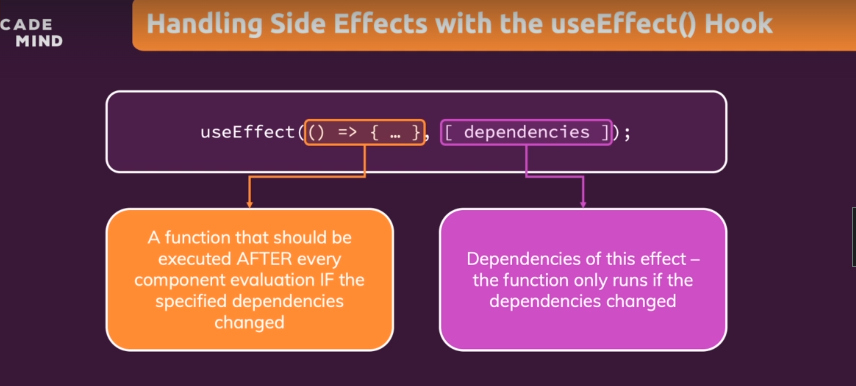
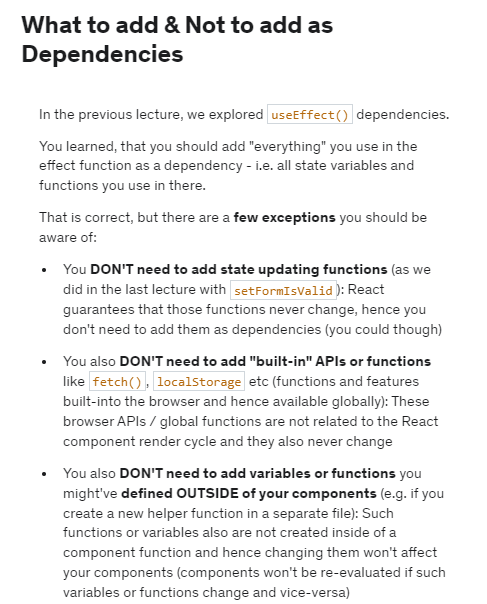
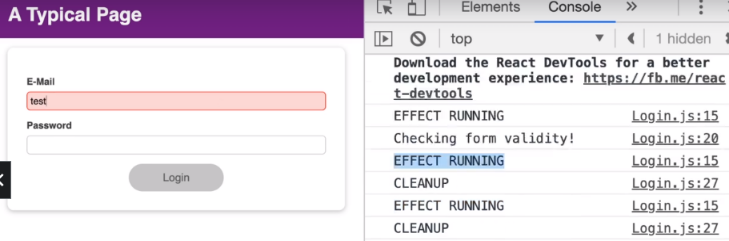
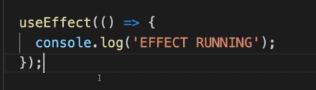
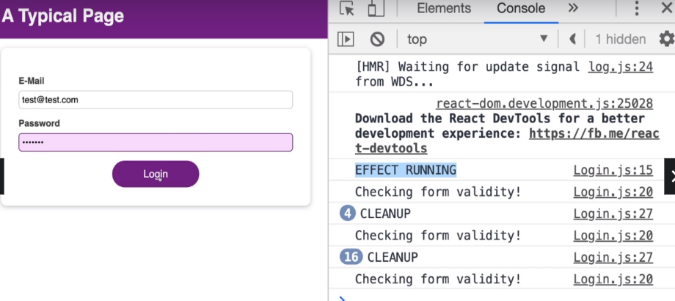
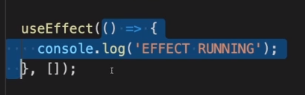
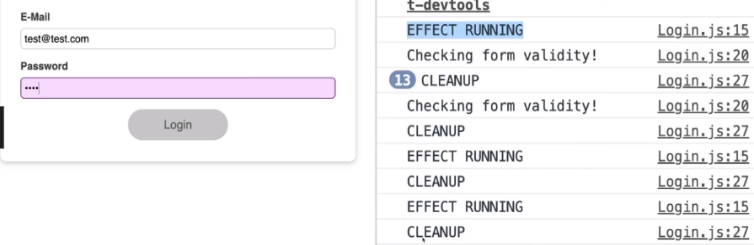
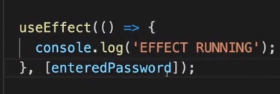
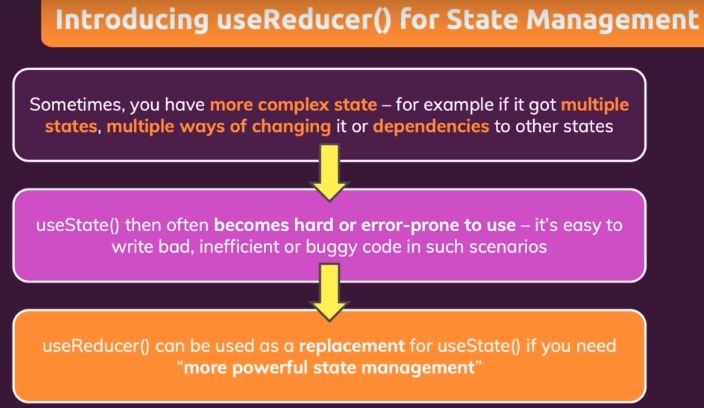
**Side Effect, reducer and app wide component**

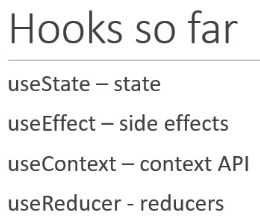
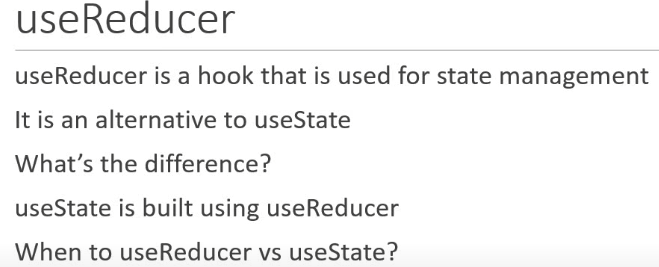
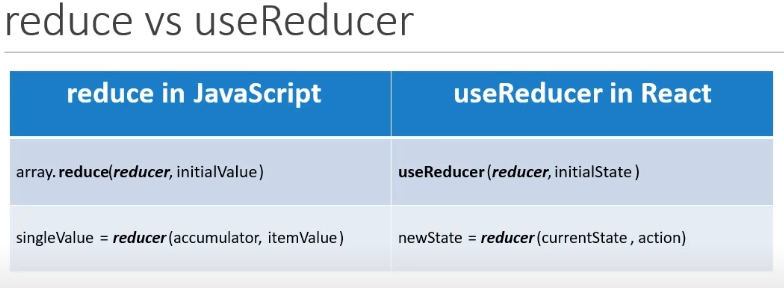
* The useEffect Hook allows you to perform side effects in your components. Some examples of side effects are: fetching data, directly updating the DOM, and timers. useEffect accepts two arguments. The second argument is optional. **useEffect(<function>, <dependency>).**
* **Main job: Render UI and make it responsive to user input.**
  + Evaluate and Render JSX, Manage state and props, react to ser events and input. Reevaluate component on state and prop changes.
  + UseState, Props, lift the state etc.
* **Other Job: Everything Else**
  + Ex – sending or receiving http requests, timer managements, storing cookies etc.
  + These need to happen outside of the normal component functioning.
  + Why? Because whenever the state changes, the react function reruns, and renders it again if there is any DOM difference.
  + **Why not?** Well, so you send a request using a main state function, then based on the response of the request, you would be changing some output on website, which will again cause a change in state, then it would again cause an http request to fetch new stuff, so causing an infinite loop.
  + **useEffect hook:** Another built in hook, which does something special.
* **UseEffect Hook:**
  + ****
  + A function that should execute only when the dependencies change. Your side effect code goes into the orange part with arguments and function body, and the other array contains dependencies.
  + ****
* **Debouncing:**
  + We wait for some time, once the user makes a pause for some time. We use setTimeout(arrowFunction, timeout duration in ms) function to check after delay of 500ms for each event like a keystroke.
  + We clear it after each key stroke, and only use the last keystroke whose timer is cleared at end.<-- through useEffect.
* **Returning using UseEfect**
* useEffect(() => {
* first
* return () => {
* second
* }
* }, [third])
* **Before the “first” runs except for the first time, the return function runs each time before sideffect function execution and when the component is removed.. It is returned inside the first function in useEffect.** This return function is called a **CLEANUP FUNCTION.**

**useEffectSummary:**

1. After every component (in which it is declared) render cycle, it changes and executes the useffect including the first time.
2. When we add an empty array – which means no dependency, so it runs only once when the component is mounted. Only clean up will run not the sideEffect function.  
   
3. Alternatively if you have a dependency, whenever the evaluated dependency is changing the effect will run for the related dependency render cycle.  
      
   Effect running shows up only once we start entering the password field because enteredPassword is a dependency. Before it, the sideEffect runs only once, and then clean up runs multiple times. But not sideEffect function..
4. Clean up function will always run when the component is unmounted

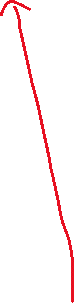
**useReducer():** [**DEMYSTIFIED HERE**](https://dev.to/dustinmyers/what-even-is-a-dispatch-function-27ma)

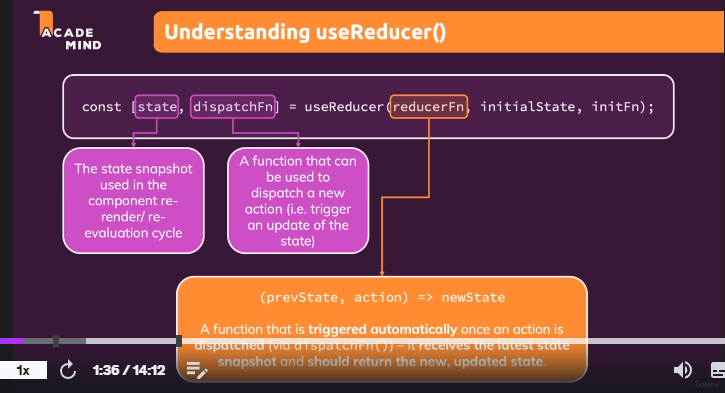




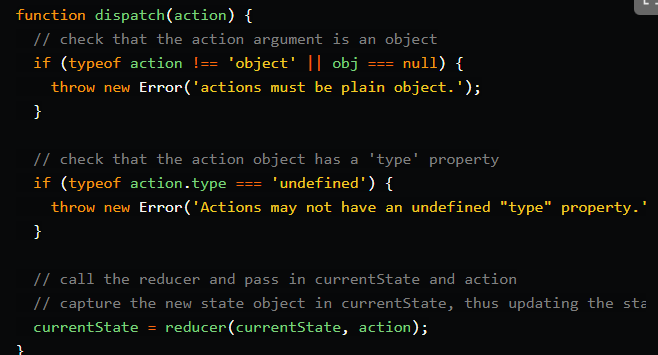
**CODE EXAMPLE :**

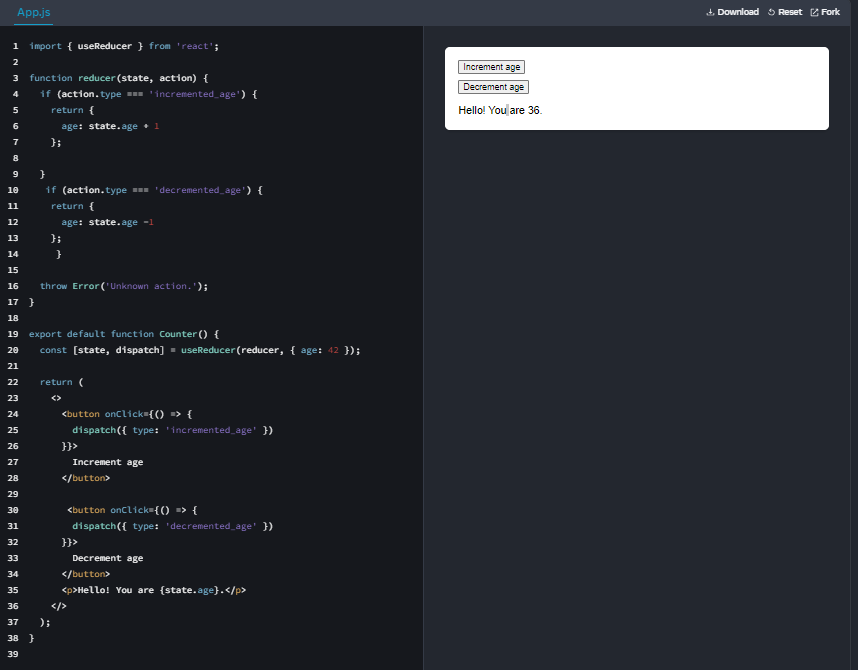




Dispatch function is like setStateValue function of useState, the dispatch function returns the action value for the same reducer function we define.Basically it requires us to define the reducer function with 2 values.

**Reverse Engineer the dispatch function:**

****

It calls the reducer function in it, with the current state, and the action/more data that we pass along with it.

**IMMUTABILITY PRINCIPLE:**

