**React.js**

1. React uses your keys to know what happened if you later insert, delete, or reorder the items.
2. When a function is declared in a *useEffect(()=>{handleClick},[])* that function is now a stale variable.
3. A stale variable is when the variable points to the memory address of pre-render value, which is containing old value and if the value is changed the new value acquire new memory address but the stale variable(*handleClick()*) points to old memory and return old value.
4. In JS functions are treated as values like string or number when allotting memory space.
5. cleanUp function of hooks() is called every re-render(in first render React saves the reference of cleanUp function and call just before second render of hooks()) just before the hook which is now obligated to re-render with the component, makes the hook run as new.
6. **Learning useEffect()**
7. After your component is removed from the DOM, React will run your cleanup function.
8. React will compare each dependency with its previous value using the Object.is comparison.
9. useEffect returns undefined.
10. If some of your dependencies are objects or functions defined inside the component, there is a risk that they will cause the Effect to re-run more often than needed.
11. If your Effect is doing something visual (for example, positioning a tooltip even you want to re-paint the screen after then a click will trigger the useEffect), and the delay is noticeable (for example, it flickers), replace useEffect with **useLayoutEffect.**
12. **Race condition (**a condition which is caused when multiple request with unknown behaviour/irregular pattern made the data which is been request via fetch in useEffect() go random**)** in useEffect is caused when one or more type of request is send to process before the result came. It can be addressed with *const abContr = new* ***AbortController()*** in react or using a Boolean value or to only show the last result of the request (the flaw in this method is that it is showing only one result the race condition may still be there).
13. Correct way to update state *setCount(prevCount => prevCount + 1);*
14. **Learning useCallback()**
15. By wrapping function like handleSubmit and etc. in useCallback, you ensure that it’s the same function between the re-renders (until dependencies change).
16. useMemo() is used to make a component stop re-render when the props passed inside it remains same.
17. useMemo caches the result of calling your function.
18. useCallback caches the function itself.
19. A prop that is always different would defeat the point of memoization.
20. There are some events that browsers offers named as **native browser API**. These are the event which get driggers when you use them as attributes of a tag ex: dragstart, dragover, and drop… but when using any other framework or react and if you want to use any event which is provided by native browser API, u need to use the wrapped version of the event. Example: dragstart(native browser API) 🡪 onDragStart(for frameworks) and etc..
21. Today I partially made a component that call itself when importing nested values and those nested values having similar properties than the parent one.
22. ss