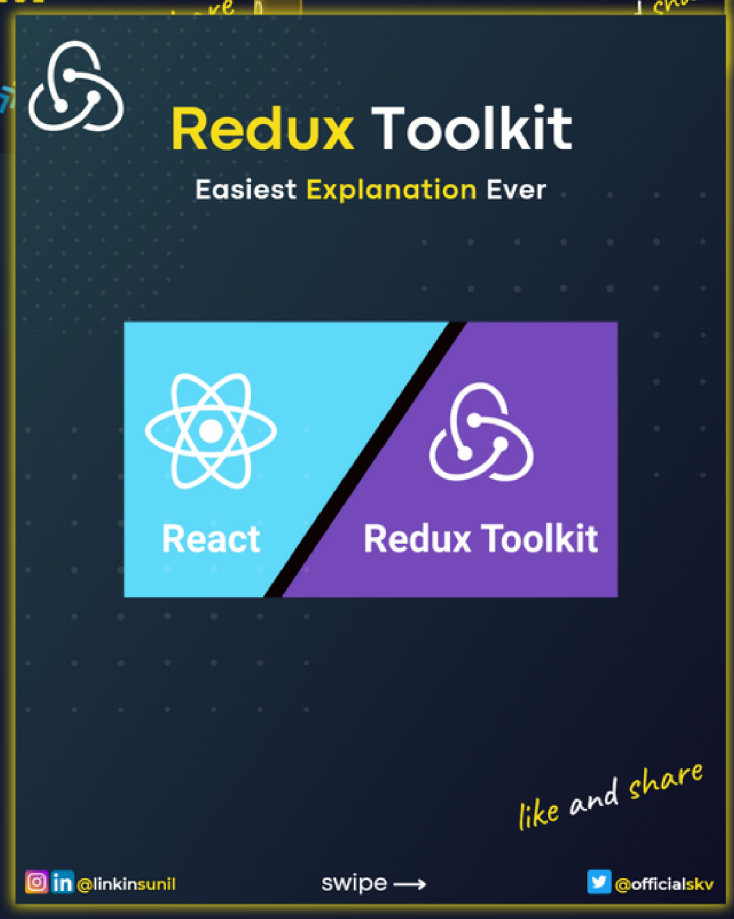
## If you liked it, checkout my other posts








## and many more short and easy explanations

P.S.

Repost this if you think your followers will like it

# Enjoyed this?

1. Follow me
2. Click the 🔔 notification
3. Never miss a post