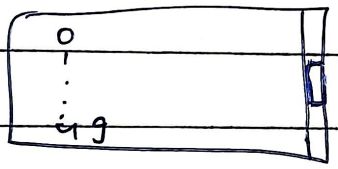
Frontend Infinite Scroll over whole window

→ Unlike normal events scroll/resize don't bubble up & they are window obj (Global) events.

→ Suppose we have a state count (init → 50) & we create a derived state e number [ ] by running a ~~loop~~ count times & display {e} in JSX.

↳ initially 0 1 2 . . . . 49 will display in a column.



→ Now for window specific events like 'scroll' we declare them & delete them on unmount in UseEffect

```
UseEffect ( () => {
  const onScroll = () => {...}

  window.addEventListener ('scroll', onScroll)

  return () => window.removeEventListen ('scroll', onScroll)

}, [ ])
```

→ now inside fn onScroll :

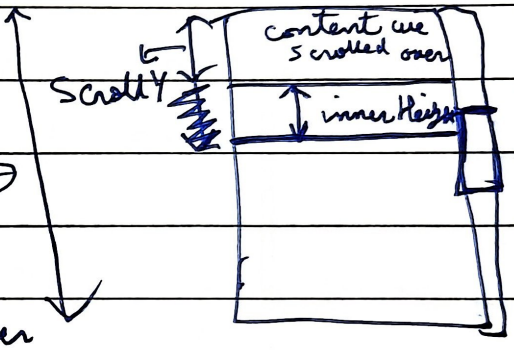
```
if (window.scrollY + window.innerHeight >=
    window.document.body.offsetHeight - 40) {
                                  ↳ buffer in px
  setCount(~~count +~~ prev => prev + 50)
}
```

→ window.scrollY

→ window.innerHeight

→ windo.docum.body.offsetHeight

→ At all times only innerHeight is visible to user



NOTE! → u can also write OO scroll for a scrollable div using useRef(attach to that div) & inside useEffect instead of window :

```
const container = divRef.current;
container.add----
return () => container.remove----
```

& inside onScroll again do ↖ & using container u can get scrollY etc. (with different names).

NOTE:2 ⇒ Also for performance optimizations we can use * debounce & * throttle & React.Memo. (to not do redundant rerender)

→ Using both debounce & throttle we aim to minimize on Scroll triggers/runs.

1) throttle ⇒ decide an interval/delay & in that interval only 1 time we do something.

2) debounce ⇒ we do something only once at end of some delay & everytime some event try to do that before delay's end we disable previous setTimeout & recreate one from this moment.

(for code see chatGpt).

↓

Here also note that ask for non 'this' arrow fⁿ based. For class based code "this" aware implementation is used (Read about that as well)